About error and warning messages

Messages are displayed with the message class first, followed by the source file name and line number where the error was detected, and finally with the text of the message itself.

The following categories of messages can occur:

Category	Indicates
Informational	Progress such as build status.
Error	A problem that should be fixed such as a missing declaration or a type mismatch.
Warning	A problem that can be overlooked.
Fatal	A problem of critical nature that prevents execution from continuing.

Be aware that the compiler generates messages as they are detected. Because C and C++ don't force any restrictions on placing statements on a line of text, the true cause of the error might occur one or more lines before or after the line number specified in the error message.

Most of the messages appear in the Message view in the IDE. For those messages, context-sensitive help is available. Point to the message and press F1 to display the message description. You can choose to display or suppress compiler and linker warning messages from the Compiler and Linker pages of the Project Options dialog box (displayed using Project|Options). You can also choose to display informational messages in the Message view while compiling and display extended error information by selecting options on the Compiler page of the Project Options dialog box.

If you are working from the command line or want to look up information on an error message, refer to the alphabetical list of Error and warning messages. Find the message you're interested in and click on it to display its description. You can also look up Resource Linker messages or MAKE errors if you are using these additional tools.

Symbols

Some messages include a symbol (such as a variable, file name, or module) that is taken from your program. In the following example, 'filename' is replaced by the name of the file causing the problem:

```
Error opening filename for output
```

There are many symbols used within the messages and usually they are self-explanatory. Here's a list of some of the symbols and what they mean:

Symbol	Meaning
address	A hexadecimal number indicating the address where the error occurred
argument	An argument
class	A class name
filename	A file name (with or without extension)
function	A function name
group	A group name
identifier	An identifier (variable name or other)
language	The name of a programming language
member	The name of a data member or member function
message	A message string
module	A module name
name	Any type of name
num, number	An actual number
option	An option
parameter	A parameter name
path	A path name

reason	Reason given in message
segment	A segment name
size	An actual number
specifier	A type specifier
symbol	A symbol name
type	A type name
variable	A program variable
unitname	A source file and its associated declarations

Some messages begin with a symbol name such as the following:

```
'filename' not found
```

These messages are listed alphabetically using the name of the symbol. The above message would be filed under f.

Compiler messages and warnings begin with numbers. These are filed alphabetically using the message text after the number.

Compiler errors and warnings

Compile-time error messages indicate errors in program syntax, command-line errors, or errors in accessing a disk or memory. When most compile-time errors occurs, the compiler completes the current phase (preprocessing, parsing, optimizing, and code-generating) of the compilation and stops. But when fatal compile-time errors happen, compilation stops completely. If a fatal error occurs, fix the error and recompile.

Compiler errors include 4-digit error numbers beginning with E for general compiler errors, F for fatal errors, and W for compiler warnings.

Warnings indicate that conditions which are suspicious but legitimate exist or that machine-dependent constructs exist in your source files. Warnings do not stop compilation.

Warnings are issued as a result of a variety of conditions, such as:

ANSI violations	Warn you of code that is acceptable to C++Builder (because of C++ code or C++Builder extensions), but is not in the ANSI definition of C.
Frequent warnings	Alert you to common programming mistakes. These warning messages point out conditions that are not in violation of the C++Builder language but can yield the wrong result.
Less frequent warnings	Alert you to less common programming mistakes. These warning messages point out conditions that are not in violation of the C++Builder language but can yield the wrong result.
Portability warnings	Alert you to possible problems with porting your code to other compilers. These usually apply to C++Builder extensions.
C++ warnings	Warn you of errors you've made in your C++ code. They might be due to obsolete items or incorrect syntax.

Runtime errors and warnings

Runtime errors occur after the program has successfully compiled and is running. Runtime errors are usually caused by logic errors in your program code. If you receive a runtime error, you must fix the error in your source code and recompile the program for the fix to take effect.

Linker errors and warnings

Linker errors occur during while the incremental linker is executing. A fatal link error stops the linker immediately. In such a case, the .EXE file is deleted. All linker errors are treated as fatal errors if you are compiling from the Integrated Development Environment (IDE) or from the command line.

When compiling from the command line,it is possible to retain the .EXE, .MAP, and TDS files using the -

Gk.

Librarian errors and warnings

Librarian errors and warnings occur when there is a problem with files or extended dictionaries, when memory runs low, or when there are problems as libraries are accessed.

Error and warning messages

Overview

{button Symbols,JI(`',`errorsxref_symbols')} {button a,JI(`',`errorsxref_A')} {button b,JI(`',`errorsxref_B')} {button c,JI(`',`errorsxref_C')} {button d,JI(`',`errorsxref_D')} {button e,JI(`',`errorsxref_E')} {button f,JI(`',`errorsxref_F')} {button g,JI(`',`errorsxref_G')} {button h,JI(`',`errorsxref_H')} {button i,JI(`',`errorsxref_I')} {button l,JI(`',`errorsxref_L')} {button m,JI(`',`errorsxref_M')} {button n,JI(`',`errorsxref_N')} {button o,JI(`',`errorsxref_O')} {button p,JI(`',`errorsxref_P')} {button q,JI(`',`errorsxref_Q')} {button r,JI(`',`errorsxref_R')} {button s,JI(`',`errorsxref_S')} {button t,JI(`',`errorsxref_T')} {button u,JI(`',`errorsxref_U')} {button v,JI(`',`errorsxref_V')} {button w,JI(`',`errorsxref_W')}

Symbols

#undef directive ignored—Compiler warning

(expected—Compiler error

) expected—Compiler error

, expected—Compiler error

: expected after private—Compiler error

: expected after protected—Compiler error

: expected after public—Compiler error

< expected—Compiler error

> expected—Compiler error

@ seen, expected a response-files name—Librarian error

{ expected—Compiler error

} expected—Compiler error

A

Abnormal program termination—Runtime error

Access can only be changed to public or protected—Compiler error

Access specifier of property identifier must be a member of function—Compiler error

Access violation. Link terminated—Linker error

Added file 'filename' does not begin correctly, ignored—Librarian warning

Additional segments need to be defined in a .DEF file—Linker error

Ambiguity between 'function1' and 'function2'—Compiler error

Ambiguous member name 'name'—Compiler error

Ambiguous operators need parentheses—Compiler warning

Ambiguous override of virtual base member 'function1': 'function2'—Compiler error

Application is running—Runtime error

Argument kind mismatch in redeclaration of template parameter 'parameter' (E2422)—Compiler error

Argument list of specialization cannot be identical to the parameter list of primary template (E2388)—Compiler error

Array allocated using 'new' may not have an initializer—Compiler error

Array bounds missing]—Compiler error

Array dimension 'specifier' could not be determined (E2483)—Compiler error

Array must have at least one element—Compiler error

Array of references is not allowed—Compiler error

Array size for 'delete' ignored —Compiler warning or error

Array size too large—Compiler error

Array variable 'identifier' is near—Compiler warning

Assembler stack overflow—Compiler error

B

C

Cannot cast from 'type1' to 'type2'—Compiler error
Cannot convert 'type1' to 'type2'—Compiler error
Cannot create instance of abstract class 'class'—Compiler error
Cannot create precompiled header: 'reason'—Compiler error
Cannot declare a member function via instantiation (E2472)—Compiler error
Cannot declare or define 'identifier' here—Compiler error
Cannot define a pointer or reference to a reference—Compiler error
Cannot define 'identifier' using a namespace alias—Compiler error
Cannot delay load of module—Linker error
Cannot find a valid specialization for 'specifier' (E2409)—Compiler error
Cannot find 'class'::operator=('class'&) to copy a vector—Compiler error
Cannot find class::class (class &) to copy a vector—Compiler error
Cannot find default constructor to initialize array element of type 'class'—Compiler error
Cannot find default constructor to initialize base class 'class'—Compiler error
Cannot find default constructor to initialize member 'identifier'—Compiler error
Cannot find MAKE.EXE—MAKE error
Cannot find tasm program: tasm.exe—IDE error
Cannot generate 'function' from template function 'template'—Compiler error
Cannot generate specialization from 'specifier' because that type is not yet defined (E2440)—Compiler error
Cannot generate template specialization from 'specifier (E2299)'—Compiler error
Cannot have a non-inline function in a local class—Compiler error
Cannot have a static data in a local class—Compiler error
Cannot have both a template class and function named 'name' (E2479)—Compiler error
Cannot have multiple paths for implicit rule—MAKE error
Cannot have path list for target—MAKE error
Cannot inherit non-RTTI class from RTTI base—Compiler error
Cannot initialize 'type1' with 'type2'—Compiler error
Cannot initialize a class member here—Compiler error
Cannot initialize a class member here—Compiler error
Cannot involve parameter 'parameter' in a complex partial specialization expression (E2386)—Compiler error
Cannot involve template parameters in complex partial specialization arguments (E2480)—Compiler error
Cannot Load Linker: linker—IDE error
Cannot modify a const object—Compiler error
Cannot overload ‘function’ (E2476)—Compiler error
Cannot overload 'main'—Compiler error
Cannot override a 'dynamic/virtual' with a 'dynamic/virtual' function—Compiler error
Cannot reference template argument 'arg' in template class 'class' this way (E2399)—Compiler error
Cannot release virtual memory at addr 'address' for n bytes (errcode errnumber)—Linker error
Cannot reserve virtual memory at addr 'address' for n bytes (errcode errnumber)—Linker error
Cannot specify default function arguments for explicit specializations (E2287)—Compiler error
Cannot take address of 'main'—Compiler error
Cannot take the address of non-type, non-reference template parameter 'parameter' (E2393)—Compiler error
Cannot throw 'type' -- ambiguous base class 'base'—Compiler error
Cannot use address of array element as non-type template argument (E2285)—Compiler error
Cannot use address of class member as non-type template argument (E2286)—Compiler error
Cannot use local type 'identifier' as template argument—Compiler error
Cannot use template 'template' without specifying specialization parameters (E2102)—Compiler error

Cannot use templates in closure arguments -- use a typedef—Compiler error
Cannot use the result of a property assignment as an rvalue'—Compiler error
Cannot write a string option—MAKE error
Cannot write GRPDEF list, extended dictionary aborted—Librarian warning
Cannot write to disk—Linker error
Case bypasses initialization of a local variable—Compiler error
Case outside of switch—Compiler error
Case statement missing :—Compiler error
'catch' expected —Compiler error
Character constant must be one or two characters long—Compiler error
Character constant too long—MAKE error
Circular dependency exists in makefile—MAKE error
Circular property definition :—Compiler error
Class 'class' may not contain pure functions—Compiler error
Class 'classname' is abstract because of 'member = 0'—Compiler error
'class' is not abstract public single inheritance class hierarchy with no data (E2246)—Compiler error
'class' is not a direct base class of 'class' (E2507)—Compiler error
Class member 'member' declared outside its class—Compiler error
Class type 'type' cannot be marked as __declspec(delphireturn)—Compiler error
Classes with properties cannot be copied by value—Compiler error
Classid requires definition of 'type' as a pointer type—Compiler error
Code has no effect—Compiler warning
Colon expected—MAKE error
Comma not allowed here: 'location'—Linker error
Command arguments too long—MAKE error
Command syntax error—MAKE error
Comparing signed and unsigned values—Compiler warning
Compiler could not generate copy constructor for class 'class'—Compiler error
Compiler could not generate default constructor for class 'class'—Compiler error
Compiler could not generate default destructor for class 'class'—Compiler error
Compiler could not generate operator = for class 'class'—Compiler error
Compiler stack overflow—Compiler error
Compiler table limit exceeded—Compiler error
Compound statement missing }—Compiler error
Condition is always false—Compiler warning
Condition is always true—Compiler warning
Conflicting type modifiers—Compiler error
Constant expression required—Compiler error
Constant is long—Compiler warning
Constant member 'member' in class without constructors—Compiler error
Constant member 'member' is not initialized—Compiler warning
Constant out of range in comparison—Compiler warning
Constant variable 'variable' must be initialized—Compiler error
Constructor cannot be declared 'const' or 'volatile'—Compiler error
Constructor cannot have a return type specification—Compiler error
'constructor' is not an unambiguous base class of 'class'—Compiler error
Constructor initializer list ignored—Compiler warning
Constructors and destructors not allowed in __automated section—Compiler error
Continuation character \ found in // comment—Compiler warning

Conversion may lose significant digits—Compiler warning
Conversion operator cannot have a return type specification—Compiler error
Conversion to 'type' will fail for members of virtual base 'class'—Compiler warning
Conversions of class to itself or base class not allowed—Compiler error
Could not allocate memory for per module data—Librarian error
Could not create 'filename' (error code 'number')—Linker error
Could not create list file 'filename'—Librarian error
Could not delete ‘item’ (project already open in IDE?)—Linker error
Could not find a 'main unit'; initialization order will follow link order—Linker error
Could not find a match for argument(s)—Compiler error
Could not find file 'filename'—Compiler error
Could not get procedure address from RLINK32.DLL—Linker error
Could not load RLINK32.DLL—Linker error
Could not open 'filename' (error code 'number')—Linker error
Could not open 'filename' (program still running?)—Linker error
Could not open 'filename' (project already open in IDE?)—Linker error
Could not strip resources from ‘target’—Linker error
Could not write output—Librarian error
Could not write to 'filename' (error code 'number')—Linker error
Couldn't get LE/LIDATA record buffer—Librarian error
Creating a package without any units—Linker warning
Cycle in include files: 'filename'—MAKE error

D

Data member definition not allowed in __automated section—Compiler error
Declaration does not specify a tag or an identifier—Compiler error
Declaration ignored—Compiler warning
Declaration is not allowed here—Compiler error
Declaration missing ;—Compiler error
Declaration of member function default parameters after a specialization has already been expanded (E2411)—Compiler error
Declaration of static function '(...)' ignored—Compiler warning
Declaration syntax error—Compiler error
Declaration terminated incorrectly—Compiler error
Declaration was expected—Compiler error
Declare operator delete (void*) or (void*, size_t)—Compiler error
Declare operator delete[] (void*) or (void*, size_t)—Compiler error
Declare type 'type' prior to use in prototype—Compiler warning
__declspec(delphireturn) class 'class' must have exactly one data member—Compiler error
__declspec(selectany) is only for initialized and externally visible variables (E2500)—Compiler error
Default argument value redeclared—Compiler error
Default argument value redeclared for parameter 'parameter'—Compiler error
Default expression may not use local variables—Compiler error
Default outside of switch—Compiler error
Default type for template template argument 'arg' does not name a primary template class (E2436)—Compiler error
Default value missing—Compiler error
Default value missing following parameter 'parameter'—Compiler error
Default values may be specified only in primary class template declarations (E2408)—Compiler error
Define directive needs an identifier—Compiler error

Delayed load module ‘module’ was not found—Linker error
Delete array size missing]—Compiler error
Deleting an object requires exactly one conversion to pointer operator—Compiler error
Dependent call specifier yields non-function 'name' (E2403)—Compiler error
Dependent template reference 'identifier' yields non-template symbol (E2405)—Compiler error
Dependent type qualifier 'qualifier' has no member symbol named 'name' (E2407)—Compiler error
Dependent type qualifier 'qualifier' has no member type named 'name' (E2404)—Compiler error
Dependent type qualifier 'qualifier' is not a class or struct type (E2406)—Compiler error
Destructor cannot be declared 'const' or 'volatile'—Compiler error
Destructor cannot have a return type specification—Compiler error
Destructor for 'class' required in conditional expression—Compiler error
Destructor for class is not accessible—Compiler error
Destructor name must match the class name—Compiler error
Destructors cannot be declared as template functions (E2414)—Compiler error
Dispid ‘number’ already used by ‘identifier’ Divide error—Compiler error
Dispid only allowed in __automated sections—Compiler error
Divide error—Runtime error
Division by zero—Compiler error
Division by zero—Compiler warning
Division by zero—MAKE error
do statement must have while—Compiler error
do-while statement missing (—Compiler error
do-while statement missing)—Compiler error
do-while statement missing ;—Compiler error
Duplicate case—Compiler error
Duplicate file ' filename' in list, not added!—Librarian error
Duplicate handler for 'type1', already had 'type2'—Compiler error
‘ dynamic’ can only be used with non-template member functions (E2504)— Compiler error
Dynamic function’function’ conflicts with base class ‘class’’— Compiler error

E

Earlier declaration of 'identifier'—Compiler error
Enum syntax error—Compiler error
Error changing file buffer size—Librarian error
Error directive: 'message'—Compiler error
Error directive: 'message'—MAKE error
Error opening 'filename'—Librarian error
Error opening 'filename' for output—Librarian error
Error processing module 'module'—Linker error
Error renaming 'filename' to 'filename'—Librarian error
Error resolving #import: problem (E2502)—Compiler error
Error while instantiating template 'template'—Compiler error
Error writing output file—Compiler error
Exceeded memory limit for block 'address' in module 'module'—Linker error
'__except' or '__finally' Expected Following '__try'—Compiler error
Exception handling not enabled—Compiler error
Exception handling variable may not be used here—Compiler error
Exception specification not allowed here—Compiler error
Expected a ‘:’ or ‘=’: ‘identifier’—Linker error

Expected a file name: ‘identifier’—Linker error
Expected an option: ‘identifier’—Linker error
Explicit instantiation can only be used at global scope (E2420)—Compiler error
Explicit instantiation must be used with a template class or function—Compiler error
Explicit instantiation only allowed at file or namespace scope—Compiler error
Explicit instantiation requires an elaborated type specifier (i.e., “class foo<int>”) (E2505)—Compiler error
Explicit specialization declarator "template<>" now required—Compiler error
Explicit specialization must be used with a template class or function—Compiler error
Explicit specialization of 'specifier' requires 'template<>' declaration (E2426)—Compiler error
Explicit specialization of 'specifier' is ambiguous: must specify template arguments (E2506)—Compiler error
Explicit specialization only allowed at file or namespace scope—Compiler error
Explicit specialization or instantiation of non-existing template 'template' (E2423)—Compiler error
Explicitly specializing an explicitly specialized class member makes no sense (W8077)—Compiler warning
'export' keyword must precede a template declaration—Compiler error
Export ‘symbol’ in module ‘module’ references ‘symbol’ in unit ‘unit’—Linker error
Exports ‘symbol’ and ‘symbol’ both have the same ordinal: ‘number’—Linker error
Expression expected—Compiler error
Expression of scalar type expected—Compiler error
Expression syntax—Compiler error
Expression syntax error in !if statement—MAKE error
Extern variable cannot be initialized—Compiler error
Extra parameter in call—Compiler error
Extra parameter in call to function—Compiler error

F

Failed to create resource file 'filename' 'number'—Linker error
Failed to create state file 'filename' (error code 'number')—Linker error
Failed read from 'filename' at offset offset for n bytes—Linker error
'__far16' may only be used with '__pascal' or '__cdecl'—Compiler error
FATAL ERROR: Cannot Load Linker: 'linker'—IDE error
FATAL ERROR: Linker CREATE Failed—IDE error
FATAL ERROR: Linker missing CREATE Entry Point—IDE error
FATAL ERROR: Linker missing DESTROY Entry Point—IDE error
FATAL ERROR: GP FAULT—MAKE error
File alignment value is bigger than section alignment—Linker error
File must contain at least one external declaration—Compiler error
File name too long—Compiler error
Filename too long—MAKE error
'filename' couldn't be created, original won't be changed—Librarian warning
'filename' does not exist—don't know how to make it—MAKE error
'filename' file not found—Librarian error
'filename' file not found—Librarian warning
'filename' not a MAKE—MAKE error
First base must be VCL class—Compiler error
Fixup to empty segment in module ‘module’—Linker error
Floating point error: Divide by 0—Runtime error
Floating point error: Domain—Runtime error
Floating point error: Overflow—Runtime error
Floating point error: Partial loss of precision—Runtime error

Floating point error: Stack fault—Runtime error
Floating point error: Underflow—Runtime error
For statement missing (—Compiler error
For statement missing)—Compiler error
For statement missing ;—Compiler error
Friends must be functions or classes—Compiler error
Function body ignored—Compiler warning
Function call missing)—Compiler error
Function call terminated by unhandled exception 'value' at address 'addr'—Compiler error
'function' cannot be a template function—Compiler error
'function' cannot be declared as static or inline—Compiler error
'function' cannot return a value—Compiler error
Function defined inline after use as extern—Compiler error
Function definition cannot be a typedef'ed declaration—Compiler error
Function 'function' cannot be static—Compiler error
Function 'function' redefined as non-inline (W8085)—Compiler warning
'function' is obsolete—Compiler warning
'function' must be declared with no parameters—Compiler error
'function' must be declared with one parameter—Compiler error
'function' must be declared with two parameters—Compiler error
Function should return a value—Compiler error
Function should return a value—Compiler warning
'function' was previously declared with the language 'language'—Compiler error
'function1' cannot be distinguished from 'function2'—Compiler error
'function1' hides virtual function 'function2'—Compiler warning or error
Functions 'function1' and 'function2' both use same dispatch number—Compiler error
Functions taking class by value arguments are not expanded inline—Compiler warning
Functions with exception specifications are not expanded inline—Compiler warning
Functions cannot return arrays or functions—Compiler error
Functions containing ‘statement’ are not expanded inline—Compiler warning
Functions may not be part of a struct or union—Compiler error

G

General error in link set—Linker error
Global anonymous union not static—Compiler error
Goto bypasses initialization of a local variable—Compiler error
Goto into an exception handler is not allowed—Compiler error
Goto statement missing label—Compiler error
Group overflowed maximum size: 'group'—Compiler error

H

Handler for 'type1' hidden by previous handler for 'type2'—Compiler error
Heap reserve size is less than the commit size—Linker error
Hexadecimal value contains more than 3 digits—Compiler error or warning

I

'identifier' cannot be declared in an anonymous union—Compiler error
'identifier' cannot start a parameter declaration—Compiler error
Identifier expected—Compiler error
Identifier 'identifier' cannot have a type qualifier—Compiler error

'identifier' is assigned a value that is never used—Compiler error or warning
'identifier' is declared as both external and static—Compiler error or warning
'identifier' is declared but never used—Compiler warning
'identifier' is not a member of 'struct'—Compiler error
'identifier' is not a non-static data member and can't be initialized here—Compiler error
'identifier' is not a parameter—Compiler error
'identifier' is not a public base class of 'classtype'—Compiler error
'identifier' must be a member function—Compiler error
'identifier' must be a member function or have parameter of class type—Compiler error
'identifier' must be a previously defined class or struct—Compiler error
'identifier' must be a previously defined enumeration tag—Compiler error
'identifier' requires VCL style class type—Compiler error
'identifier' specifies multiple or duplicate access—Compiler error
If statement missing (—Compiler error
If statement missing)—Compiler error
If statement too long—MAKE error
Ifdef statement too long—MAKE error
Ifndef statement too long—MAKE error
Ignored 'module', path is too long—Librarian warning
Ignored 'path'—path is too long—Librarian error
Ill-formed pragma—Compiler warning
Illegal base class type: formal type 'type' resolves to 'type' (E2402)—Compiler error
Illegal character 'character' (0x'value')—Compiler error
Illegal character in constant expression 'expression'—MAKE error
Illegal fixup type 'type' at offset 'address' in module 'module'—Linker error
Illegal initialization—Compiler error
Illegal number format: ‘symbol’—Linker error
Illegal number suffix—Compiler error
Illegal octal digit—Compiler error
Illegal octal digit—MAKE error
Illegal option: ‘option’—Linker error
Illegal parameter to __emit__—Compiler error
Illegal pointer subtraction—Compiler error
Illegal structure operation—Compiler error
Illegal to take address of bit field—Compiler error
Illegal type ‘type’ in automated section—Compiler error
Illegal 'type' fixup index in module 'module'—Linker error
Illegal use of closure pointer—Compiler error
Illegal use of floating point—Compiler error
Illegal use of member pointer—Compiler error
Illegal use of pointer—Compiler error
Illegal/invalid option in CMDSWITCHES directive 'option'—MAKE error
Image linked as an executable, but with a .DLL or .BPL extension—Linker warning
Implicit conversion of 'type1' to 'type2' not allowed—Compiler error
Import 'symbol' in module 'module' clashes with prior module—Librarian error
Improper use of typedef 'identifier'—Compiler error
Include files nested too deep—Compiler error
Incompatible type conversion—Compiler error
Incompatible version of RLINK32.DLL—Linker error

Incorrect command line argument: —MAKE error
Incorrect number format—Compiler error
Incorrect option—Compiler error
Incorrect use of default—Compiler error
Incorrect use of #pragma alias “aliasname”=”substituteName (W8086)—Compiler error
Incorrect use of #pragma codeseg [seg_name] ["seg_class"] [group] (W8093)—Compiler error
Incorrect use of #pragma code_seg(["seg_name"["seg_class"]]) (W8096)—Compiler error
Incorrect use of #pragma comment(<type> ["string"]) (W8094)—Compiler error
Incorrect use of #pragma message("string") (W8095)—Compiler error
Initialization is only partially bracketed—Compiler warning
Initializer for object 'x' ignored—Compiler warning
Initializing 'identifier with 'identifier'—Compiler error
Initializing enumeration with type—Compiler warning
Initializing 'type' with 'type'—Compiler warning
Inline assembly not allowed—Compiler error
Inline assembly not allowed in inline and template function—Compiler error
In-line data member initialization requires an integral constant expression—Compiler error
Instantiating 'specifier' (E2441)—Compiler error
Int and string types compared—MAKE error
Integer arithmetic overflow—Compiler warning
Internal code generator error—Compiler error
Internal compiler error—Compiler error
Invalid call to uuidof(struct type|variable) (E2496) —Compiler error
Invalid combination of opcode and operands—Compiler error
Invalid __declspec(uuid(GuidString)) format (E2499)—Compiler error
Invalid explicit specialization of 'specifier' (E2473)—Compiler error
Invalid GUID string (E2493)—Compiler error
Invalid indirection—Compiler error
Invalid macro argument separator—Compiler error
Invalid object file 'filename'—Linker error
Invalid page size value ignored—Librarian warning
Invalid pointer addition—Compiler error
Invalid register combination (e.g. [BP+BX])—Compiler error
Invalid template argument list—Compiler error
Invalid template declarator list (E2100)—Compiler error
Invalid template declaration—Compiler error
Invalid use of dot—Compiler error
Invalid use of namespace 'identifier'—Compiler error
Invalid use of template keyword—Compiler error
Invalid use of template ‘template’ (E2107)—Compiler error
Irreducible expression tree—Compiler error

L

Last parameter of 'operator' must have type 'int'—Compiler error
Library contains COMDEF records—extended dictionary not created—Librarian warning
Library too large, please restart with /P 'size'—Librarian error
Library too large, restart with library page size 'size'—Librarian error
LIN%s: Last line lineno (hex) is less than first line lineno (hex) for symbol \—Linker warning
LIN%s: First line lineno (hex) is less than previous last line lineno (hex) for symbol \—Linker warning

Linkage specification not allowed—Compiler error
Linker CREATE Failed—IDE error
Linker missing CREATE Entry Point—IDE error
Linker missing DESTROY Entry Point—IDE error
Link terminated by user—IDE error
Local data exceeds segment size limit—Compiler error
Lvalue required—Compiler error

M

Macro argument syntax error—Compiler error
Macro definition ignored—Compiler warning
Macro expansion too long—Compiler error
Macro expansion too long—MAKE error
Macro replace text 'string' is too long—MAKE error
Macro substitute text 'string' is too long—MAKE error
'macroname'—')' missing in macro invocation—MAKE error
'main' cannot be declared as static or inline—Compiler error
'main' cannot be a template function—Compiler error
Main must have a return type of int—Compiler error
Malloc of number bytes failed in module, line number—Linker error
Matching base class function 'function' has different dispatch number—Compiler error
Matching base class function 'function' is not dynamic—Compiler error
Maximum instantiation depth exceeded; check for recursion (E2418)—Compiler error
Maximum option context replay depth exceeded; check for recursion (E2289)—Compiler error
Maximum precision used for member pointer type type—Compiler error
Maximum token reply depth exceeded; check for recursion (E2288)—Compiler error
Maximum VIRDEF count exceeded; check for recursion (E2491)—Compiler error
Member function must be called or its address taken—Compiler error
Member identifier expected—Compiler error
Member is ambiguous: 'member1' and 'member2'—Compiler error
'Member' is not a member of 'class', because the type is not yet defined—Compiler error
'member' is not accessible—Compiler error
'member' is not a valid template type member—Compiler error
Member 'member' cannot be used without an object—Compiler error
Member 'member' has the same name as its class—Compiler error
Member 'member' is initialized more than once—Compiler error
Member pointer required on right side of .* or ->*—Compiler error
Memory full listing truncated!—Librarian warning
Memory reference expected—Compiler error
Mismatch in kind of substitution argument and template parameter 'parameter' (E2389)—Compiler error
Misplaced break—Compiler error
Misplaced continue—Compiler error
Misplaced decimal point—Compiler error
Misplaced elif directive—Compiler error
Misplaced elif statement—MAKE error
Misplaced else—Compiler error
Misplaced else statement—MAKE error
Misplaced else directive—Compiler error
Misplaced endif directive—Compiler error

Misplaced endif statement—MAKE error
Missing or incorrect version of TypeLibImport.dll (E2503)—Compiler error
Missing template parameters for friend template 'template' (E2410)—Compiler error
'module' already in LIB, not changed!—Librarian warning
'module' contains invalid OMF record, type 0xHH—Linker error
'module' ILINK32 does not support segmentation - use TLINK32—Linker error
'module' not found in library—Librarian warning
Mixing pointers to different 'char' types—Compiler error
Mixing pointers to signed and unsigned char—Compiler warning
Multiple base classes not supported for VCL classes—Compiler error
Multiple base classes require explicit class names—Compiler error
Multiple declaration for 'identifier'—Compiler error
Must take address of a memory location—Compiler error

N

Namespace member 'identifier' declared outside its namespace—Compiler error
Namespace name expected—Compiler error
Need an identifier to declare—Compiler error
Need previously defined struct GUID (E2498)—Compiler error
Need to include header <typeinfo> to use typeid—Compiler error
Negating unsigned value—Compiler warning
No : following the ?—Compiler error
No base class to initialize—Compiler error
No closing quote—MAKE error
No declaration for function 'function'—Compiler error or warning
No file name ending—Compiler error
No file names given—Compiler error
No filename ending—MAKE error
No GUID associated with type ‘type’ (E2497)—Compiler error
No macro before =—MAKE error
No match found for wildcard 'expression'—MAKE error
No terminator specified for in-line file operator—MAKE error
No type OBJ file present. Disabling external types option—Compiler warning
No type information—Compiler error
Non-ANSI Keyword Used: 'keyword'—Compiler error
Non-const function function called for const object—Compiler error
Non-constant function 'function' called for constant object—Compiler warning
Non-virtual function 'function' declared pure—Compiler error
Non-volatile function 'function' called for volatile object—Compiler error or warning
Nonportable pointer comparison—Compiler error or warning
Nonportable pointer conversion—Compiler error
Nonportable pointer conversion—Compiler warning
Nontype template argument must be of scalar type—Compiler error
Non-type template parameters cannot be of floating point, class, or void type (E2431)—Compiler error
Not a valid partial specialization of 'specifier' (E2429)—Compiler error
Not all options can be restored at this time—Compiler error
Not an allowed type—Compiler error
Not enough memory—MAKE error
Not enough memory for command-line buffer—Librarian error

Nothing allowed after pragma option pop—Compiler error
Null pointer assignment—Runtime error
Number of allowable option contexts exceeded!—Compiler error
Number of template parameters does not match in redeclaration of 'specifier' (E2430)—Compiler error
Numeric constant too large—Compiler error

O

Objects of type 'type' cannot be initialized with { }—Compiler error
Only __fastcall functions allowed in __automated section—Compiler error
Only member functions may be 'const' or 'volatile'—Compiler error
Only one of a set of overloaded functions can be functions can be "C"—Compiler error
Only <<KEEP or <<NOKEEP—MAKE error
Only read or write clause allowed in property declaration in__automated section—Compiler error
Operand of 'delete' must be non-const pointer—Compiler error
operator -> must return a pointer or a class—Compiler error
operator [] missing]—Compiler error
operator delete must return void—Compiler error
Operator delete[] must return void—Compiler error
Operator must be declared as function—Compiler error
'operator' must be declared with one or no parameters—Compiler error
'operator' must be declared with one or two parameters—Compiler error
Operator new must have an initial parameter of type size_t—Compiler error
Operator new must return an object of type void *—Compiler error
Operator new[] must have an initial parameter of type size_t—Compiler error
Operator new[] must return an object of type void—Compiler error
'operator::operator==' must be publicly visible to be contained by a 'type' (W8087)—Compiler error
'operator::operator<' must be publicly visible to be contained by a 'type' (W8089)—Compiler error
'operator::operator<' must be publicly visible to be used with 'type' (W8090)—Compiler error
Operator 'operator' not implemented in type 'type' for arguments of the same type—Compiler error
Operator 'operator' not implemented in type 'type' for arguments of type 'type'—Compiler error
Operators may not have default argument values—Compiler error
Out of disk space—Linker error
Out of memory—Compiler error
Out of memory—Librarian error
Out of memory—Linker error
Out of memory creating extended dictionary—Librarian error
Out of memory reading LE/LIDATA record from object module—Librarian error
Out of space allocating per module debug struct—Librarian error
Output device is full—Librarian error
Overlays only supported in medium, large, and huge memory models—Compiler error
Overloadable operator expected—Compiler error
Overloaded 'function name' ambiguous in this context—Compiler error
Overloaded 'function name' ambiguous in this context—Compiler error
Overloaded function resolution not supported—Compiler error

P

Packages must be linked with the startup code in C0PKG32.OBJ—Linker error
Parameter mismatch in 'specifier' access specifier of property 'property'—Compiler error
Parameter names are used only with a function body—Compiler error
Parameter 'number' missing name—Compiler error

Parameter 'parameter' is never used—Compiler error and warning
Partial specializations may not specialize dependent non-type parameters ('parameter') (E2387)—Compiler error
Pointer to overloaded function 'function' doesn't match 'type'—Compiler error
Pointer to structure required on left side of -> or ->*—Compiler error
Possible overflow in shift operation—Compiler warning
Possible use of 'identifier' before definition—Compiler error and warning
Possibly incorrect assignment—Compiler error and warning
pragma checkoption failed: options are not as expected—Compiler error
Pragma option pop with no matching option push—Compiler error
Pragma pack pop with no matching pack push (W8083)—Compiler warning
Previous options and warnings not restored—Compiler error
Printf/Scanf floating point formats not linked—Runtime error
Properties may only be assigned using a simple statement, e.g. \"prop = value;\" (E2492)—Compiler error
Public symbol for EXPDEF ‘symbol’ not found in module ‘module’—Linker error
Public 'symbol' in module 'module1' clashes with prior module 'module2'—Librarian error
Public symbol 'symbol' defined in both library module 'module1' and 'module2'—Linker warning
Public symbol 'symbol' defined in both module 'module1' and 'module2'—Linker warning
__published or __automated sections only supported for VCL classes—Compiler error
Published property access functions must use __fastcall calling convention—Compiler error
Pure virtual function called—Runtime error

Q

Qualifier 'identifier' is not a class or namespace name—Compiler error

R

Realloc of number bytes failed in module, line number—Linker error
'reason'—extended dictionary not created—Librarian warning
Record kind 'num' found, expected theadr or lheadr in module 'filename'—Librarian error
Record length 'len' exceeds available buffer in module 'module'—Librarian error
Record type 'type' found, expected theadr or lheadr in module—Librarian error
Recursive template function: '" instantiated "'—Compiler error
Redeclaration of #pragma package with different arguments—Compiler error
Redeclaration of property not allowed in __automated section'—Compiler error
Redefinition of 'macro' is not identical—Compiler error or warning
Redefinition of target 'filename'—MAKE error
Redefinition of uuid is not identical (E2495)—Compiler error
Reference initialized with 'type1', needs lvalue of type 'type2'—Compiler error
Reference member 'member' Initialized with a non-reference parameter—Compiler error
Reference member 'member' in class without constructors—Compiler error
Reference member 'member' is not initialized—Compiler error
Reference member 'member' needs a temporary for initialization—Compiler error
Reference variable 'variable' must be initialized—Compiler error
Register allocation failure—Compiler error
Repeat count needs an lvalue—Compiler error
Restarting compile using assembly—Compiler warning
Results are safe in file 'filename'—Librarian warning
RLINK32 was not initialized—Linker error
RTL helper function ‘function’ not found—Linker error
RTTI not available for expression evaluation—Compiler error

Rule line too long—MAKE error

S

Section 'section' defined in .def file is empty—Linker error
Section 'section' not found—Linker warning
Side effects are not allowed—Compiler error
Size of 'identifier' is unknown or zero—Compiler error
Size of the type 'identifier' is unknown or zero—Compiler error
Size of the type is unknown or zero—Compiler error
sizeof may not be applied to a bit field—Compiler error
sizeof may not be applied to a function—Compiler error
Specialization after first use of template—Compiler error
Specialization within template classes not yet implemented (E2290)—Compiler error
'specifier' has already been included—Compiler error
Stack overflow—Runtime error
Stack reserve size is less than the commit size—Linker error
Statement missing ;—Compiler error
Static data members not allowed in __published or __automated sections—Compiler error
Storage class 'storage class' is not allowed here—Compiler error
Storage specifier not allowed for array properties—Compiler error
String constant expected (E2482)—Compiler error
String type not allowed with this operand—MAKE error
Stripping relocations from a DLL may cause it to malfunction—Linker warning
Structure packing size has changed—Compiler warning
Structure passed by value—Compiler error or warning
Structure required on left side of . or .*—Compiler error
Structure size too large—Compiler error
Style of function definition is now obsolete—Compiler error or warning
Subscripting missing]—Compiler error
Suggest parentheses to clarify precedence (W8084)—Compiler warning
Superfluous & with function—Compiler error or warning
Suspicious pointer arithmetic—Compiler warning
Suspicious pointer conversion—Compiler error or warning
Switch selection expression must be of integral type—Compiler error
Switch statement missing (—Compiler error
Switch statement missing)—Compiler error
Symbol ‘symbol’ from delayed load module ‘module’ is a data reference—Linker error
Symbol 'symbol' marked as __import in 'module' is public in 'module'—Linker error
Symbol ‘symbol1’ is aliased to ‘symbol2,’ which is already aliased—Linker error

T

Template argument cannot have static or local linkage (E2397)—Compiler error
Template argument must be a constant expression—Compiler error
Template class nesting too deep: 'class'—Compiler error
Template declaration missing template parameters ('template<...>') (E2434)—Compiler error
Template function argument 'argument' not used in argument types—Compiler error
Template functions may only have 'type-arguments'—Compiler error
Template instance 'specifier' is already instantiated (W8076)—Compiler warning
Template instance 'template' is already instantiated (E2392)—Compiler error

'template' qualifier must name a template class or function instance—Compiler error
'template' qualifier must specify a member template name (E2105)—Compiler error
Template template arguments must name a class (E2438)—Compiler error
Templates and overloaded operators cannot have C linkage—Compiler error
Templates can only be declared at namespace or class scope—Compiler error
Templates must be classes or functions—Compiler error
Templates not supported—Compiler error
Temporary used for parameter 'parameter'—Compiler warning
Temporary used for parameter 'number'—Compiler warning
Temporary used for parameter 'number' in call to 'function'—Compiler warning
Temporary used for parameter 'parameter'—Compiler warning
Temporary used for parameter 'parameter'—Compiler warning
Temporary used for parameter 'parameter' in call to 'function'—Compiler warning
Temporary used to initialize 'identifier'—Compiler error or warning
The '...' handler must be last—Compiler error
The combinations '+*' or '*+' are not allowed—Librarian error
The constructor 'constructor' is not allowed—Compiler error
The name of template class 'class' cannot be overloaded (E2484)—Compiler error
The unit name 'unit' is redefined by module 'module' (original definition in 'module')—Linker error
The value for 'identifier' is not within the range of an int—Compiler error
There can only be one 'main unit' per project—Linker error
'this' can only be used within a member function—Compiler error
Throw expression violates exception specification—Compiler warning
Too few arguments passed to template 'template'—Compiler error
Too few parameters in call—Compiler error
Too few parameters in call to function—Compiler error
Too few template parameters were declared for template 'template' (E2477)—Compiler error
Too many arguments passed to template 'template'—Compiler error
Too many commas on command line: ‘identifier’—Linker error
Too many candidate template specializations from 'specifier' (E2295)—Compiler error
Too many decimal points—Compiler error
Too many DEF file names: ‘identifier’—Linker error
Too many default cases—Compiler error
Too many error or warning messages—Compiler error
Too many errors; stopping link—Linker error
Too many EXE file names:’identifier’—Linker error
Too many exponents—Compiler error
Too many exports: only 65535 permitted—Linker error
Too many initializers—Compiler error
Too many MAP file names:’identifier’—Linker error
Too many returns in response file: ‘filename’—Linker error
Too many rules for target 'target'—MAKE error
Too many section/segment definitions found in .def file—Linker error
Too many storage classes in declaration—Compiler error
Too many suffixes in .SUFFIXES list—MAKE error
Too many template parameter sets were specified (E2435)—Compiler error
Too many template parameters were declared for template 'template' (E2478)—Compiler error
Too many types in declaration—Compiler error
Too much global data defined in file—Compiler error

Two consecutive dots—Compiler error
Two operands must evaluate to the same type—Compiler error
type argument ‘specifier’ passed to ‘function’ is an ‘iterator category’ iterator: ‘iterator category’ iterator required (W8091)—Compiler error
type argument ‘specifier’ passed to ‘function’ is not an iterator: ‘type’ iterator required (W8092)—Compiler error
'type' is not a polymorphic class type—Compiler error
Type 'type' is not a defined class with virtual functions—Compiler error
Type 'typename' may not be defined here—Compiler error
Type index 'number' is bad in module 'module'—Linker error
Type mismatch in default argument value—Compiler error
Type mismatch in default value for parameter 'parameter'—Compiler error
Type mismatch in parameter 'number'—Compiler error
Type mismatch in parameter 'number' in call to 'function'—Compiler error
Type mismatch in parameter 'number' in template class name 'template'—Compiler error
Type mismatch in parameter 'parameter'—Compiler error
Type mismatch in parameter 'parameter' in call to 'function'—Compiler error
Type mismatch in parameter 'parameter' in template name 'template'—Compiler error
Type mismatch in redeclaration of 'identifier'—Compiler error
Type name expected—Compiler error
'typename' is only allowed in template declarations (E2439)—Compiler error
'typename' should be followed by a qualified, dependent type name (E2437)—Compiler error

U

Unable to create output file 'filename'—Compiler error
Unable to create turboc.$ln—Compiler error
Unable to execute command 'command'—Compiler error
Unable to execute command: 'command'—MAKE error
Unable to load DLL ‘filename'—Linker warning
Unable to load RW32CORE.DLL—Resource compiler error
Unable to open include file 'filename'—MAKE error
Unable to open file 'filename'—MAKE error
Unable to open file 'filename'—Linker error
Unable to open 'filename'—Compiler error
Unable to open 'filename' for output—Librarian error
Unable to open import file 'filename' (E2501)—Compiler error
Unable to open include file 'filename'—Compiler error
Unable to open input file 'filename'—Compiler error
Unable to open makefile—MAKE error
Unable to perform incremental linkperforming full link...—Linker error
Unable to redirect input or output—MAKE error
Unable to rename 'filename1' to 'filename2'—Librarian error
#undef directive ignored—Compiler warning
Undefined label 'identifier'—Compiler error
Undefined structure 'structure'—Compiler error
Undefined structure 'structure'—Compiler warning
Undefined symbol 'identifier'—Compiler error
Unexpected }—Compiler error
Unexpected char X in command line —Librarian error
Unexpected end of file—MAKE error

Unexpected end of file in comment started on 'line number'—Compiler error
Unexpected end of file in conditional started at line 'line number'—MAKE error
Unexpected end of file in conditional started on 'line number'—Compiler error
Unexpected string constant (E2481)—Compiler error
Unexpected termination during compilation [Module Seg#:offset]—Compiler error
Union cannot be a base type—Compiler error
Union cannot have a base type—Compiler error
Union member 'member' is of type class with constructor—Compiler error
Union member 'member' is of type class with destructor—Compiler error
Union member 'member' is of type class with operator =—Compiler error
Unions cannot have virtual member functions—Compiler error
Unit 'unit' (defined by 'name') depends on unit 'unit', but no implementation was found—Linker error
Unknown assembler instruction—Compiler warning
Unknown CMDSWITCHES operator 'operator'—MAKE error
Unknown command line switch 'X' ignored—Librarian warning
Unknown fatal error—Resource compiler error
Unknown error (# errornum)—IDE error
Unknown language, must be C or C++—Compiler error
Unknown preprocessor directive: 'identifier'—Compiler error
Unknown preprocessor statement—MAKE error
Unknown RLINK32 error—Linker error
Unknown unit directive: 'directive'—Compiler error
Unreachable code—Compiler warning
Unrecognized __declspec modifier (E2494)—Compiler error
Unresolved external 'symbol' referenced from 'module'—Linker error
Unsupported 16-bit segment(s) in module 'module'—Linker error
Unterminated string or character constant—Compiler error
Use '> >' for nested templates instead of '>>'—Compiler error
Use . or -> to call function—Compiler error
Use . or -> to call 'member', or & to take its address—Compiler error
Use :: to take the address of a member function—Compiler error
Use __declspec(spec1[, spec2]) to combine multiple __declspecs—Compiler error
Use of : and :: dependents for target 'target'—MAKE error
Use of dispid with a property requires a getter or setter—Compiler error
Use qualified name to access member type 'identifier'—Compiler warning
User break—Compiler error
User break—IDE error
User break, library aborted—Librarian error
User-defined message—Compiler error
Using namespace symbol 'symbol' conflicts with intrinsic of the same name—Compiler error

V

Value after -g or -j should be between 0 and 255 inclusive—Compiler error
Value of type void is not allowed—Compiler error
Variable 'identifier' is initialized more than once—Compiler error
'variable' requires runtime initialization/finalization—Compiler error
Variable 'variable' has been optimized and is not available—Compiler error
VCL classes have to be derived from VCL classes—Compiler error
VCL style class must be constructed using operator new—Compiler error

VCL style classes must be caught by reference—Compiler error
VCL style classes need virtual destructors—Compiler error
VCL style classes require exception handling to be enabled—Compiler error
VIRDEF name conflict for 'function'—Compiler error
Virtual base classes not supported for VCL classes—Compiler error
'virtual' can only be used with member functions—Compiler error
'virtual' can only be used with non-template member functions—Compiler error
Virtual function 'function1' conflicts with base class 'base'—Compiler error
virtual specified more than once—Compiler error
void & is not a valid type—Compiler error
Void functions may not return a value—Compiler warning

W

Weak package unit 'unit' cannot contain inits—Linker error
While statement missing (—Compiler error
While statement missing)—Compiler error
Write error on file 'filename'—MAKE error
Wrong number of arguments in call of macro 'macro'—Compiler error

Informational messages Compiler error

The compiler displays status information while compiling if you have checked “Show general messages” on the Compiler page of the Project Options dialog box. Most of the messages are self-explanatory and state information about compiling and linking; for example:

```
[C++] Compiling: D:\Program Files\Borland\CBuilder\Bin\Unit1.cpp
```

You may also see a message such as

```
[C++] Including vcl1.h instead of vcl.h due to -Hr switch
```

This message indicates that the IDE-managed precompiled header file was replaced due to additional VCL classes being included in the project. It does not indicate a failure condition. Sometimes the header file name being included may be the same if it is being included with a different #define statement.

E 2199 Template friend function 'function' must be previously declared

Compiler error

Not used

F1000 Compiler table limit exceeded Compiler error

One of the compiler's internal tables overflowed.

This usually means that the module being compiled contains too many function bodies.

This limitation will not be solved by making more memory available to the compiler. You need to simplify the file being compiled.

F1007 Irreducible expression tree Compiler error

An expression on the indicated line of the source file caused the code generator to be unable to generate code. Avoid using the expression. Notify Inprise if an expression consistently reproduces this error.

F1011 Register allocation failure Compiler error

Possible Causes

An expression on the indicated line of the source file was so complicated that the code generator could not generate code for it.

Solutions

Simplify the expression. If this does not solve the problem, avoid the expression.

Notify Inprise if an expression can consistently reproduce this error.

F1006 Bad call of intrinsic function Compiler error

You have used an intrinsic function without supplying a prototype. You may have supplied a prototype for an intrinsic function that was not what the compiler expected.

F1009 Unable to open input file 'filename'

Compiler error

This error occurs if the source file can't be found.

Check the spelling of the name. Make sure the file is on the specified disk or directory.

Verify that the proper directory paths are listed. If multiple paths are required, use a semicolon to separate them.

F1002 Unable to create output file 'filename'

Compiler error

This error occurs if the work disk is full or write protected.

This error also occurs if the output directory does not exist.

Solutions

If the disk is full, try deleting unneeded files and restarting the compilation.

If the disk is write-protected, move the source files to a writeable disk and restart the compilation.

F1013 Error writing output file Compiler error

A DOS error that prevents the C++ IDE from writing an .OBJ, .EXE, or temporary file.

Solutions

Make sure that the Output directory in the Directories dialog box is a valid directory.

Check that there is enough free disk space.

F1003 Error directive: 'message'

Compiler error

This message is issued when an **#error** directive is processed in the source file.

`'message'` is the text of the **#error** directive.

F1008 Out of memory Compiler error

The total working storage is exhausted.

This error can occur in the following circumstances:

- Not enough virtual memory is available for compiling a particular file. In this case, shut down any other concurrent applications. You may also try to reconfigure your machine for more available virtual memory, or break up the source file being compiled into smaller separate components. You can also compile the file on a system with more available RAM.
- The compiler has encountered an exceedingly complex or long expression at the line indicated and has insufficient reserves to parse it. Break the expression down into separate statements.

F1010 Unable to open 'filename'

Compiler error

This error occurs if the specified file can't be opened.

Make sure the file is on the specified disk or directory. Verify the proper paths are listed. If multiple paths are required, use a semicolon to separate them.

E2141 Declaration syntax error Compiler error

Your source file contained a declaration that was missing a symbol or had an extra symbol added to it. Check for a missing semicolon or parenthesis on that line or on previous lines.

E2219 Wrong number of arguments in call of macro 'macro' Compiler error

Your source file called the named macro with an incorrect number of arguments.

E2022 Array size too large Compiler error

The declared array is larger than 64K and the 'huge' keyword was not used.

If you need an array of this size, either use the 'huge' modifier, like this:

```
int huge array[70000L];  /* Allocate 140000 bytes */
```

or dynamically allocate it with farmalloc( ) or farcalloc( ), like this:

```
int huge *array = (int huge *) farmalloc (sizeof (int) * 70000); ?? Allocate
  140,000 bytes
```

E2220 Invalid macro argument separator Compiler error

In a macro definition, arguments must be separated by commas.

The compiler encountered some other character after an argument name.

This is correct:

```
#define tri_add(a, b, c)  ((a) + (b) + (c))
```

This is incorrect:

```
#define tri_add(a  b. c)  ((a) + (b) + (c))
```

E2026 Assembler statement too long Compiler error

Inline assembly statements can't be longer than 480 bytes.

E2221 Macro argument syntax error Compiler error

An argument in a macro definition must be an identifier.

The compiler encountered some non-identifier character where an argument was expected.

E2046 Bad file name format in include directive Compiler error

OR E2046 Bad file name format in line directive

Include and line directive file names must be surrounded by quotes (`"filename.h"`) or angle brackets (`<filename.h>`).

The file name was missing the opening quote or angle bracket.

If a macro was used, the resulting expansion text is not surrounded by quote marks.

E2062 Invalid indirection

Compiler error

The indirection operator (*) requires a pointer as the operand.

Example

```
int main (void)
{
  int p;
  *p = 10;     /* ERROR: Invalid Indirection */
  return 0;
}
```

E2087 Illegal use of pointer

Compiler error

Pointers can only be used with these operators:

- addition `(+)`
- subtraction `(-)`
- assignment `(=)`
- comparison `(==)`
- indirection `(*)`
- arrow `(->)`

Your source file used a pointer with some other operator.

Example

```
int main (void)
{
  char *p;
  p /= 7;     /* ERROR: Illegal Use of Pointer */
  return 0;
}
```

E2109 Not an allowed type

Compiler error

Your source file declared some sort of forbidden type; for example, a function returning a function or array.

E2110 Incompatible type conversion Compiler error

The cast requested can't be done.

E2039 Misplaced decimal point Compiler error

The compiler encountered a decimal point in a floating-point constant as part of the exponent.

E2041 Incorrect use of default Compiler error

The compiler found no colon after the default keyword.

E2051 Invalid use of dot Compiler error

An identifier must immediately follow a period operator (.). This is a rare message that can only occur in some specialized inline assembly statements.

Example

```
struct foo {
  int x;
  int y;
}p = {0,0};
int y;
int main (void)
{
  asm mov eax.(foo)x, 1;
  asm mov eax.(foo)4, 1;       /* Error: Invalid use of dot */
  return 0;
}
```

E2121 Function call missing) Compiler error

The function call argument list had some sort of syntax error, such as a missing or mismatched right parenthesis.

E2127 Case statement missing : Compiler error

A case statement must have a constant expression followed by a colon.

The expression in the case statement either was missing a colon or had an extra symbol before the colon.

E2129 Character constant must be one or two characters long Compiler error

Character constants can only be one or two characters long.

E2134 Compound statement missing }

Compiler error

The compiler reached the end of the source file and found no closing brace.
This is most commonly caused by mismatched braces.

E2024 Cannot modify a const object Compiler error

This indicates an illegal operation on an object declared to be const, such as an assignment to the object.

E2139 Declaration missing ; Compiler error

Your source file contained a struct or union field declaration that was not followed by a semicolon. Check previous lines for a missing semicolon.

E2153 Define directive needs an identifier Compiler error

The first non-whitespace character after a **#define** must be an identifier.

The compiler found some other character.

E2201 Too much global data defined in file Compiler error

The sum of the global data declarations exceeds 64K bytes. This includes any data stored in the DGROUP (all global variables, literal strings, and static locals).

Solutions

Check the declarations for any array that might be too large. You can also remove variables from the DGROUP.

Here's how:

- Declare the variables as automatic. This uses stack space.
- Dynamically allocate memory from the heap using calloc, malloc, or farmalloc for the variables. This requires the use of pointers.

Literal strings are also put in the DGROUP. Get the file farstr.zip from our BBS to extract literal strings into their own segment.

E2442 Two consecutive dots Compiler error

Because an ellipsis contains three dots (...), and a decimal point or member selection operator uses one dot (.), two consecutive dots cannot legally occur in a C program.

E2175 Too many storage classes in declaration Compiler error

A declaration can never have more than one storage class, either Auto, Register, Static, or Extern.

E2176 Too many types in declaration Compiler error

A declaration can never have more than one basic type. Examples of basic types are:

- char
- class
- int
- float
- double
- struct
- union
- enum
- typedef name

E2053 Misplaced elif directive Compiler error

The compiler encountered an **#elif** directive without any matching **#if**, **#ifdef**, or **#ifndef** directive.

E2055 Misplaced else directive Compiler error

The compiler encountered an **#else** directive without any matching **#if**, **#ifdef**, or **#ifndef** directive.

E2056 Misplaced endif directive

Compiler error

The compiler encountered an **#endif** directive without any matching **#if**, **#ifdef**, or **#ifndef** directive.

E2184 Enum syntax error

Compiler error

An **enum** declaration did not contain a properly formed list of identifiers.

E2186 Unexpected end of file in comment started on line 'number' Compiler error

The source file ended in the middle of a comment.

This is normally caused by a missing close of comment (*/).

E2187 Unexpected end of file in conditional started on line 'number' Compiler error

The source file ended before the compiler (or MAKE) encountered **#endif**.

The **#endif** either was missing or misspelled.

Every **#if** statement needs a matching **#endif** statement.

E2188 Expression syntax Compiler error

This is a catch-all error message when the compiler parses an expression and encounters a serious error.

Possible Causes

This is most commonly caused by one of the following:

- two consecutive operators
- mismatched or missing parentheses
- a missing semicolon on the previous statement.

Solutions

If the line where the error occurred looks syntactically correct, look at the line directly above for errors.

Try moving the line with the error to a different location in the file and recompiling.

If the error still occurs at the moved statement, the syntax error is occurring somewhere in that statement.

If the error occurred in another statement, the syntax error is probably in the surrounding code.

E2192 Too few parameters in call

Compiler error

This error message occurs when a call to a function with a prototype (via a function pointer) had too few arguments. Prototypes require that all parameters be given. Make certain that your call to a function has the same parameters as the function prototype.

E2193 Too few parameters in call to 'function'

Compiler error

A call to the named function (declared using a prototype) has too few arguments.

Make certain that the parameters in the call to the function match the parameters of the function prototype.

E2060 Illegal use of floating point Compiler error

Floating-point operands are not allowed in these operators

- shift (SHL, SHR)
- bitwise Boolean (AND, OR, XOR, NOT)
- conditional (? :)
- indirection (*)
- certain others

The compiler found a floating-point operand with one of these prohibited operators.

E2197 File name too long Compiler error

The file name given in an **#include** directive was too long for the compiler to process. File names in DOS must be no more than 79 characters long.

E2138 Conflicting type modifiers Compiler error

This occurs when a declaration is given that includes more than one addressing modifier on a pointer or more than one language modifier for a function.

Only one language modifier (for example, **__cdecl**, **__pascal**, or **__fastcall**) can be given for a function.

E2271 Goto statement missing label Compiler error

The **goto** keyword must be followed by an identifier.

E2204 Group overflowed maximum size: 'name' Compiler error

The total size of the segments in a group (for example, DGROUP) exceeded 64K.

E2206 Illegal character 'character' (0x'value')

Compiler error

The compiler encountered some invalid character in the input file.

The hexadecimal value of the offending character is printed.

This can also be caused by extra parameters passed to a function macro.

E2063 Illegal initialization Compiler error

Initializations must be one of the following:

- constant expressions
- the address of a global extern or static variable plus or minus a constant

E2209 Unable to open include file 'filename' Compiler error

The compiler could not find the named file.

Possible Causes

- The named file does not exist.
- An **#include** file included itself.
- You do not have FILES set in CONFIG.SYS on your root directory.

Solutions

- Verify that the named file exists.
- Set FILES = 20 in CONFIG.SYS.

E2265 No file name ending Compiler error

The file name in an **#include** statement was missing the correct closing quote or angle bracket.

E2277 Lvalue required Compiler error

The left side of an assignment operator must be an addressable expression.

Addressable expressions include the following:

- numeric or pointer variables
- structure field references or indirection through a pointer
- a subscripted array element

E2228 Too many error or warning messages

Compiler error

There were more errors or warnings than allowed.

E2376 statement missing (

Compiler error

In a do, for, if, switch, or while statement, the compiler found no left parenthesis after the while keyword or test expression.

E2377 statement missing) Compiler error

In a do, for, if, switch, or while statement, the compiler found no right parenthesis after the while keyword or test expression.

E2378 do-while or for statement missing ; Compiler error

In a do or for statement, the compiler found no semicolon after the right parenthesis.

E2356 Type mismatch in redeclaration of 'identifier'

Compiler error

Your source file redeclared a variable with a different type than was originally declared for the variable.

Possible Causes

This can occur if a function is called and subsequently declared to return something other than an integer.

Solutions

If this has happened, you must declare the function before the first call to it.

E2156 Default outside of switch

Compiler error

The compiler encountered a default statement outside a switch statement.

This is most commonly caused by mismatched braces.

E2222 Macro expansion too long

Compiler error

A macro can't expand to more than 4,096 characters.

E2223 Too many decimal points

Compiler error

The compiler encountered a floating-point constant with more than one decimal point.

E2224 Too many exponents

Compiler error

The compiler encountered more than one exponent in a floating-point constant.

E2225 Too many initializers Compiler error

The compiler encountered more initializers than were allowed by the declaration being initialized.

E2309 Inline assembly not allowed Compiler error

Your source file contains inline assembly language statements and you are compiling it from within the integrated environment.

You must use the BCC command to compile this source file from the DOS command line.

E2322 Incorrect number format Compiler error

The compiler encountered a decimal point in a hexadecimal number.

E2324 Numeric constant too large

Compiler error

String and character escape sequences larger than hexadecimal or octal 77 can't be generated.

Two-byte character constants can be specified by using a second backslash. For example,

```
\\
```

represents a two-byte constant.

A numeric literal following an escape sequence should be broken up like this:

```
printf("\x0A" "12345");
```

This prints a carriage return followed by 12345.

E2325 Illegal octal digit Compiler error

The compiler found an octal constant containing a non-octal digit (8 or 9).

E2341 Type mismatch in parameter 'number' in call to 'function'

Compiler error

Your source file declared the named function with a prototype, and the given parameter number (counting left to right from 1) could not be converted to the declared parameter type.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is usually "Cannot convert 'type1' to 'type2'", but the mismatch might be due to many other reasons.

E2343 Type mismatch in parameter 'parameter' in call to 'function'

Compiler error

Your source file declared the named function with a prototype, and the named parameter could not be converted to the declared parameter type.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is usually "Cannot convert 'type1' to 'type2'" but the mismatch might be due to many other reasons.

E2340 Type mismatch in parameter 'number'

Compiler error

The function called, via a function pointer, was declared with a prototype.

However, the given parameter number (counting left to right from 1) could not be converted to the declared parameter type.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is usually "Cannot convert 'type1' to 'type2'" but the mismatch might be due to many other reasons.

E2342 Type mismatch in parameter 'parameter'

Compiler error

Your source file declared the function called via a function pointer with a prototype.

However, the named parameter could not be converted to the declared parameter type.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is usually "Cannot convert 'type1' to 'type2'" but the mismatch might be due to many other reasons.

E2288 Pointer to structure required on left side of -> or ->*

Compiler error

Nothing but a pointer is allowed on the left side of the arrow (->) in C or C++.
In C++ a -> operator is allowed.

E2085 Invalid pointer addition Compiler error

Your source file attempted to add two pointers together.

E2086 Illegal pointer subtraction Compiler error

This is caused by attempting to subtract a pointer from a non-pointer.

E2096 Illegal structure operation

Compiler error

Structures can only be used with dot (.), address-of (&) or assignment (=) operators, or be passed to or from a function as parameters.

The compiler encountered a structure being used with some other operator.

E2047 Bad 'directive' directive syntax

Compiler error

A macro definition starts or ends with the ## operator, or contains the # operator that is not followed by a macro argument name.

An example of this might be:

```
Bad ifdef directive syntax
```

Note that an **#ifdef** directive must contain a single identifier (and nothing else) as the body of the directive.

Another example is:

```
Bad undef directive syntax
```

An **#undef** directive must also contain only one identifier as the body of the directive.

E2380 Unterminated string or character constant Compiler error

The compiler found no terminating quote after the beginning of a string or character constant.

E2381 Structure size too large Compiler error

Your source file declared a structure larger than 64K.

E2048 Unknown preprocessor directive: 'identifier'

Compiler error

The compiler encountered a # character at the beginning of a line. The directive name that followed the # was not one of the following:

- define
- else
- endif
- if
- ifdef
- ifndef
- include
- line
- pragma
- undef

E2451 Undefined symbol 'identifier' Compiler error

The named identifier has no declaration.

Possible Causes

- Actual declaration of identifier has been commented out.
- Misspelling, either at this point or at the declaration.
- An error in the declaration of the identifier.
- The header file in which the identifier is declared was not included using **#include**

Tools to help track down the problem:

- GREP

E2194 Could not find file 'filename' Compiler error

The compiler is unable to find the file supplied on the command line.

E2349 Nonportable pointer conversion

Compiler error

An implicit conversion between a pointer and an integral type is required, but the types are not the same size. You must use an explicit cast.

This conversion might not make any sense, so be sure this is what you want to do.

E2226 Extra parameter in call Compiler error

A call to a function, via a pointer defined with a prototype, had too many arguments.

E2227 Extra parameter in call to function

Compiler error

A call to the named function (which was defined with a prototype) had too many arguments given in the call.

E2119 User break Compiler error

You typed a Ctrl+Break while compiling in the IDE.
(This is not an error, just a confirmation.)

E2114 Multiple base classes require explicit class names

Compiler error

In a C++ class constructor, if there is more than one immediate base class, each base class constructor call in the constructor header must include the base class name.

E2014 Member is ambiguous: 'member1' and 'member2' Compiler error

You must qualify the member reference with the appropriate base class name.

In C++ class 'class', member 'member' can be found in more than one base class, and it was not qualified to indicate which one you meant.

This applies only in multiple inheritance, where the member name in each base class is not hidden by the same member name in a derived class on the same path.

The C++ language rules require that this test for ambiguity be made before checking for access rights (private, protected, public).

It is possible to get this message even though only one (or none) of the members can be accessed.

E2013 'function1' cannot be distinguished from 'function2'

Compiler error

The parameter type lists in the declarations of these two functions do not differ enough to tell them apart.

Try changing the order of parameters or the type of a parameter in one declaration.

E2009 Attempt to grant or reduce access to 'identifier'

Compiler error

A C++ derived class can modify the access rights of a base class member, but only by restoring it to the rights in the base class.

It can't add or reduce access rights.

E2021 Array must have at least one element

Compiler error

ANSI C and C++ require that an array be defined to have at least one element (objects of zero size are not allowed).

An old programming trick declares an array element of a structure to have zero size, then allocates the space actually needed with malloc.

You can still use this trick, but you must declare the array element to have (at least) one element if you are compiling in strict ANSI mode.

Declarations (as opposed to definitions) of arrays of unknown size are still allowed.

Example

```
char ray[];         /* definition of unknown size -- ILLEGAL */
char ray[0];        /* definition of 0 size -- ILLEGAL */
extern char ray[];  /* declaration of unknown size -- OK */
```

E2028 operator -> must return a pointer or a class Compiler error

The C++ operator -> function must be declared to either return a class or a pointer to a class (or struct or union).

In either case, it must be something to which the -> operator can be applied.

E2029 'identifier' must be a previously defined class or struct Compiler error

You are attempting to declare 'identifier' to be a base class, but either it is not a class or it has not yet been fully defined.

Correct the name or rearrange the declarations.

E2030 Misplaced break Compiler error

The compiler encountered a break statement outside a switch or looping construct.

You can only use break statements inside of switch statements or loops.

E2383 Switch selection expression must be of integral type

Compiler error

The selection expression in parentheses in a **switch** statement must evaluate to an integral type (**char, short, int, long, enum**).

You might be able to use an explicit cast to satisfy this requirement.

E2031 Cannot cast from 'type1' to 'type2' Compiler error

A cast from type 'ident1' to type 'ident2' is not allowed.

In C++, you cannot cast a member function pointer to a normal function pointer.

For example:

```
class A {
public:
   int myex();
};
typedef int (*fp)();
test()
{
   fp myfp = (fp) &A::myex; //error
   return myfp();
}
```

The reason being that a class member function takes a hidden parameter, the this pointer, thus it behaves very differently than a normal function pointer.

A static member function behaves as normal function pointer and can be cast.

For example:

```
class A {
public:
   static int myex();
};
typedef int (*fp)();
test()
{
   fp myfp = (fp) &A::myex; //ok
   return myfp();
}
```

However, static member functions can only access static data members of the class.

In C

- A pointer can be cast to an integral type or to another pointer.
- An integral type can be cast to any integral, floating, or pointer type.
- A floating type can be cast to an integral or floating type.

Structures and arrays can't be cast to or from.

You usually can't cast from a void type.

In C++

User-defined conversions and constructors are checked for. If one can't be found, the preceding rules apply (except for pointers to class members).

Among integral types, only a constant zero can be cast to a member pointer.

A member pointer can be cast to an integral type or to a similar member pointer.

A similar member pointer points to a data member (or to a function) if the original does. The qualifying class of the type being cast to must be the same as (or a base class of) the original.

E2136 Constructor cannot have a return type specification

Compiler error

C++ constructors have an implicit return type used by the compiler, but you can't declare a return type or return a value.

E2033 Misplaced continue Compiler error

The compiler encountered a continue statement outside a looping construct.

E2040 Declaration terminated incorrectly Compiler error

A declaration has an extra or incorrect termination symbol, such as a semicolon placed after a function body.

A C++ member function declared in a class with a semicolon between the header and the opening left brace also generates this error.

E2146 Need an identifier to declare Compiler error

In this context, an identifier was expected to complete the declaration.

This might be a typedef with no name, or an extra semicolon at file level.

In C++, it might be a class name improperly used as another kind of identifier.

E2259 Default value missing Compiler error

When a C++ function declares a parameter with a default value, all of the following parameters must also have default values.

In this declaration, a parameter with a default value was followed by a parameter without a default value.

E2042 Declare operator delete (void*) or (void*, size_t)

E2043 Declare operator delete[] (void*) or (void*, size_t) Compiler error

Declare the operator delete with one of the following:

1. A single void* parameter, or
2. A second parameter of type size_t

If you use the second version, it will be used in preference to the first version.

The global operator delete can only be declared using the single-parameter form.

E2044 operator delete must return void

E2044 operator delete[] must return void Compiler error

This C++ overloaded operator delete was declared in some other way.

Declare the operator delete with one of the following:

1. A single void* parameter, or
2. A second parameter of type size_t

If you use the second version, it will be used in preference to the first version.

The global operator delete can only be declared using the single-parameter form.

E2045 Destructor name must match the class name

Compiler error

In a C++ class, the tilde (~) introduces a declaration for the class destructor.

The name of the destructor must be same as the class name.

In your source file, the ~ preceded some other name.

E2165 Destructor cannot have a return type specification

Compiler error

C++ destructors never return a value, and you can't declare a return type or return a value.

E2260 Default value missing following parameter 'parameter' Compiler error

All parameters following the first parameter with a default value must also have defaults specified.

E2054 Misplaced else Compiler error

The compiler encountered an **else** statement without a matching if statement.

Possible Causes

- An extra "else" statement
- An extra semicolon
- Missing braces
- Some syntax error in a previous "if" statement

E2059 Unknown language, must be C or C++ Compiler error

In the C++ construction

```
extern "name" type func( /*...*/ );
```

the given "name" must be "C" or "C++" (use the quotes); other language names are not recognized.

You can declare an external Pascal function without the compiler's renaming like this:

```
extern "C" int pascal func( /*...*/ );
```

To declare a (possibly overloaded) C++ function as Pascal and allow the usual compiler renaming (to allow overloading), you can do this:

```
extern int pascal func( /*...*/ );
```

E2167 'function' was previously declared with the language 'language'

Compiler error

Only one language modifier (cdecl pascal) can be given for a function.

This function has been declared with different language modifiers in two locations.

E2084 Parameter names are used only with a function body Compiler error

When declaring a function (not defining it with a function body), you must use either empty parentheses or a function prototype.

A list of parameter names only is not allowed.

Example declarations

```
int func();             /* declaration without prototype -- OK */
int func(int, int);     /* declaration with prototype -- OK */
int func(int i, int j); /* parameter names in prototype -- OK */
int func(i, j);         /* parameter names only -- ILLEGAL */
```

E2189 extern variable cannot be initialized Compiler error

The storage class extern applied to a variable means that the variable is being declared but not defined here--no storage is being allocated for it.

Therefore, you can't initialize the variable as part of the declaration.

E2064 Cannot initialize 'type1' with 'type2' Compiler error

You are attempting to initialize an object of type 'type1' with a value of type 'type2' which is not allowed. The rules for initialization are essentially the same as for assignment.

E2131 Objects of type 'type' cannot be initialized with { } Compiler error

Ordinary C structures can be initialized with a set of values inside braces.

C++ classes can only be initialized with constructors if the class has constructors, private members, functions, or base classes that are virtual.

E2072 Operator new must return an object of type void *

E2072 Operator new[] must return an object of type void * Compiler error

This C++ overloaded operator new was declared in some other way.

E2079 'function' must be declared with no parameters

Compiler error

This C++ operator function was incorrectly declared with parameters.

E2080 'function' must be declared with one parameter Compiler error

This C++ operator function was incorrectly declared with more than one parameter.

E2081 'function' must be declared with two parameters Compiler error

This C++ operator function was incorrectly declared with other than two parameters.

E2082 'identifier' must be a member function or have a parameter of class type

Compiler error

Most C++ operator functions must have an implicit or explicit parameter of class type.

This operator function was declared outside a class and does not have an explicit parameter of class type.

E2337 Only one of a set of overloaded functions can be "C"

Compiler error

C++ functions are by default overloaded, and the compiler assigns a new name to each function.

If you wish to override the compiler's assigning a new name by declaring the function **extern** "C", you can do this for only one of a set of functions with the same name.

(Otherwise the linker would find more than one global function with the same name.)

E2317 'identifier' is not a parameter Compiler error

In the parameter declaration section of an old-style function definition, 'identifier' is declared but not listed as a parameter. Either remove the declaration or add 'identifier' as a parameter.

E2090 Qualifier 'identifier' is not a class or namespace name Compiler error

The C++ qualifier in the construction qual::identifier is not the name of a struct or class.

E2354 Two operands must evaluate to the same type

Compiler error

The types of the expressions on both sides of the colon in the conditional expression operator (`?:`) must be the same, except for the usual conversions.

These are some examples of usual conversions

- **char** to **int**
- **float** to **double**
- void* to a particular pointer

In this expression, the two sides evaluate to different types that are not automatically converted.

This might be an error or you might merely need to cast one side to the type of the other.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is usually "Cannot convert 'type1' to 'type2'" but the mismatch might be due to many other reasons.

E2297 'this' can only be used within a member function Compiler error

In C++, "this" is a reserved word that can be used only within class member functions.

E2113 Virtual function 'function1' conflicts with base class 'base'

Compiler error

A virtual function has the same argument types as one in a base class, but differs in one or more of the following:

- Return type
- Calling convention
- Exception specification (throw list)

E2115 Bit field too large Compiler error

This error occurs when you supply a bit field with more than 16 bits.

E2116 Bit fields must contain at least one bit

Compiler error

You can't declare a named bit field to have 0 (or less than 0) bits.

You can declare an unnamed bit field to have 0 bits.

This is a convention used to force alignment of the following bit field to a byte boundary (or to a word boundary.

E2076 Overloadable operator expected

Compiler error

Almost all C++ operators can be overloaded.

These are the only ones that can't be overloaded:

- the field-selection dot (`.`)
- dot-star (`.*`)
- double colon (`::`)
- conditional expression (`?:`)

The preprocessor operators (# and ##) are not C or C++ language operators and thus can't be overloaded.

Other non-operator punctuation, such as semicolon (;), can't be overloaded.

E2149 Default argument value redeclared Compiler error

When a parameter of a C++ function is declared to have a default value, this value can't be changed, redeclared, or omitted in any other declaration for the same function.

E2148 Default argument value redeclared for parameter 'parameter' Compiler error

When a parameter of a C++ function is declared to have a default value, this value can't be changed, redeclared, or omitted in any other declaration for the same function.

E2140 Declaration is not allowed here Compiler error

Declarations can't be used as the control statement for while, for, do, if, or switch statements.

E2243 Array allocated using 'new' may not have an initializer Compiler error

When initializing a vector (array) of classes, you must use the constructor that has no arguments.

This is called the default constructor, which means that you can't supply constructor arguments when initializing such a vector.

E2160 Trying to derive a far class from the near base 'base' Compiler error

If a class is declared (or defaults to) near, all derived classes must also be near.

E2163 Trying to derive a near class from the far base 'base' Compiler error

If a class is declared (or defaults to) far, all derived classes must also be far.

E2166 Destructor for ‘class’ is not accessible

Compiler error

The destructor for this C++ class is protected or private, and can't be accessed here to destroy the class.

If a class destructor is private, the class can't be destroyed, and thus can never be used. This is probably an error.

A protected destructor can be accessed only from derived classes.

This is a useful way to ensure that no instance of a base class is ever created, but only classes derived from it.

E2168 Division by zero Compiler error

Your source file contains a divide or remainder in a constant expression with a zero divisor.

E2155 Too many default cases Compiler error

The compiler encountered more than one default statement in a single switch.

E2169 'identifier' specifies multiple or duplicate access Compiler error

A base class can be declared public or private, but not both.

This access specifier can appear no more than once for a base class.

E2170 Base class 'class' is included more than once

Compiler error

A C++ class can be derived from any number of base classes, but can be directly derived from a given class only once.

E2172 Duplicate case Compiler error

Each case of a switch statement must have a unique constant expression value.

E2444 Member 'member' is initialized more than once Compiler error

In a C++ class constructor, the list of initializations following the constructor header includes the same member name more than once.

E2238 Multiple declaration for 'identifier' Compiler error

This identifier was improperly declared more than once.

This might be caused by conflicting declarations such as:

- `int a; double a;`
- a function declared two different ways, or
- a label repeated in the same function, or
- some declaration repeated other than an extern function or a simple variable

This can also happen by inadvertently including the same header file twice. For example, given:

```
//a.h
struct A { int a; };

//b.h
#include "a.h"

//myprog.cpp
#include "a.h"
#include "b.h"
```

myprog.cpp will get two declarations for the struct A. To protect against this, one would write the a.h header file as:

```
//a.h
#ifndef __A_H
#define __A_H

struct A { int a; };

#endif
```

This will allow one to safely include a.h several times in the same source code file.

E2443 Base class 'class' is initialized more than once

Compiler error

In a C++ class constructor, the list of initializations following the constructor header includes base class 'class' more than once.

E2361 'specifier' has already been included Compiler error

This type specifier occurs more than once in this declaration.

Delete or change one of the occurrences.

E2171 Body has already been defined for function 'function'

Compiler error

A function with this name and type was previously supplied a function body.

A function body can only be supplied once.

One cause of this error is not declaring a default constructor which you implement. For example:

```
class A {
public:
   virtual myex();
};
A::A() {} // error
```

Having not seen you declare the default constructor in the class declaration, the compiler has had to generate one, thus giving the error message when it sees one. this is a correct example:

```
class A {
public:
   A();
   virtual myex();
};
A::A() {}
```

E2179 virtual specified more than once

Compiler error

The C++ reserved word "virtual" can appear only once in one member function declaration.

E2183 File must contain at least one external declaration

Compiler error

This compilation unit was logically empty, containing no external declarations. ANSI C and C++ require that something be declared in the compilation unit.

E2247 'member' is not accessible Compiler error

You are trying to reference C++ class member 'member,' but it is private or protected and can't be referenced from this function.

This sometimes happens when you attempt to call one accessible overloaded member function (or constructor), but the arguments match an inaccessible function.

The check for overload resolution is always made before checking for accessibility.

If this is the problem, try an explicit cast of one or more parameters to select the desired accessible function.

Virtual base class constructors must be accessible within the scope of the most derived class. This is because C++ always constructs virtual base classes first, no matter how far down the hierarchy they are. For example:

```
class A {
public:
   A();
};
class B : private virtual A {};

class C : private B {
public:
C();
};

C::C() {} // error, A::A() is not accessible
```

Since A is private to B, which is private to C, it makes A's constructor not accessible to C. However, the constructor for C must be able to call the constructors for its virtual base class, A. If B inherits A publicly, the above example would compile.

E2212 Function defined inline after use as extern Compiler error

Functions can't become inline after they have already been used.

Either move the inline definition forward in the file or delete it entirely.

The compiler encountered something like:

```
myex();
twoex() { myex(); }
inline myex() { return 2; } // error
```

and already used the function as an extern before it saw that it was specified as inline. This would be correct:

```
myex();
inline myex() { return 2; }

twoex() { myex(); }
```

or better:

```
inline myex();
inline myex() { return 2; }

twoex() { myex(); }
```

E2448 Undefined label 'identifier' Compiler error

The named label has a **goto** in the function, but no label definition.

E2190 Unexpected }

Compiler error

An extra right brace was encountered where none was expected. Check for a missing {.

Useful Tip:

The IDE has a mechanism for finding a matching curly brace. If you put the cursor on the '{' or '}' character, hold down Ctrl, hit 'Q' and then '{' or '}', it will position the cursor on the matching brace.

E2015 Ambiguity between 'function1' and 'function2' Compiler error

Both of the named overloaded functions could be used with the supplied parameters. This ambiguity is not allowed.

E2333 Class member 'member' declared outside its class

Compiler error

C++ class member functions can be declared only inside the class declaration.

Unlike nonmember functions, they can't be declared multiple times or at other locations.

E2312 'constructor' is not an unambiguous base class of 'class'

Compiler error

A C++ class constructor is trying to call a base class constructor 'constructor.'

This error can also occur if you try to change the access rights of 'class::constructor.'

Check your declarations.

E2239 'identifier' must be a member function

Compiler error

Most C++ operator functions can be members of classes or ordinary non-member functions, but these are required to be members of classes:

- operator =
- operator ->
- operator ( )
- type conversions

This operator function is not a member function but should be.

E2287 Parameter 'number' missing name

Compiler error

In a function definition header, this parameter consisted only of a type specifier 'number' with no parameter name.

This is not legal in C.

(It is allowed in C++, but there's no way to refer to the parameter in the function.)

E2250 No base class to initialize

Compiler error

This C++ class constructor is trying to implicitly call a base class constructor, but this class was declared with no base classes.

Check your declarations.

E2011 Illegal to take address of bit field

Compiler error

It is not legal to take the address of a bit field, although you can take the address of other kinds of fields.

E2233 Cannot initialize a class member here

Compiler error

Individual members of structs, unions, and C++ classes can't have initializers.

A struct or union can be initialized as a whole using initializers inside braces.

A C++ class can only be initialized by the use of a constructor.

E2254 : expected after private/protected/private

Compiler error

When used to begin a private, protected, or public section of a C++ class, the reserved words "private," "protected," and "public" must be followed by a colon.

E2256 No : following the ? Compiler error

The question mark (?) and colon (:) operators do not match in this expression.

The colon might have been omitted, or parentheses might be improperly nested or missing.

E2257 , expected Compiler error

A comma was expected in a list of declarations, initializations, or parameters.

This problem is often caused by a missing syntax element earlier in the file or one of its included headers.

E2305 Cannot find 'class::class' ('class'&) to copy a vector Compiler error

OR E2305 Cannot find 'class'::operator=('class'&) to copy a vector

Cannot find class::class ...

When a C++ class 'class1' contains a vector (array) of class 'class2', and you want to construct an object of type 'class1' from another object of type 'class 1', you must use this constructor:

```
class2::class2(class2&)
```

so that the elements of the vector can be constructed.

The constructor, called a copy constructor, takes just one parameter (which is a reference to its class).

Usually, the compiler supplies a copy constructor automatically.

However, if you have defined a constructor for class 'class2' that has a parameter of type 'class2&' and has additional parameters with default values, the copy constructor can't exist and can't be created by the compiler.

This is because these two can't be distinguished:

```
class2::class2(class2&)
class2::class2(class2&, int = 1)
```

You must redefine this constructor so that not all parameters have default values.

You can then define a reference constructor or let the compiler create one.

Cannot find class::operator= ...

When a C++ class 'class1' contains a vector (array) of class 'class2', and you want to copy a class of type 'class1', you must use this assignment operator:

```
class2::class2(class2&)
```

so that the elements of the vector can be copied.

Usually, the compiler automatically supplies this operator.

However, if you have defined an operator= for class 'class2' that does not take a parameter of type 'class2&,' the compiler will not supply it automatically--you must supply one.

E2258 Declaration was expected

Compiler error

A declaration was expected here but not found.

This is usually caused by a missing delimiter such as a comma, semicolon, right parenthesis, or right brace.

E2447 'identifier' must be a previously defined enumeration tag

Compiler error

This declaration is attempting to reference 'ident' as the tag of an enum type, but it has not been so declared.

Correct the name, or rearrange the declarations.

E2264 Expression expected Compiler error

An expression was expected here, but the current symbol can't begin an expression.

This message might occur where the controlling expression of an if or while clause is expected or where a variable is being initialized.

This message is often due to a symbol that is missing or has been added.

E2280 Member identifier expected

Compiler error

The name of a structure or C++ class member was expected here, but not found. The right side of a dot (.) or arrow (->) operator must be the name of a member in the structure or class on the left of the operator.

E2316 'identifier' is not a member of 'struct' Compiler error

You are trying to reference 'identifier' as a member of 'struct', but it is not a member.

Check your declarations.

E2061 Friends must be functions or classes

Compiler error

A friend of a C++ class must be a function or another class.

E2200 Functions may not be part of a struct or union Compiler error

This C struct or union field was declared to be of type function rather than pointer to function. Functions as fields are allowed only in C++.

E2272 Identifier expected

Compiler error

An identifier was expected here, but not found.

In C, an identifier is expected in the following situations:

- in a list of parameters in an old-style function header
- after the reserved words struct or union when the braces are not present, and
- as the name of a member in a structure or union (except for bit fields of width 0).

In C++, an identifier is also expected in these situations:

- in a list of base classes from which another class is derived, following a double colon (::), and
- after the reserved word "operator" when no operator symbol is present.

E2304 ‘Constant/Reference’ variable 'variable' must be initialized

Compiler error

This C++ object is declared constant or as a reference, but is not initialized.

It must be initialized at the point of declaration.

E2211 Inline assembly not allowed in inline and template functions

Compiler error

The compiler can't handle inline assembly statements in a C++ inline or template function.

You could eliminate the inline assembly code or, in the case of an inline function, make this a macro, and remove the inline storage class.

E2275 { expected Compiler error

A left brace was expected at the start of a block or initialization.

E2363 Attempting to return a reference to local variable 'identifier'

Compiler error

This C++ function returns a reference type, and you are trying to return a reference to a local (auto) variable.

This is illegal, because the variable referred to disappears when the function exits.

You can return a reference to any static or global variable, or you can change the function to return a value instead.

E2276 (expected Compiler error

A left parenthesis was expected before a parameter list.

E2210 Reference member 'member' is not initialized

Compiler error

References must always be initialized, in the constructor for the class.

A class member of reference type must have an initializer provided in all constructors for that class.

This means you can't depend on the compiler to generate constructors for such a class, because it has no way of knowing how to initialize the references.

E2285 Could not find a match for ‘argument(s)’ Compiler error

No C++ function could be found with parameters matching the supplied arguments. Check parameters passed to function or overload function for parameters that are being passed.

E2255 Use :: to take the address of a member function Compiler error

If f is a member function of class c, you take its address with the syntax

```
&c::f
```

Note the use of the class type name (not the name of an object) and the :: separating the class name from the function name.

(Member function pointers are not true pointer types, and do not refer to any particular instance of a class.)

E2251 Cannot find default constructor to initialize base class 'class' Compiler error

Whenever a C++ derived class 'class2' is constructed, each base class 'class1' must first be constructed.

If the constructor for 'class2' does not specify a constructor for 'class1' (as part of 'class2's' header), there must be a constructor `class1::class1()` for the base class.

This constructor without parameters is called the default constructor.

The compiler will supply a default constructor automatically unless you have defined any constructor for class 'class1'.

In that case, the compiler will not supply the default constructor automatically--you must supply one.

```
class Base {
public:
   Base(int) {}
};
class Derived = public Base {
   Derived():Base(1) {}
}

// must explicitly call the Base constructor, or provide a
// default constructor in Base.
```

Class members with constructors must be initialized in the class' initializer list, for example:

```
class A {
public
   A( int );
};
class B {
public:
   A a;
   B() : a( 3 ) {}; //ok
};
```

E2279 Cannot find default constructor to initialize member 'identifier'

Compiler error

When the following occurs

1. A C++ class 'class1' contains a member of class 'class2,'

and

2. You want to construct an object of type 'class1' (but not from another object of type 'class1'). There must be a constructor class2::class2() so that the member can be constructed.

This constructor without parameters is called the default constructor.

The compiler will supply a default constructor automatically unless you have defined any constructor for class 'class2'.

In that case, the compiler will not supply the default constructor automatically you must supply one.

E2283 Use . or -> to call ‘function’ Compiler error

You attempted to call a member function without providing an object. This is required to call a member function.

```
class X {
   member func() {}
};
X x;
X*xp = new X;
X.memberfunc();
Xp-> memberfunc();
```

E2290 ’code’ missing] Compiler error

This error is generated if any of the following occur:

- Your source file declared an array in which the array bounds were not terminated by a right bracket.
- The array specifier in an operator is missing a right bracket.
- The operator [ ] was declared as operator [.
- A right bracket is missing from a subscripting expression.

Add the bracket or fix the declaration.

Check for a missing or extra operator or mismatched parentheses.

E2291 } expected Compiler error

A right brace was expected at the end of a block or initialization.

E2027 Must take address of a memory location

Compiler error

Your source file used the address-of operator (&) with an expression that can't be used that way; for example, a register variable.

E2293) expected Compiler error

A right parenthesis was expected at the end of a parameter list.

E2379 Statement missing ; Compiler error

The compiler encountered an expression statement without a semicolon following it.

E2373 Bit field cannot be static Compiler error

Only ordinary C++ class data members can be declared static, not bit fields.

E2020 Global anonymous union not static Compiler error

In C++, a global anonymous union at the file level must be static.

E2294 Structure required on left side of . or .* Compiler error

The left side of a dot (.) operator (or C++ dot-star operator, .*) must evaluate to a structure type. In this case it did not.

This error can occur when you create an instance of a class using empty parentheses, and then try to access a member of that 'object'.

E2313 Constant expression required Compiler error

Arrays must be declared with constant size.

This error is commonly caused by misspelling a **#define** constant.

E2314 Call of nonfunction Compiler error

The name being called is not declared as a function.

This is commonly caused by incorrectly declaring the function or misspelling the function name.

E2128 Case outside of switch Compiler error

The compiler encountered a case statement outside a switch statement.
This is often caused by mismatched braces.

E2365 Member pointer required on right side of .* or ->* Compiler error

The right side of a C++ dot-star (.*) or an arrow star (->*) operator must be declared as a pointer to a member of the class specified by the left side of the operator.

In this case, the right side is not a member pointer.

E2303 Type name expected Compiler error

One of these errors has occurred:

- In declaring a file-level variable or a struct field, neither a type name nor a storage class was given.
- In declaring a typedef, no type for the name was supplied.
- In declaring a destructor for a C++ class, the destructor name was not a type name (it must be the same name as its class).
- In supplying a C++ base class name, the name was not the name of a class.

E2455 union cannot have a base type

Compiler error

In general, a C++ class can be of union type, but such a class can't be derived from any other class.

E2468 Value of type void is not allowed

Compiler error

A value of type **void** is really not a value at all, so it can't appear in any context where an actual value is required.

Such contexts include the following:

- the right side of an assignment
- an argument of a function
- the controlling expression of an if, for, or while statement.

E2308 do statement must have while Compiler error

Your source file contained a do statement that was missing the closing while keyword.

E2089 Identifier 'identifier' cannot have a type qualifier Compiler error

A C++ qualifier class::identifier can't be applied here.

A qualifier is not allowed on the following:

- typedef names
- function declarations (except definitions at the file level)
- on local variables or parameters of functions
- on a class member--except to use its own class as a qualifier (redundant but legal).

E2371 sizeof may not be applied to a bit field Compiler error

sizeof returns the size of a data object in bytes, which does not apply to a bit field.

E2372 sizeof may not be applied to a function

Compiler error

sizeof can be applied only to data objects, not functions.
You can request the size of a pointer to a function.

E2092 Storage class 'storage class' is not allowed here Compiler error

The given storage class is not allowed here.

Probably two storage classes were specified, and only one can be given.

E2345 Access can only be changed to public or protected

Compiler error

A C++ derived class can modify the access rights of a base class member, but only to public or protected.

A base class member can't be made private.

E2445 Variable 'identifier' is initialized more than once Compiler error

This variable has more than one initialization. It is legal to declare a file level variable more than once, but it can have only one initialization (even if two are the same).

E2108 Improper use of typedef 'identifier' Compiler error

Your source file used a typedef symbol where a variable should appear in an expression. Check for the declaration of the symbol and possible misspellings.

E2449 Size of 'identifier' is unknown or zero

Compiler error

This identifier was used in a context where its size was needed.

A struct tag might only be declared (the struct not defined yet), or an extern array might be declared without a size.

It's illegal then to have some references to such an item (like sizeof) or to dereference a pointer to this type.

Rearrange your declaration so that the size of 'identifier' is available.

E2453 Size of the type 'identifier' is unknown or zero

Compiler error

This type was used in a context where its size was needed.

For example, a struct tag might only be declared (the struct not defined yet).

It's illegal then to have some references to such an item (like sizeof) or to dereference a pointer to this type.

Rearrange your declarations so that the size of this type is available.

E2452 Size of the type is unknown or zero Compiler error

This error message indicates that an array of unspecified dimension nested within another structure is initialized and the **-A** (ANSI) switch is on. For example:

```
struct
{
  char a[];       //Size of 'a' is unknown or zero
}
  b = { "hello" };      //Size of the type is
                  //unknown or zero
```

E2036 Conversion operator cannot have a return type specification Compiler error

This C++ type conversion member function specifies a return type different from the type itself.

A declaration for conversion function operator can't specify any return type.

E2215 Linkage specification not allowed

Compiler error

Linkage specifications such as extern "C" are only allowed at the file level.

Move this function declaration out to the file level.

E2358 Reference member 'member' needs a temporary for initialization

Compiler error

You provided an initial value for a reference type that was not an lvalue of the referenced type.

This requires the compiler to create a temporary for the initialization.

Because there is no obvious place to store this temporary, the initialization is illegal.

E2019 'identifier' cannot be declared in an anonymous union Compiler error

The compiler found a declaration for a member function or static member in an anonymous union. Such unions can only contain data members.

E2374 Function 'function' cannot be static Compiler error

Only ordinary member functions and the operators new and delete can be declared static. Constructors, destructors and other operators must not be static.

E2339 Cannot overload 'main' Compiler error

You cannot overload main.

E2311 Non-virtual function 'function' declared pure Compiler error

Only virtual functions can be declared pure, because derived classes must be able to override them.

E2088 Bad syntax for pure function definition

Compiler error

Pure virtual functions are specified by appending "= 0" to the declaration, like this:

```
class A { virtual void f () = 0;}
class B : public A { void f () {}; }
```

You wrote something similar, but it was not correct.

E2352 Cannot create instance of abstract class 'class' Compiler error

Abstract classes (those with pure virtual functions) can't be used directly, only derived from.

When you derive an abstract base class, with the intention to instantiate instances of this derived class, you must override each of the pure virtual functions of the base class exactly as they are declared.

For example:

```
class A {
public:
   virtual myex( int ) = 0;
   virtual twoex( const int ) const = 0;
};
class B : public A {
public:
   myex( int );
   twoex( const int );
};
B b;   // error
```

The error occurs because we have not overridden the virtual function in which `twoex` can act on const objects of the class. We have created a new one which acts on non-const objects. This would compile:

```
class A {
public:
   virtual myex( int ) = 0;
   virtual twoex( const int ) const = 0;
};
class B : public A {
public:
   myex( int );
   twoex( const int ) const;
};
B b;   // ok
```

E2350 Cannot define a pointer or reference to a reference

Compiler error

It is illegal to have a pointer to a reference or a reference to a reference.

E2023 Array of references is not allowed Compiler error

It is illegal to have an array of references, because pointers to references are not allowed and array names are coerced into pointers.

E2466 void & is not a valid type

Compiler error

A reference always refers to an object, but an object cannot have the type void.
Thus, the type void is not allowed.

E2068 'identifier' is not a non-static data member and can't be initialized here

Compiler error

Only data members can be initialized in the initializers of a constructor.

This message means that the list includes a static member or function member.

Static members must be initialized outside of the class, for example:

```
class A { static int i; };
int A::i = -1;
```

E2137 Destructor for 'class' required in conditional expression Compiler error

If the compiler must create a temporary local variable in a conditional expression, it has no good place to call the destructor because the variable might or might not have been initialized.

The temporary can be explicitly created, as with classname(val, val), or implicitly created by some other code.

You should recast your code to eliminate this temporary value.

E2185 The value for 'identifier' is not within the range of an int Compiler error

All enumerators must have values that can be represented as an integer.

You have attempted to assign a value that is out of the range of an integer.

If you need a constant of this value, use a const integer.

E2231 Member 'member' cannot be used without an object

Compiler error

This means that you have written class::member where 'member' is an ordinary (non-static) member, and there is no class to associate with that member.

For example, it is legal to write this:

```
obj.class::member
```

but not to write this:

```
class::member
```

E2292 Function should return a value

Compiler warning

Your source file declared the current function to return some type other than **int** or **void**, but the compiler encountered a return with no value. This usually indicates some sort of error.

Functions declared as returning **int** are exempt because older versions of C did not support **void** function return types.

E2450 Undefined structure 'structure'

Compiler warning

The named structure was used in the source file, probably on a pointer to a structure, but had no definition in the source file.

This is probably caused by a misspelled structure name or a missing declaration.

E2463 'base' is an indirect virtual base class of 'class'

Compiler error

You can't create a pointer to a C++ member of a virtual base class.

You have attempted to create such a pointer (either directly, or through a cast) and access an inaccessible member of one of your base classes.

E2319 'identifier' is not a public base class of 'classtype'

Compiler error

The right operand of a .*, ->*, or ::operator was not a pointer to a member of a class that is either identical to (or an unambiguous accessible base class of) the left operand's class type.

E2077 'operator' must be declared with one or no parameters Compiler error

When operator ++ or operator -- is declared as a member function, it must be declared to take either:

- No parameters (for the prefix version of the operator), or
- One parameter of type int (for the postfix version)

E2078 'operator' must be declared with one or two parameters Compiler error

When operator ++ or operator -- is declared as a non-member function, it must be declared to take either:

- one parameter (for the prefix version of the operator), or
- two parameters (for the postfix version)

E2464 'virtual' can only be used with member functions Compiler error

A data member has been declared with the **virtual** specifier.

Only member functions can be declared virtual.

For example:

```
class myclass
{
public:
   virtual int a;    //error
};
```

E2336 Pointer to overloaded function 'function' doesn't match 'type'

Compiler error

A variable or parameter is assigned (or initialized with) the address of an overloaded function.

However, the type of the variable or parameter doesn't match any of the overloaded functions with the specified name.

E2017 Ambiguous member name 'name' Compiler error

Whenever a structure member name is used in inline assembly, such a name must be unique. (If it is defined in more than one structure, all of the definitions must agree as to its type and offset within the structures). In this case, an ambiguous member name has been used.

For example:

```
struct A
{
   int a;
   int b;
};
...
asm ax,.a;
```

E2025 Assignment to 'this' not allowed, use X::operator new instead Compiler error

In early versions of C++, the only way to control allocation of class of objects was by assigning to the 'this' parameter inside a constructor.

This practice is no longer allowed, because a better, safer, and more general technique is to define a member function operator new instead.

For example:

```
this = malloc(n);
```

E2245 Cannot allocate a reference

Compiler error

You have attempted to create a reference using the new operator.

This is illegal, because references are not objects and can't be created through new.

E2384 Cannot call near class member function with a pointer of type 'type'

Compiler error

Also E2385 Cannot call near class member function 'function' with a pointer of type 'type'

Member functions of near classes can't be called via a member pointer.

This also applies to calls using pointers to members.

(Remember, classes are near by default in the tiny, small, and medium memory models.)

Either change the pointer to be near, or declare the class as far.

E2034 Cannot convert 'type1' to 'type2'

Compiler error

An assignment, initialization, or expression requires the specified type conversion to be performed, but the conversion is not legal.

In C++, the compiler will convert one function pointer to another only if the signature for the functions are the same. Signature refers to the arguments and return type of the function. For example:

```
myex( int );
typedef int ( *ffp )( float );
test()
{
   ffp fp = myex; //error
}
```

Seeing that `myex` takes an int for its argument, and `fp` is a pointer to a function which takes a float as argument, the compiler will not convert it for you.

In cases where this is what is intended, performing a typecast is necessary:

```
myex( int );
typedef int ( *ffp )( float );
test()
{
   ffp fp = (ffp)myex;  //ok
}
```

E2123 Class 'class' may not contain pure functions

Compiler error

The class being declared cannot be abstract, and therefore it cannot contain any pure functions.

E2125 Compiler could not generate copy constructor for class 'class'

Compiler error

OR E2125 Compiler could not generate default constructor for class 'class'

OR E2125 Compiler could not generate operator = for class 'class'

Sometimes the compiler is required to generate a member function for the user.

Whenever such a member function can't be generated due to applicable language rules, the compiler issues one of these error messages.

E2232 Constant/Reference member 'member' in class without constructors

Compiler error

A class that contains constant or reference members (or both) must have at least one user-defined constructor.

Otherwise, there would be no way to ever initialize such members.

E2135 Constructor/Destructor cannot be declared 'const' or 'volatile' Compiler error

A constructor or destructor has been declared as **const** or **volatile**.

This is not allowed.

E2152 Default expression may not use local variables Compiler error

A default argument expression is not allowed to use any local variables or other parameters.

E2069 Illegal use of member pointer Compiler error

Pointers to class members can only be passed as arguments to functions, or used with the following operators:

- assignment
- comparison
- `.*`
- `->*`
- `?:`
- `&&`
- `||`

The compiler has encountered a member pointer being used with a different operator.

In order to call a member function pointer, one must supply an instance of the class for it to call upon.

For example:

```
class A {
public:
   myex();
};
typedef int (A::*Amfptr)();
myex()
{
   Amfptr mmyex = &A::myex;
   return (*mmyex)();  //error
}
```

This will compile:

```
cl_ass A {
public:
   myex();
};
typedef int (A::*Amfptr)();
foo()
{
   A a;
   Amfptr mmyex = &A::myex;
   return (a.*mmyex)();
}
```

E2207 Implicit conversion of 'type1' to 'type2' not allowed

Compiler error

When a member function of a class is called using a pointer to a derived class, the pointer value must be implicitly converted to point to the appropriate base class.

In this case, such an implicit conversion is illegal.

E2083 Last parameter of 'operator' must have type 'int' Compiler error

When a postfix operator ++ or operator -- is overloaded, the last parameter must be declared with the type int.

E2235 Member function must be called or its address taken

Compiler error

A reference to a member function must be called, or its address must be taken with & operator.

In this case, a member function has been used in an illegal context.

For example:

```
class A
{
   typedef int (A::* infptr)(void);
public:
   A();
   int myex(void);
   int three;
} a;
A::A()
{
   infptr one = myex;       //illegal - call myex or take address?
   infptr two = &A::myex;   //correct
   three = (a.*one)() + (a.*two)();
}
```

E2357 Reference initialized with 'type1', needs lvalue of type 'type2'

Compiler error

A reference variable that is not declared constant must be initialized with an lvalue of the appropriate type.

In this case, the initializer either wasn't an lvalue, or its type didn't match the reference being initialized.

E2310 Only member functions may be 'const' or 'volatile'

Compiler error

Something other than a class member function has been declared **const** or **volatile**.

E2158 Operand of 'delete' must be non-const pointer

Compiler error

It is illegal to delete a variable that is not a pointer. It is also illegal to delete a pointer to a constant.

For example:

```
const int x=10;
   const int * a = &x;
   int * const b = new int;
   int &c = *b;
   delete a;   //illegal - deleting pointer to constant
   delete b;   //legal
   delete c;   //illegal - operand not of pointer type
               //should use 'delete&c' instead
```

E2330 Operator must be declared as function

Compiler error

An overloaded operator was declared with something other than function type.

For example:

```
class A
{
   A& operator +;    ..note missing parenthesis
};
```

In the example, the function operator '()' is missing, so the operator does not have function type and generates this error.

E2327 Operators may not have default argument values Compiler error

It is illegal for overloaded operators to have default argument values.

E2335 Overloaded 'function name' ambiguous in this context Compiler error

The only time an overloaded function name can be used or assigned without actually calling the function is when a variable or parameter of the correct function pointer type is initialized or assigned the address of the overload function.

In this case, an overloaded function name has been used in some other context, for example, the following code will generate this error:

```
class A{
   A(){myex;}           //calling the function
   void myex(int) {}    //or taking its address?
   void myex(float){}
};
```

E2150 Type mismatch in default argument value Compiler error

The default parameter value given could not be converted to the type of the parameter.

The message "Type mismatch in default argument value" is used when the parameter was not given a name.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is most often "Cannot convert 'type1' to 'type2'" but the mismatch could be due to another reason.

E2151 Type mismatch in default value for parameter 'parameter'

Compiler error

The default parameter value given could not be converted to the type of the parameter.

The message "Type mismatch in default argument value" is used when the parameter was not given a name.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is usually "Cannot convert 'type1' to 'type2'" but the mismatch might be due to many other reasons.

E2454 union cannot be a base type Compiler error

A union can't be used as a base type for another class type.

E2465 unions cannot have virtual member functions

Compiler error

A union can't have virtual functions as its members.

E2456 Union member 'member' is of type class with ‘constructor’ (or destructor, or operator =) Compiler error

A union can't contain members that are of type class with user-defined constructors, destructors, or operator =.

E2284 Use . or -> to call 'member', or & to take its address Compiler error

A reference to a non-static class member without an object was encountered.

Such a member can't be used without an object, or its address must be taken with the & operator.

E2467 'Void function' cannot return a value Compiler error

A function with a return type void contains a return statement that returns a value; for example, an int.

Default = displayed

E2147 'identifier' cannot start a parameter declaration Compiler error

An undefined 'identifier' was found at the start of an argument in a function declarator.

Often the type name is misspelled or the type declaration is missing. This is usually caused by not including the appropriate header file.

E2274 < expected Compiler error

The keyword **template** was not followed by <.

Every template declaration must include the template formal parameters enclosed within < >, immediately following the template keyword.

E2364 Attempting to return a reference to a local object Compiler error

You attempted to return a reference to a temporary object in a function that returns a reference type. This may be the result of a constructor or a function call.

This object will disappear when the function returns, making the reference illegal.

E2142 Base class 'class' contains dynamically dispatchable functions

Compiler error

This error occurs when a class containing a DDVT function attempts to inherit DDVT functions from multiple parent classes.

Currently, dynamically dispatched virtual tables do not support the use of multiple inheritance.

W8005 Bit fields must be signed or unsigned int Compiler warning

In ANSI C, bit fields may only be signed or unsigned int (not char or long, for example).

E2010 Cannot add or subtract relocatable symbols

Compiler error

The only arithmetic operation that can be performed on a relocatable symbol in an assembler operand is addition or subtraction of a constant.

Variables, procedures, functions, and labels are relocatable symbols.

E2214 Cannot have a ‘non-inline function/static data’ in a local class Compiler error

All members of classes declared local to a function must be entirely defined in the class definition.

This means that local classes cannot contain any static data members, and all of their member functions must have bodies defined within the class definition.

E2126 Case bypasses initialization of a local variable

Compiler error

In C++ it is illegal to bypass the initialization of a local variable.

This error indicates a case label that can transfer control past this local variable.

E2321 Declaration does not specify a tag or an identifier Compiler error

This declaration doesn't declare anything.

This may be a struct or union without a tag or a variable in the declaration. C++ requires that something be declared.

For example:

```
struct
{
int a
};
//no tag or identifier
```

E2320 Expression of scalar type expected Compiler error

The !, ++, and -- operators require an expression of scalar type.

Only these types are allowed:

- char
- short
- int
- long
- enum
- float
- double
- long double
- pointer

E2395 Too many arguments passed to template 'template' Compiler error

A template class name specified too many actual values for its formal parameters.

E2446 Function definition cannot be a typedef'ed declaration Compiler error

In ANSI C, a function body cannot be defined using a typedef with a function Type.

Redefine the function body.

E2145 Functions 'function1' and 'function2' both use the same dispatch number

Compiler error

This error indicates a dynamically dispatched virtual table (DDVT) problem.

E2203 Goto bypasses initialization of a local variable

Compiler error

In C++, it is illegal to bypass the initialization of a local variable.

This error indicates a goto statement that can transfer control past this local variable.

E2182 Illegal parameter to __emit__ Compiler error

There are some restrictions on inserting literal values directly into your code with the __emit__ function. For example, you cannot give a local variable as a parameter to __emit__.

E2329 Invalid combination of opcode and operands

Compiler error

The built-in assembler does not accept this combination of operands.

Possible causes

- There are too many or too few operands for this assembler opcode.
- The number of operands is correct, but their types or order do not match the opcode.

E2360 Invalid register combination (e.g. [BP+BX]) Compiler error

The built-in assembler detected an illegal combination of registers in an instruction.

These are valid index register combinations:

- `[BX]`
- `[BP]`
- `[SI]`
- `[DI]`
- `[BX+SI]`
- `[BX+DI]`
- `[BP+SI]`
- `[BP+DI]`

Other index register combinations are not allowed.

E2401 Invalid template argument list Compiler error

This error indicates that an illegal template argument list was found.

In a template declaration, the keyword **template** must be followed by a list of formal arguments enclosed within < and > delimiters.

E2432 'template' qualifier must name a template class or function instance'

Compiler error

When defining a template class member, the actual arguments in the template class name used as the left operand for the :: operator must match the formal arguments of the template class.

E2104 Invalid use of template keyword

Compiler error

You can only use a template class name without specifying its actual arguments inside a template definition.

Using a template class name without specifying its actual arguments outside a template definition is illegal.

E2143 Matching base class function 'function' has different dispatch number

Compiler error

If a DDVT function is declared in a derived class, the matching base class function must have the same dispatch number as the derived function.

E2144 Matching base class function 'function' is not dynamic Compiler error

If a DDVT function is declared in a derived class, the matching base class function must also be dynamic.

E2229 Member 'member' has the same name as its class

Compiler error

A static data member, enumerator, member of an anonymous union, or nested type cannot have the same name as its class.

Only a member function or a non-static member can have a name that is identical to its class.

E2234 Memory reference expected Compiler error

The built-in assembler requires a memory reference.

You probably forgot to put square brackets around an index register operand.

E2400 Nontype template argument must be of scalar type

Compiler error

A nontype formal template argument must have scalar type; it can have an integral, enumeration, or pointer type.

E2071 operator new must have an initial parameter of type size_t

E2071 Operator new[] must have an initial parameter of type size_t Compiler error

Operator new can be declared with an arbitrary number of parameters.

It must always have at least one, the amount of space to allocate.

E2396 Template argument must be a constant expression

Compiler error

A non-type template class argument must be a constant expression of the appropriate type.

This includes constant integral expressions and addresses of objects or functions with external linkage or members.

E2424 Template class nesting too deep: 'class'

Compiler error

The compiler imposes a certain limit on the level of template class nesting. This limit is usually only exceeded through a recursive template class dependency.

When this nesting limit is exceeded, the compiler issues this error message for all of the nested template classes. This usually makes it easy to spot the recursion.

This error message is always followed by the fatal error "Out of memory".

E2415 Template functions may only have 'type-arguments'

Compiler error

A function template was declared with a non-type argument.

This is not allowed with a template function, as there is no way to specify the value when calling it.

E2398 Template function argument 'argument' not used in argument types

Compiler error

The given argument was not used in the argument list of the function.

The argument list of a template function must use all of the template formal arguments; otherwise, there is no way to generate a template function instance based on actual argument types.

E2218 Templates can only be declared at namespace or class scope Compiler error

Templates cannot be declared inside classes or functions. They are only allowed in the global scope, or file level.

For example:

```
void func()
{
  template <class T> myClass {  // Error
    T i;
  };
}
```

E2428 Templates must be classes or functions Compiler error

The declaration in a template declaration must specify either a class type or a function.

E2037 The constructor 'constructor' is not allowed

Compiler error

Constructors of the form

```
  X(X);  // Error
};
```

are not allowed.

This is the correct way to write a copy constructor:

```
class X {
  X(const X&);    // Copy constructor
};
```

E2394 Too few arguments passed to template 'template' Compiler error

A template class name was missing actual values for some of its formal parameters.

E2159 Trying to derive a far class from the huge base 'base' Compiler error

This error is no longer generated by the compiler.

E2161 Trying to derive a huge class from the far base 'base'

Compiler error

This error is no longer generated by the compiler.

E2162 Trying to derive a huge class from the near base 'base' Compiler error

This error is no longer generated by the compiler.

E2164 Trying to derive a near class from the huge base 'base' Compiler error

This error is no longer generated by the compiler.

E2391 Type mismatch in parameter 'parameter' in template class name 'template'

Compiler error

The actual template argument value supplied for the given parameter did not exactly match the formal template parameter type.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is usually "Cannot convert 'type1' to 'type2'" but the mismatch might be due to many other reasons.

E2390 Type mismatch in parameter 'number' in template class name 'template'

Compiler error

The actual template argument value supplied for the given parameter did not exactly match the formal template parameter type.

When compiling C++ programs, this message is always preceded by another message that explains the exact reason for the type mismatch.

That other message is usually "Cannot convert 'type1' to 'type2'" but the mismatch might be due to many other reasons.

E2118 Bit fields must have integral type

Compiler error

In C++, bit fields must have an integral type. This includes enumerations.

E2248 Cannot find default constructor to initialize array element of type 'class'

Compiler error

When declaring an array of a class that has constructors, you must either explicitly initialize every element of the array, or the class must have a default constructor.

The compiler will define a default constructor for a class unless you have defined any constructors for the class.

E2298 Cannot generate 'function' from template function 'template'

Compiler error

A call to a template function was found, but a matching template function cannot be generated from the function template.

E2332 Variable 'variable' has been optimized and is not available

Compiler error

You have tried to inspect, watch, or otherwise access a variable which the optimizer removed.
This variable is never assigned a value and has no stack location.

W8070 Function should return a value

Compiler warning

(Command-line option to suppress warning: **-w-rvl**)

This function was declared (maybe implicitly) to return a value.

The compiler found a return statement without a return value, or it reached the end of the function without finding a return statement.

Either return a value or change the function declaration to return void.

W8073 Undefined structure 'structure'

Compiler warning

(Command-line option to display warning = **-wstu**)

Your source file used the named structure on some line before where the error is indicated (probably on a pointer to a structure) but had no definition for the structure.

This is probably caused by a misspelled structure name or a missing declaration.

W8065 Call to function 'function' with no prototype

Compiler warning

This message is given if the "Prototypes required" warning is enabled and you call function 'function' without first giving a prototype for that function.

W8006 Initializing 'identifier' with 'identifier'** Compiler warning

`(Command-line option to suppress warning: **-w-bei**)`

You're trying to initialize an **enum** variable to a different type.

For example, the following initialization will result in this warning, because 2 is of type int, not type enum count:

```
enum count zero, one, two x = 2;
```

It is better programming practice to use an enum identifier instead of a literal integer when assigning to or initializing enum types.

This is an error, but is reduced to a warning to give existing programs a chance to work.

W8061 Initialization is only partially bracketed

Compiler warning

(Command-line option to display warning: **-wpin**)

When structures are initialized, braces can be used to mark the initialization of each member of the structure. If a member itself is an array or structure, nested pairs of braces can be used. This ensures that the compiler's idea and your idea of what value goes with which member are the same. When some of the optional braces are omitted, the compiler issues this warning.

W8020 'identifier' is declared as both external and static Compiler warning

(Command-line option to suppress warning: **-w-ext**)

This identifier appeared in a declaration that implicitly or explicitly marked it as global or external, and also in a static declaration.

The identifier is taken as static.

You should review all declarations for this identifier.

W8015 Declare 'type' prior to use in prototype Compiler warning

(Command-line option to suppress warning: **-w-dpu**)

When a function prototype refers to a structure type that has not previously been declared, the declaration inside the prototype is not the same as a declaration outside the prototype.

For example,

```
int func(struct s *ps);  struct s  /* ... */ ;
```

Because there is no "struct s" in scope at the prototype for func, the type of parameter ps is pointer to undefined struct s, and is not the same as the "struct s" that is later declared.

This will result in later warning and error messages about incompatible types, which would be very mysterious without this warning message.

To fix the problem, you can move the declaration for "struct s" ahead of any prototype that references it, or add the incomplete type declaration "struct s;" ahead of any prototype that references "structs".

If the function parameter is a struct, rather than a pointer to struct, the incomplete declaration is not sufficient.

You must then place the struct declaration ahead of the prototype.

W8082 Division by zero Compiler warning

(Command-line option to suppress warning: **-w-zdi**)

A divide or remainder expression had a literal zero as a divisor.

W8000 Ambiguous operators need parentheses Compiler warning

(Command-line option to display warning: **-wamb**)

This warning is displayed whenever two shift, relational, or bitwise-Boolean operators are used together without parentheses.

Also, an addition or subtraction operator that appears without parentheses with a shift operator will produce this warning.

W8001 Superfluous & with function Compiler warning

(Command-line option to display warning: **-wamp**)

An address-of operator (&) is not needed with function name; any such operators are discarded.

W8057 Parameter 'parameter' is never used Compiler warning

(Command-line option to suppress warning: **-w-par**)

The named parameter, declared in the function, was never used in the body of the function.

This might or might not be an error and is often caused by misspelling the parameter.

This warning can also occur if the identifier is redeclared as an automatic (local) variable in the body of the function.

The parameter is masked by the automatic variable and remains unused.

W8002 Restarting compile using assembly Compiler warning

(Command-line option to suppress warning: **-w-asc**)

The compiler encountered an asm with no accompanying or **#pragma** inline statement.

The compile restarts using assembly language capabilities.

Default = On

W8003 Unknown assembler instruction

Compiler warning

(Command-line option to suppress warning: **-w-asm**)

The compiler encountered an inline assembly statement with a disallowed opcode or an unknown token. Check the spelling of the opcode or token.

Note: You will get a separate error message from the assembler if you entered illegal assembler source code.

This warning is off by default.

W8007 Hexadecimal value contains more than three digits

Compiler warning

(Command-line option to suppress warning = **-w-big**)

Under older versions of C, a hexadecimal escape sequence could contain no more than three digits.

The ANSI standard allows any number of digits to appear as long as the value fits in a byte.

This warning results when you have a long hexadecimal escape sequence with many leading zero digits (such as \x00045).

Older versions of C would interpret such a string differently.

W8009 Constant is long Compiler warning

(Command-line option to display warning: **-wcln**)

The compiler encountered one of the following:

- a decimal constant greater than 32,767 or
- an octal, hexadecimal, or decimal constant greater than 65,535 without a letter l or L following it

The constant is treated as a long.

W8011 Nonportable pointer comparison

Compiler warning

(Command-line option to suppress warning: **-w-cpt**)

Your source file compared a pointer to a non-pointer other than the constant 0.

You should use a cast to suppress this warning if the comparison is proper.

W8028 Temporary used to initialize 'identifier'

Compiler warning

(Command-line option to suppress warning: **-w-lin**)

In C++, a variable or parameter of reference type must be assigned a reference to an object of the same type.

If the types do not match, the actual value is assigned to a temporary of the correct type, and the address of the temporary is assigned to the reference variable or parameter.

The warning means that the reference variable or parameter does not refer to what you expect, but to a temporary variable, otherwise unused.

Related Topics

Example

Example for "Temporary used ..." error messages Compiler error

In this example, function f requires a reference to an int, and c is a char:

```
f(int&);
char c;
f(c);
```

Instead of calling f with the address of c, the compiler generates code equivalent to the C++ source code:

```
int X = c, f(X);
```

W8031 Temporary used for parameter 'parameter'

Compiler warning

OR W8029 Temporary used for parameter 'number'

OR W8030 Temporary used for parameter 'parameter' in call to 'function'

OR W8032 Temporary used for parameter 'number' in call to 'function'

(Command-line option to suppress warning: **-w-lvc**)

In C++, a variable or parameter of reference type must be assigned a reference to an object of the same type.

If the types do not match, the actual value is assigned to a temporary of the correct type, and the address of the temporary is assigned to the reference variable or parameter.

The warning means that the reference variable or parameter does not refer to what you expect, but to a temporary variable, otherwise unused.

W8038 constant member 'identifier' is not initialized

Compiler warning

(Command-line option to suppress warning: **-w-nci**)

This C++ class contains a constant member 'member' that doesn't have an initialization.

Note that constant members can be initialized only; they can't be assigned to.

W8027 Functions containing ‘statement’ are not expanded inline

Compiler warning

(Command-line option to suppress warning: **-w-inl**)

Where:

‘statement’ can be any of the following:

- Static variables
- Aggregate initializers
- Some return statements
- Local destructors
- Some if statements
- Local classes
- Missing return statements
- Disallowed reserved words listed under “Reserved words” below.

Reserved words

Functions containing any of these reserved words can't be expanded inline, even when specified as inline:

asm	except
break	finally
case	for
continue	goto
default	switch
do	while

The function is still perfectly legal, but will be treated as an ordinary static (not global) function.

A copy of the function will appear in each compilation unit where it is called.

Description

If an inline function becomes too complex, the compiler is unable to expand it inline. However, because the function is so complex, expanding it inline is unlikely to provide significant performance enhancements.

For local destructors

You've created an inline function for which the compiler turns off inlining. You can ignore this warning; the function will be generated out of line.

W8052 Base initialization without a class name is now obsolete

Compiler warning

(Command-line option to suppress warning: **-w-obi**)

Early versions of C++ provided for initialization of a base class by following the constructor header with just the base class constructor parameter list.

It is now recommended to include the base class name.

This makes the code much clearer, and is required when you have multiple base classes.

Old way

```
derived::derived(int i) : (i, 10) { ... }
```

New way

```
derived::derived(int i) : base(i, 10) { ... }
```

W8054 Style of function definition is now obsolete

Compiler warning

(Command-line option to suppress warning = **-w-ofp**)

In C++, this old C style of function definition is illegal:

```
int func(p1, p2) int p1, p2; { /* ... */ }
```

This practice might not be allowed by other C++ compilers.

W8018 Assigning 'type' to 'enumeration'

Compiler warning

(Command-line option to suppress warning: **-w-eas**)

Assigning an integer value to an enum type.

This is an error in C++, but is reduced to a warning to give existing programs a chance to work.

W8037 Non-const function 'function' called for const object

Compiler warning

(Command-line option to suppress warning = **-w-ncf**)

A non-const member function was called for a const object.

(This is an error, but was reduced to a warning to give existing programs a chance to work.)

W8051 Non-volatile function 'function' called for volatile object

Compiler warning

(Command-line option to suppress warning: **-w-nvf**)

In C++, a class member function was called for a volatile object of the class type, but the function was not declared with volatile following the function header. Only a volatile member function can be called for a volatile object.

For example, if you have

```
class c
{
public:
  f() volatile;
  g();
};
volatile c vcvar;
```

it is legal to call `vcvar.f()`, but not to call `vcvar.g()`.

W8022 'function1' hides virtual function 'function2'

Compiler warning

(Command-line option to suppress warning: **-w-hid**)

A virtual function in a base class is usually overridden by a declaration in a derived class.

In this case, a declaration with the same name but different argument types makes the virtual functions inaccessible to further derived classes.

W8013 Possible use of 'identifier' before definition

Compiler warning

(Command-line option to display warning: **-wdef**)

Your source file used the variable 'identifier' in an expression before it was assigned a value.

The compiler uses a simple scan of the program to determine this condition.

If the use of a variable occurs physically before any assignment, this warning will be generated.

Of course, the actual flow of the program can assign the value before the program uses it.

W8068 Constant out of range in comparison

Compiler warning

(Command-line option to suppress warning: **-w-rng**)

Your source file includes a comparison involving a constant sub-expression that was outside the range allowed by the other sub-expression's type.

For example, comparing an unsigned quantity to -1 makes no sense.

To get an unsigned constant greater than 32,767 (in decimal), you should either

- cast the constant to unsigned--for example, (unsigned) 65535, or
- append a letter u or U to the constant--for example, 65535u.

Whenever this message is issued, the compiler still generates code to do the comparison.

If this code ends up always giving the same result (such as comparing a char expression to 4000), the code will still perform the test.

W8017 Redefinition of 'x' is not identical Compiler warning

(Command-line option to suppress warning: **-w-dup**)

Your source file redefined the macro 'ident' using text that was not exactly the same as the first definition of the macro.

The new text replaces the old.

W8023 Array variable 'identifier' is near

Compiler warning

(Command-line option to suppress warning: **-w-ias**)

When you use set the Far Data Threshold option, the compiler automatically makes any global variables that are larger than the threshold size be far.

When the variable is an initialized array with an unspecified size, its total size is not known when the compiler must decide whether to make it near or far, so the compiler makes it near.

The compiler issues this warning if the number of initializers given for the array causes the total variable size to exceed the data size threshold.

If the fact that the compiler made the variable be near causes problems, make the offending variable explicitly far.

To do this, insert the keyword "far" immediately to the left of the variable name in its definition.

W8060 Possibly incorrect assignment

Compiler warning

(Command-line option to suppress warning: **-w-pia**)

This warning is generated when the compiler encounters an assignment operator as the main operator of a conditional expression (part of an if, while, or do-while statement).

This is usually a typographical error for the equality operator.

If you want to suppress this warning, enclose the assignment in parentheses and compare the whole thing to zero explicitly.

For example, this code

```
if (a = b) ...
```

should be rewritten as

```
if ((a = b) != 0) ...
```

W8071 Conversion may lose significant digits

Compiler warning

(Command-line option to display warning: **-wsig**)

For an assignment operator or some other circumstance, your source file requires a conversion from a larger integral data type to a smaller integral data type where the conversion exists.

Because the integral data type variables don't have the same size, this kind of conversion might alter the behavior of a program.

W8045 No declaration for function 'function'

Compiler warning

(Command-line option to display warning: **-wnod**)

This message is given if you call a function without first declaring that function.

In C, you can declare a function without presenting a prototype, as in

```
int func();
```

In C++, every function declaration is also a prototype; this example is equivalent to

```
int func(void);
```

The declaration can be either classic or modern (prototype) style.

W8019 Code has no effect Compiler warning

(Command-line option to suppress warning: **-w-eff**)

This warning is issued when the compiler encounters a statement with some operators that have no effect.

For example, the statement

```
a + b;
```

has no effect on either variable.

The operation is unnecessary and probably indicates a bug.

W8004 'identifier' is assigned a value that is never used Compiler warning

(Command-line option to suppress warning: **-w-aus**)

The variable appears in an assignment, but is never used anywhere else in the function just ending.

The warning is indicated only when the compiler encounters the closing brace.

The **#pragma warn -aus** switch has function-level granularity. You cannot turn off this warning for individual variables within a function; it is either off or on for the whole function.

W8025 Ill-formed pragma Compiler warning

(Command-line option to suppress warning: **-w-ill**)

A pragma does not match one of the pragmas expected by the compiler.

W8064 Call to function with no prototype Compiler warning

(Command-line option to suppress warning: **-w-pro**)

This message is given if the "Prototypes required" warning is enabled and you call a function without first giving a prototype for that function.

W8075 Suspicious pointer conversion

Compiler warning

(Command-line option to suppress warning: **-w-sus**)

The compiler encountered some conversion of a pointer that caused the pointer to point to a different type.

You should use a cast to suppress this warning if the conversion is proper.

A common cause of this warning is when the C compiler converts a function pointer of one type to another (the C++ compiler generates an error when asked to do that). It can be suppressed by doing a typecast. Here is a common occurrence of it for Windows programmers:

```
#define STRICT
#include <windows.h>

LPARAM _export WndProc( HWND , UINT , WPARAM , LPARAM );

test() {
   WNDCLASS wc;
   wc.lpfnWndProc = WndProc;  //warning
}
```

It is suppressed by making the assignment to lpfnWndProc as follows:

```
wc.lpfnWndProc = ( WNDPROC ) WndProc;
```

W8066 Unreachable code

Compiler warning

(Command-line option to suppress warning: **-w-rch**)

A break, continue, goto, or return statement was not followed by a label or the end of a loop or function.

The compiler checks while, do, and for loops with a constant test condition, and attempts to recognize loops that can't fall through.

W8067 Both return and return with a value used Compiler warning

(Command-line option to suppress warning: **-w-ret**)

The current function has return statements with and without values.

This is legal C, but almost always generates an error.

Possibly a return statement was omitted from the end of the function.

W8069 Nonportable pointer conversion

Compiler warning

(Command-line option to suppress warning: **-w-rpt**)

A nonzero integral value is used in a context where a pointer is needed or where an integral value is needed; the sizes of the integral type and pointer are the same.

Use an explicit cast if this is what you really meant to do.

W8081 Void functions may not return a value

Compiler warning

(Command-line option to suppress warning: **-w-voi**)

Your source file declared the current function as returning void, but the compiler encountered a return statement with a value. The value of the return statement will be ignored.

Example:

```
// This HAS to be in a "C" file.  In a "C++" file this would be an error
void foo()
{
        return 0;       // Can't return a value from a void function
}
```

W8074 Structure passed by value Compiler warning

(Command-line option to display warning: **-wstv**)

This warning is generated any time a structure is passed by value as an argument.

It is a frequent programming mistake to leave an address-of operator (&) off a structure when passing it as an argument.

Because structures can be passed by value, this omission is acceptable.

This warning provides a way for the compiler to warn you of this mistake.

W8079 Mixing pointers to different 'char' types Compiler warning

(Command-line option to display warning: **-wucp**)

You converted a signed char pointer to an unsigned char pointer, or vice versa, without using an explicit cast. (Strictly speaking, this is incorrect, but it is often harmless.)

W8080 'identifier' is declared but never used

Compiler warning

(Command-line option to display warning: **-wuse**)

The specified identifier was never used. This message can occur in the case of either local or static variables. It occurs when the source file declares the named local or static variable as part of the block just ending, but the variable was never used.

In the case of local variables, this warning occurs when the compiler encounters the closing brace of the compound statement or function. In the case of static variables, this warning occurs when the compiler encounters the end of the source file.

For example:

```
// Need to compile with -w to make this warning show up!
#pragma option -w
int foo()
{
        int x;
        return 0;
}
```

W8008 Condition is always true OR W8008 Condition is always false Compiler warning

(Command-line option to suppress warning: **-w-ccc**)

Whenever the compiler encounters a constant comparison that (due to the nature of the value being compared) is always true or false, it issues this warning and evaluates the condition at compile time.

For example:

```
void proc(unsigned x){
  if (x >= 0)        /* always 'true' */
  {
     ...
  }
}
```

W8016 Array size for 'delete' ignored Compiler warning

(Command-line option to suppress warning: **-w-dsz**)

The C++ IDE issues this warning when you've specified the array size when deleting an array.

With the new C++ specification, you don't need to make this specification. The compiler ignores this construct.

This warning lets older code compile.

W8024 Base class 'class1' is also a base class of 'class2'

Compiler warning

(Command-line option to suppress warning: **-w-ibc**)

A class inherits from the same base class both directly and indirectly. It is best to avoid this non-portable construct in your program code.

E2117 Bit fields must be signed or unsigned int Compiler warning

(Command-line option to display warning: **-wbbf**)

In ANSI C, bit fields may not be of type signed char or unsigned char.

When you're not compiling in strict ANSI mode, the compiler allows these constructs, but flags them with this warning.

W8063 Overloaded prefix operator 'operator' used as a postfix operator

Compiler warning

(Command-line option to suppress warning: **-w-pre**)

The C++ specification allows you to overload both the prefix and postfix versions of the ++ and -- operators.

Whenever the prefix operator is overloaded, but is used in a postfix context, the compiler uses the prefix operator and issues this warning.

This allows older code to compile.

W8048 Use qualified name to access member type 'identifier' Compiler warning

(Command-line option to suppress warning: **-w-nst**)

In previous versions of the C++ specification, **typedef** and tag names declared inside classes were directly visible in the global scope.

In the latest specification of C++, these names must be prefixed with `class::qualifier` if they are to be used outside of their class scope.

The compiler issues this warning whenever a name is uniquely defined in a single class. The compiler permits this usage without `class::`. This allows older versions of code to compile.

W8033 Conversion to 'type' will fail for members of virtual base 'class'

Compiler warning

(Command-line option to suppress warning: **-w-mpc**)

This warning is issued only if the **-Vv** option is in use.

The warning may be issued when a member pointer to one type is cast to a member pointer of another type and the class of the converted member pointer has virtual bases.

Encountering this warning means that at runtime, if the member pointer conversion cannot be completed, the result of the cast will be a NULL member pointer.

Stack overflow Runtime message

This error is reported when you compile a function with the Test Stack Overflow option on, but there is not enough stack space to allocate the function's local variables.

This error can also be caused by the following:

- infinite recursion, or
- an assembly language procedure that does not maintain the stack project
- a large array in a function

Abnormal program termination Runtime message

The program called abort because there wasn't enough memory to execute.

This message can be caused by memory overwrites.

Divide error Runtime message

You tried to divide an integer by zero, which is illegal.

Floating point error: Divide by 0

Runtime messages

OR Floating point error: Domain

OR Floating point error: Overflow

These fatal errors result from a floating-point operation for which the result is not finite:

- Divide by 0 means the result is `+INF` or `-INF` exactly, such as `1.0/0.0`.
- Domain means the result is `NAN` (not a number), like `0.0/0.0`.
- Overflow means the result is `+INF` (infinity) or `-INF` with complete loss of precision, such as assigning `1e200*1e200` to a double.

Floating point error: Partial loss of precision Runtime message

OR Floating point error: Underflow

These exceptions are masked by default, because underflows are converted to zero and losses of precision are ignored.

Floating point error: Stack fault

Runtime message

The floating-point stack has been overrun. This error may be due to assembly code using too many registers or due to a misdeclaration of a floating-point function.

The program prints the error message and calls abort and _exit.

These floating-point errors can be avoided by masking the exception so that it doesn't occur, or by catching the exception with signal.

E2111 Type 'typename' may not be defined here Compiler error

Class and enumeration types may not be defined in a function return type, a function argument type, a conversion operator type, or the type specified in a cast.

You must define the given type before using it in one of these contexts.

Note: This error message is often the result of a missing semicolon (`;`) for a class declaration. You might want to verify that all the class declarations preceding the line on which the error occurred end with a semicolon.

E2120 Cannot call 'main' from within the program Compiler error

C++ does not allow recursive calls of main( ).

Application is running Runtime message

The application you tried to run is already running.

For Windows, make sure the message loop of the program has properly terminated.

```
PostQuitMessage(0);
```

W8034 Maximum precision used for member pointer type 'type'

Compiler warning

(Command-line option to suppress warning: **-w-mpd**)

When a member pointer type is declared, its class has not been fully defined, and the **-Vmd** option has been used, the compiler has to use the most general (and the least efficient) representation for that member pointer type. This can cause less efficient code to be generated (and make the member pointer type unnecessarily large), and can also cause problems with separate compilation; see the **-Vm** compiler switch for details.

F1005 Include files nested too deep Compiler error

This message flags (directly or indirectly) recursive **#include** directives.

E2012 Cannot take address of 'main' Compiler error

In C++, it is illegal to take the address of the main function.

E2016 Ambiguous override of virtual base member 'base_function': 'derived_function' Compiler error

A virtual function in a virtual base class was overridden with two or more different functions along different paths in the inheritance hierarchy. For example,

```
struct VB
{
  virtual f();
};

struct A:virtual VB
{
  virtual f();
};

struct B:virtual VB
  virtual f();
}
```

E2018 Cannot throw 'type' -- ambiguous base class 'base' Compiler error

It is not legal to throw a class that contains more than one copy of a (non-virtual) base class.

E2057 Exception specification not allowed here Compiler error

Function pointer type declarations are not allowed to contain exception specifications.

E2058 Exception handling variable may not be used here

Compiler error

An attempt has been made to use one of the exception handling values that are restricted to particular exception handling constructs, such as GetExceptionCode().

E2091 Functions cannot return arrays or functions

Compiler error

A function was defined to return an array or a function. Check to see if either the intended return was a pointer to an array or function (and perhaps the * is missing) or if the function definition contained a request for an incorrect datatype.

E2173 Duplicate handler for 'type1', already had 'type2' Compiler error

It is not legal to specify two handlers for the same type.

E2174 The *name* handler must be last

Compiler error

In a list of catch handlers, if the specified handler is present, it must be the last handler in the list (that is, it cannot be followed by any more catch handlers).

E2178 VIRDEF name conflict for 'function' Compiler error

The compiler must truncate mangled names to a certain length because of a name length limit that is imposed by the linker. This truncation may (in very rare cases) cause two names to mangle to the same linker name. If these names happen to both be VIRDEF names, the compiler issues this error message. The simplest workaround for this problem is to change the name of 'function' so that the conflict is avoided.

E2202 Goto into an exception handler is not allowed

Compiler error

It is not legal to jump into a try block, or an exception handler that is attached to a try block.

E2252 'catch' expected Compiler error

In a C++ program, a 'try' block must be followed by at least one 'catch' block.

E2307 Type 'type' is not a defined class with virtual functions Compiler error

A dynamic_cast was used with a pointer to a class type that is either undefined, or doesn't have any virtual member functions.

E2270 > expected Compiler error

A new-style cast (for example, dynamic_cast) was found with a missing closing ">".

E2318 'type' is not a polymorphic class type

Compiler error

This error is generated if the -RT compiler option (for runtime type information) is disabled and either

- dynamic_cast was used with a pointer to a class

or

- you tried to delete a pointer to an object of a class that has a virtual destructor

E2262 '__except' or '__finally' expected following '__try' Compiler error

In C, a '__try block' must be followed by a '__except' or '__finally' handler block.

E2359 Reference member 'member' initialized with a non-reference parameter

Compiler error

An attempt has been made to bind a reference member to a constructor parameter. Since the parameter will cease to exist the moment the constructor returns to its caller, this will not work correctly.

W8036 Non-ANSI keyword used: 'keyword' Compiler warning

(Command-line option to display warning: **-wnak**)

A non-ANSI keyword (such as '__fastcall') was used when strict ANSI conformance was requested via the -A option.

W8021 Handler for 'type1' hidden by previous handler for 'type2'

Compiler warning

(Command-line option to suppress warning: **-w-hch**)

This warning is issued when a handler for a type 'D' that is derived from type 'B' is specified after a handler for B', since the handler for 'D' will never be invoked.

W8058 Cannot create pre-compiled header: 'reason'

Compiler warning

(Command-line option to suppress warning: **-w-pch**)

This warning is issued when pre-compiled headers are enabled but the compiler could not generate one, for one of the following reasons:

Reason	Explanation
write failed	The compiler could not write to the pre-compiled header file. This is usually due to the disk being full.
code in header	One of the headers contained a non-inline function body.
initialized data in header	One of the headers contained a global variable definition (in C, a global variable with an initializer; in C++ any variable not declared as 'extern').
header incomplete	The pre-compiled header ended in the middle of a declaration, for example, inside a class definition (this often happens when there is a missing"}" in a header file).

W8049 Use '> >' for nested templates Instead of '>>'

Compiler warning

(Command-line option to suppress warning: **-w-ntd**)

Whitespace is required to separate the closing ">" in a nested template name, but since it is an extremely common mistake to leave out the space, the compiler accepts a ">>" with this warning.

Initializing enumeration with type Compiler error

You're trying to initialize an enum variable to a different type. For example,

```
enum count { zero, one, two } x = 2;
```

will result in this warning, because 2 is of type int, not type enum count. It is better programming practice to use an enum identifier instead of a literal integer when assigning to or initializing enum types.

This is an error, but is reduced to a warning to give existing programs a chance to work.

E2067 Main must have a return type of int Compiler error

In C++, function main has special requirements, one of which is that it cannot be declared with any return type other than int.

W8053 'ident' is obsolete Compiler warning

(Command-line option to suppress warning: **-w-obs**)

Issues a warning upon usage for any "C" linkage function that has been specified. This will warn about functions that are "obsolete".

Here's an example of it's usage:

```
#ifdef __cplusplus
extern "C" {
#endif
void my_func(void);
#ifdef __cplusplus
}
#endif

#pragma obsolete my_func

main()
{
  my_func();   // Generates warning about obsolete function
}
```

Unexpected termination during compilation [Module Seg#:offset] Compiler error

OR Unexpected termination during linking [Module Seg#:offset]

If either of these errors occur, it indicates a catastrophic failure of the Borland tools. You should contact Inprise to report the problem and to find a potential work around for your specific case. By isolating the test case as well as possible, you will increase the chance for either Inprise or yourself to find a work around for the problem.

Commonly, compiler failures can be worked around by moving the source code that is currently being compiled. Simple cases might be switching the order of variable declarations, or functions within the source module. Moving the scope and storage of variables also helps in many cases.

For linker failures, you can reduce the amount of debugging information that the linker has to work with. Try compiling only one or two modules with debug information instead of an entire project.

Similarly, switching the order in which object modules are handed to the linker can work around the problem. The IDE hands objects to the linker in the order that they are listed in the project tree. Try moving a source up or down in the list.

F1012 Compiler stack overflow Compiler error

The compiler's stack has overflowed. This can be caused by a number of things, among them deeply nested statements in a function body (for example, if/else) or expressions with a large number of operands. You must simplify your code if this message occurs. Adding more memory to your system will not help.

E2000 286/287 instructions not enabled

Compiler error

Use the -2 command-line compiler option to enable 286/287 opcodes. Be aware that the resulting code cannot be run on 8086- and 8088-based machines.

E2075 Incorrect ‘type’ option: option Compiler error

An error has occurred in either the configuration file or a command-line option. The compiler may not have recognized the configuration file parameter as legal; check for a preceding hyphen (-), or the compiler may not have recognized the command-line parameter as legal.

This error can also occur if you use a **#pragma** option in your code with an invalid option.

E2133 Unable to execute command 'command' Compiler error

The linker or assembler cannot be found, or possibly the disk is bad.

E2195 Cannot evaluate function call Compiler error

The error message is issued if someone tries to explicitly construct an object or call a virtual function.

In integrated debugger expression evaluation, calls to certain functions (including implicit conversion functions, constructors, destructors, overloaded operators, and inline functions) are not supported.

E2198 Not a valid expression format type Compiler error

Invalid format specifier following expression in the debug evaluate or watch window. A valid format specifier is an optional repeat value followed by a format character (c, d, f[n], h, x, m, p, r, or s).

E2208 Cannot access an inactive scope

Compiler error

You have tried to evaluate or inspect a variable local to a function that is currently not active. (This is an integrated debugger expression evaluation message.)

E2216 Unable to create turboc.$ln

Compiler error

The compiler cannot create the temporary file TURBOC.$LN because it cannot access the disk or the disk is full.

E2240 Conversion of near pointer not allowed

Compiler error

A near pointer cannot be converted to a far pointer in the expression evaluation box when a program is not currently running. This is because the conversion needs the current value of DS in the user program, which doesn't exist.

E2244 'new' and 'delete' not supported

Compiler error

The integrated debugger does not support the evaluation of the new and delete operators.

E2263 Exception handling not enabled

Compiler error

A 'try' block was found with the exception handling disabled.

E2266 No file names given Compiler error

The command line contained no file names. You must specify a source file name.

E2278 Multiple base classes not supported for Delphi classes Compiler error

Delphi style classes cannot have multiple base classes.

Example:

```
struct __declspec(delphiclass) base1  {};
struct __declspec(delphiclass) base2  {};
struct derived : base1, base2         {}; // Error
```

E2286 Overloaded function resolution not supported

Compiler error

In integrated debugger expression evaluation, resolution of overloaded functions or operators is not supported, not even to take an address.

E2296 Templates not supported

Compiler error

An error has occurred while using the command-line utility H2ASH. See the online file "tsm_util.txt" for further information about this utility.

E2302 No type information

Compiler error

The integrated debugger has no type information for this variable. Ensure that you've compiled the module with debug information. If it has, the module may have been compiled by another compiler or assembler.

E2306 Virtual base classes not supported for Delphi classes Compiler error

Delphi style classes cannot be derived virtually, not even from other Delphi style classes.

Example:

```
struct __declspec(delphiclass) base  {};
struct derived : virtual base        {};    // Error
```

E2338 Overlays only supported in medium, large, and huge memory models

Compiler error

The compiler no longer issues this error.

E2362 Repeat count needs an lvalue Compiler error

The expression before the comma (,) in the Watch or Evaluate window must be an accessible region of storage. For example, expressions like this one are not valid:

```
i++,10d
x = y, 10m
```

E2366 Can't inherit non-RTTI class from RTTI base

Compiler error

OR E2367 Can't inherit RTTI class from non-RTTI base

When virtual functions are present, the RTTI attribute of all base classes must match that of the derived class.

E2382 Side effects are not allowed

Compiler error

Side effects such as assignments, ++, or -- are not allowed in the debugger watch window. A common error is to use x = y (not allowed) instead of x == y to test the equality of x and y.

E2425 'member' is not a valid template type member

Compiler error

A member of a template with some actual arguments that depend on the formal arguments of an enclosing template was found not to be a member of the specified template in a particular instance.

E2461 '%s' requires run-time initialization/finalization Compiler error

This message is issued when a global variable that is declared as __thread (a Win32-only feature) or a static data member of a template class is initialized with a non-constant initial value.

This message is also issued when a global variable that is declared as __thread (a Win32-only feature) or a static data member of a template class has the type class with constructor or destructor.

E2469 Cannot use tiny or huge memory model with Windows Compiler error

The compiler no longer issues this error.

W8014 Declaration ignored Compiler warning

`(Command-line option to suppress warning: **-w-dig**)`

An error has occurred while using the command-line utility H2ASH. See the online file "tsm_util.txt" for further information about this utility.

Default = On

W8043 Macro definition ignored

Compiler warning

(Command-line option to suppress warning: **-w-nma**)

An error has occurred while using the command-line utility H2ASH. See the online file "tsm_util.txt" for further information about this utility.

W8039 Constructor initializer list ignored Compiler warning

(Command-line option to suppress warning: **-w-ncl**)

An error has occurred while using the command-line utility H2ASH. See the online file "tsm_util.txt" for further information about this utility.

W8040 Function body ignored Compiler warning

(Command-line option to suppress warning: **-w-nfd**)

An error has occurred while using the command-line utility H2ASH. See the online file "tsm_util.txt" for further information about this utility.

W8042 Initializer for object 'x' ignored

Compiler warning

(Command-line option to suppress warning: **-w-nin**)

An error has occurred while using the command-line utility H2ASH. See the online file "tsm_util.txt" for further information about this utility.

W8026 Functions with exception specifications are not expanded inline

Compiler warning

Also: Functions taking class by value arguments are not expanded inline

(Command-line option to suppress warning: **-w-inl**)

Exception specifications are not expanded inline: Check your inline code for lines containing exception specification.

Functions taking class-by-value argument(s) are not expanded inline: When exception handling is enabled, functions that take class arguments by value cannot be expanded inline.

Note: Functions taking class parameters by reference are not subject to this restriction.

W8044 #undef directive ignored

Compiler warning

(Command-line option to suppress warning: **-w-nmu**)

An error has occurred while using the command-line utility H2ASH. See the online file "tsm_util.txt" for further information about this utility.

W8047 Declaration of static function *function* ignored Compiler warning

(Command-line option to suppress warning: **-w-nsf**)

An error has occurred while using the command-line utility H2ASH. See the online file "tsm_util.txt" for further information about this utility.

W8029 Temporary used for parameter '???' Compiler warning

(Command-line option to suppress warning: **-w-lvc**)

In C++, a variable or parameter of reference type must be assigned a reference to an object of the same type. If the types do not match, the actual value is assigned to a temporary of the correct type, and the address of the temporary is assigned to the reference variable or parameter.

The warning means that the reference variable or parameter does not refer to what you expect, but to a temporary variable, otherwise unused.

In the following example, function f requires a reference to an int, and c is a char:

```
f(int &amp);
char c;
f(c);
```

Instead of calling f with the address of c, the compiler generates code equivalent to the C++ source code:

```
int X = c, f(X);
```

W8032 Temporary used for parameter 2 in call to '???' Compiler warning

(Command-line option to suppress warning: **-w-lvc**)

In C++, a variable or parameter of reference type must be assigned a reference to an object of the same type. If the types do not match, the actual value is assigned to a temporary of the correct type, and the address of the temporary is assigned to the reference variable or parameter.

The warning means that the reference variable or parameter does not refer to what you expect, but to a temporary variable, otherwise unused.

In the following example, function f requires a reference to an int, and c is a char:

```
f(int &amp);
char c;
f(c);
```

Instead of calling f with the address of c, the compiler generates code equivalent to the C++ source code:

```
int X = c, f(X);
```

F1004 Internal compiler error Compiler error

An error occurred in the internal logic of the compiler. This error shouldn't occur in practice, but is generated in the event that a more specific error message is not available.

E2237 DPMI programs must use the large memory model

Compiler error

The compiler no longer issues this error.

E2355 Recursive template function: 'x' instantiated 'y' Compiler error

The compiler has detected a recursive template function instance. For example:

```
template<class T> void f(T x)
{
   f((T*)0);   // recursive template function!
}

void main()
{
   f(0);
}
```

The compiler issue one message for each nesting of the recursive instantiation, so it is usually obvious where the recursion has occurred. To fix a recursive template, either change the dependencies or provide a specialized version that will stop the recursion. For example, adding the following function definition to the above program will remove the endless recursion:

```
void f(int **)
{
}
```

E2353 Class 'classname' is abstract because of 'member = 0' Compiler error

This message is issued immediately after the "Cannot create instance of abstract class 'classname' error message and is intended to make it easier to figure out why a particular class is considered abstract by the compiler.

For example, consider the following example of an illegal attempt to instantiate an abstract class:

```
struct VB
{
  virtualvoid  f() = 0;
  virtualvoid  g() = 0;
  virtualvoid  h() = 0;
};
struct D1 : virtual VB
{
    void   f();
};
struct D2 : virtual VB
{
    void   h();
};
struct DD : D1, D2
{
}
  v;   // error 'DD' is an abstract class
```

The above code will cause the following two error messages:

```
Error TEST.CPP 21: Cannot create instance of abstract class 'DD'
Error TEST.CPP 21: Class 'DD' is abstract because of 'VB::g() = 0'
```

W8035 User-defined message Compiler warning

The error message for which you have requested Help is a user-defined warning.

In C++Builder code, user-defined messages are introduced by using the **#pragma** message compiler syntax.

Note: In addition to messages that you introduce with the **#pragma** message compiler syntax, user-defined warnings can be introduced by third party libraries. Should you require Help about a third party warning, please contact the vendor of the header file that issued the warning.

F1001 Internal code generator error Compiler error

An error has occurred in the internal logic of the code generator. Contact Inprise technical support.

E2035 Conversions of class to itself or base class not allowed Compiler error

You tried to define a conversion operator to the same class or a base class.

E2038 Cannot declare or define 'identifier' here Compiler error

You tried to declare a template in an illegal place or a namespace member outside of its namespace.

E2070 Invalid use of namespace 'identifier' Compiler error

A namespace identifier was used in an illegal way, for example, in an expression.

E2154 Cannot define 'identifier' using a namespace alias

Compiler error

You cannot use a namespace alias to define a namespace member outside of its namespace.

E2282 Namespace name expected Compiler error

The name of a namespace symbol was expected.

E2334 Namespace member 'identifier' declared outside its namespace

Compiler error

Namespace members must be declared inside their namespace. You can only use explicit qualification to define a namespace member (for example, to give a body for a function declared in a namespace). The declaration itself must be inside the namespace.

E2413 Invalid template declaration Compiler error

After the declarator of a template member, either a semicolon, an initialization, or a body was expected, but some other, illegal token was found. This message appears when a template member is declared outside of the template, but the syntax was wrong.

E2421 Cannot use local type 'identifier' as template argument Compiler error

A local type was used in an actual template type argument, which is illegal.

E2006 CodeGuarded programs must use the large memory model and be targeted for Windows Compiler error

The compiler no longer issues this error.

E2217 Local data exceeds segment size limit

Compiler error

The local variables in the current function take up more than 64K.

E2368 RTTI not available for expression evaluation

Compiler error

Expressions requiring RTTI are not supported by the expression evaluator in the integrated debugger. This error message is only issued by the expression evaluator (if you try to Inspect, Watch, or Evaluate), not by the compiler.

E2433 Specialization after first use of template

Compiler error

An ANSI C++ rule requires that a specialization for a function template be declared before its first use. This error message is only issued when the ANSI conformance option (-A) is active.

E2344 Earlier declaration of 'identifier'

Compiler error

This error message only shows up after the messages "Multiple declaration for 'identifier'" and "Type mismatch in redeclaration of 'identifier'". It tells you where the previous definition of the identifier in question was found by the compiler, so you don't have to search for it.

E2132 Templates and overloaded operators cannot have C linkage Compiler error

You tried to use a linkage specification with a template or overloaded operator. The most common cause for this error message is having the declaration wrapped in an extern "C" { linkage specification.

W8059 Structure packing size has changed Compiler warning

(Command-line option to suppress warning: **-w-pck**)

This warning message is issued when the structure alignment is different after including a file than it was before including that file.

The intention is to warn you about cases where an include file changes structure packing, but by mistake doesn't restore the original setting at the end. If this is intentional, you can give a `#pragma nopackwarning` directive at the end of an include file to disable the warning for this file.

The warning can be disabled altogether by `#pragma warn -pck`.

W8010 Continuation character \ found in // comment

Compiler warning

(Command-line option to suppress warning: **-w-com**)

This warning message is issued when a C++ // comment is continued onto the next line with backslash line continuation.

The intention is to warn about cases where lines containing source code unintentionally become part of a comment because that comment happened to end in a backslash.

If you get this warning, check carefully whether you intend the line after the // comment to be part of the comment. If you don't, either remove the backslash or put some other character after it. If you do, it's probably better coding style to start the next comment line with // also.

The warning can be disabled altogether with `#pragma warn -com`.

E2375 Assembler stack overflow

Compiler error

The assembler ran out of memory during compilation. Review the portion of code flagged by the error message to ensure that it uses memory correctly.

E2268 Call to undefined function 'function' Compiler error

Your source file declared the current function to return some type other than **void** in C++ (or **int** in C), but the compiler encountered a return with no value. All **int** functions are exempt in C because in old versions of C, there was no **void** type to indicate functions that return nothing.

Null pointer assignment Runtime message

When a small or medium memory model program exits, a check is made to determine if the contents of the first few bytes within the program's data segment have changed. These bytes would never be altered by a working program. If they have been changed, this message is displayed to inform you that (most likely) a value was stored to an uninitialized pointer.

The program might appear to work properly in all other respects; however, this is a serious bug which should be attended to immediately. Failure to correct an uninitialized pointer can lead to unpredictable behavior (including locking the computer up in the large, compact, and huge memory models).

You can use the integrated debugger to track down null pointers.

Pure virtual function called Runtime message

This is a runtime error. It is generated if the body of a pure virtual function was never generated and somehow the compiler tried to call it.

E2213 Invalid 'expression' in scope override

Compiler error

The evaluator issues this message when there is an error in a scope override in an expression you are watching or inspecting. You can specify a symbol table, a compilation unit, a source file name, etc. as the scope of the expression, and the message will appear whenever the compiler cannot access the symbol table, compilation unit, or whatever.

E2095 String literal not allowed in this context

Compiler error

This error message is issued by the evaluator when a string literal appears in a context other than a function call.

E2269 The function 'function' is not available

Compiler error

You tried to call a function that is known to the evaluator, but which was not present in the program being debugged, for example, an inline function.

E2236 Missing 'identifier' in scope override Compiler error

The syntax of a scope override is somehow incomplete. The evaluator issues this message.

E2196 Cannot take address of member function 'function'

Compiler error

An expression takes the address of a class member function, but this member function was not found in the program being debugged. The evaluator issues this message.

E2124 Invalid function call

Compiler error

A requested function call failed because the function is not available in the program, a parameter cannot be evaluated, and so on. The evaluator issues this message.

Printf/Scanf floating-point formats not linked

Runtime message

Floating-point formats contain formatting information that is used to manipulate floating-point numbers in certain runtime library functions, such as scanf() and atof(). Typically, you should avoid linking the floating-point formats (which take up about 1K) unless they are required by your application. However, you must explicitly link the floating-point formats for programs that manipulate fields in a limited and specific way.

Refer to the following list of potential causes (listed from most common to least common) to determine how to resolve this error:

- **CAUSE**: Floating point set to None. You set the floating-point option to None when it should be set to either Fast or Normal.

  FIX: Set Floating Point to Fast or Normal.

- **CAUSE**: Either the compiler is over-optimizing or the floating-point formats really do need to be linked. You need the floating-point formats if your program manipulates floats in a limited and specific way. Under certain conditions, the compiler will ignore floating-point usage in scanf(). For example, this may occur when trying to read data into a float variable that is part of an array contained in a structure.

  FIX: Add the following code to one source module:

  ```
  extern _floatconvert;
  #pragma extref _floatconvert
  ```

- **CAUSE**: You forgot to put the address operator & on the scanf variable expression. For example:

  ```
  float foo;
  scanf("%f", foo);
  ```

  FIX: Change the code so that the & operator is used where needed. For example, change the above code to the following:

  ```
  float foo;
  scanf("%f", &foo);
  ```

E2323 Illegal number suffix Compiler error

A numeric literal is followed by a suffix that is not recognized by the compiler.

Example:

```
int i = 1234i15;   // Error: no i15 suffix
int j = 1234i16;   // OK
```

E2130 Circular property definition Compiler error

Indicates that a property definition relies directly or indirectly on itself.

Example:

```
struct pbase
{
     int __property ip1 = {read = ip2, write = ip2};
     int __property ip2 = {read = ip1, write = ip1};
};
```

The above code sample will cause this error message on any usage of ip1 or ip2.

E2346 ‘x’ access specifier of property ‘property’ must be a member function

Compiler error

Only member functions or data members are allowed in access specifications of properties.

Example:

```
int GlobalGetter(void)
{
     return 0;
}

struct pbase
{
     int MemberGetter(void) {return 1;}

     int __property ip1 = { read = GlobalGetter }; // Error
     int __property ip2 = { read = MemberGetter }; // OK
};
```

E2347 Parameter mismatch in access specifier ‘specifier’ of property ‘property’

Compiler error

The parameters of the member function used to access a property don’t match the expected parameters.

Example:

```
struct pbase
{
        void Setter1(void)      {}
        void Setter2(int){}

     int __property ip1 = { write = Setter1 }; // Error
     int __property ip2 = { write = Setter2 }; // OK
};
```

E2348 Storage specifier not allowed for array properties Compiler error

Array properties cannot have a storage specification.

Example:

```
struct pbase
{
    int __property ap[char *] =
                           { stored = false }; // Error
};
```

E2459 Delphi style classes must be constructed using operator new Compiler error

Delphi style classes cannot be statically defined. They have to be constructed on the heap.

Example:

```
voidfoo(void)
{
     Tobject   o1;     // Error;
     Tobject  *o2 = new TObject();
}
```

E2457 Delphi style classes must be caught by reference Compiler error

You can only catch a Delphi style object by pointer.

Example:

```
void foo(TObject *p)
{
     try
     {
          throw(p);
     }

     catch    (TObject o)    // Error
     {
     }

     catch  (TObject *op)    // OK
     {
     }
}
```

E2458 Delphi classes have to be derived from Delphi classes Compiler error

You cannot derive a Delphi style class from a non-Delphi style class.

Example:

```
struct base // base not a Delphi style class
{
  int  basemem;
};

struct __declspec(delphiclass) derived : base // or
{
  int  derivedmem;
};
```

E2242 *Specifier* requires Delphi style class type Compiler error

The stored, default, and nodefault storage specifiers are only allowed within property declarations of Delphi style class types.

Example:

```
struct regclass
{
     int __property ip1 = { stored  = false }; // Error
     int __property ip2 = { default = 42    }; // Error
     int __property ip3 = { nodefault       }; // Error
};

struct __declspec(delphiclass) vclclass
{
     int __property ip1 = { stored  = false }; // OK
     int __property ip2 = { default = 42    }; // OK
     int __property ip3 = { nodefault       }; // OK
};
```

E2460 Delphi style classes require exception handling to be enabled Compiler error

If you are using Delphi style classes in your program, you cannot turn off exception handling (compiler option **-x-**) when compiling your source code.

E2281 __classid requires definition of (TClass) as a pointer type

Compiler error

To use __classid, there needs to be a definition for *TClass* which can be found in vcl.h.

Example:

```
// #include <vcl/vcl.h> missing

struct __declspec(delphiclass)bar
{
     virtual int barbara(void);
};

void *foo(void)
{
     return classid(bar);    // Error
}
```

E2289 __published or __automated sections only supported for Delphi classes

Compiler error

The compiler needs to generate a special kind of vtable for classes containing __published and __automated sections. Therefore, these sections are only supported for Delphi style classes.

Example:

```
struct regclass
{
  int  mem;
__published:      // Error: no Delphi style class
  int __property ip = { read = mem, write = mem };
};

struct __declspec(delphiclass) vclclass
{
  int  mem;
__published:      // OK
  int __property ip = { read = mem, write = mem };
};
```

E2351 Static data members not allowed in __published or __automated sections

Compiler error

Only nonstatic data members and member functions are allowed in __published or __automated sections.

Example:

```
struct __declspec(delphiclass) vclclass
{
__published:
  static int      staticDataMember; // Error
};
```

E2205 Illegal type *type* in __automated section

Compiler error

Only certain types are allowed in __automated sections.

Example:

```
struct __declspec(delphiclass) vclclass
{
__automated:
  int  __fastcall fooInt(int);      // OK
  long __fastcall fooLong(long);    // Error: long illegal
};
```

E2003 Data member definition not allowed in __automated section

Compiler error

Only member function declarations are allowed in __automated sections.

Example:

```
struct __declspec(delphiclass) vclclass
{
__automated:
  int  __fastcall fooInt(int);      // OK
  int  memInt;                      // Error
};
```

E2002 Only __fastcall functions allowed in __automated section

Compiler error

The calling convention for functions declared in an __automated section must be __fastcall.

Example:

```
struct __declspec(delphiclass) vclclass
{
__automated:
  int  __fastcall fooInt(int);      // OK
  int  __cdecl    barInt(int);      // Error
};
```

E2001 Constructors and destructors not allowed in __automated section

Compiler error

Only member function declarations are allowed in __automated sections.

Example:

```
struct __declspec(delphiclass) vclclass
{
__automated:
  int  __fastcall fooInt(int);      // OK
  vclclass()  {}                    // Error
};
```

E2005 Redeclaration of property not allowed in __automated section Compiler error

If you declare a property in an __automated section it has be a new declaration. Property hoisting is not allowed.

Example:

```
struct __declspec(delphiclass) vclbaseclass
{
  int  __fastcall Get(void);
  void   __fastcall Set(int);
  int __property ip1 = { read = Get, write = Set };
};

struct vclderivedclass : vclbaseclass
{
  int  __fastcall NewGetter(void);
__automated:
  __property ip1; // Error
  int __property ip2 = { read = Get, write = Set }; // OK
};
```

E2004 Only read or write clause allowed in property declaration in __automated section

Compiler error

Storage specifiers stored, default, and nodefault are not allowed in property declarations in __automated sections.

Example:

```
struct __declspec(delphiclass) vclclass
{
  int  __fastcall Get(void);
__automated:
  int __property ip1 = { read = Get };                // OK
  int __property ip2 = { read = Get, default = 42 }; // Error
};
```

E2180 Dispid *number* already used by *identifier* Compiler error

Dispids must be unique and the compiler checks for this.

Example:

```
struct __declspec(delphiclass) vclclass
{
__automated:
  int  __fastcall foo1(void) __dispid(42);      // OK
  int  __fastcall foo2(void) __dispid(42);      // Error
};
```

E2007 Dispid only allowed in __automated sections

Compiler error

The definition of dispids is only permitted in __automated sections.

Example:

```
struct __declspec(delphiclass) vclclass
{
  int  __fastcall foo1(void) __dispid(42);      // Error
__automated:
  int  __fastcall foo2(void) __dispid(43);      // OK
};
```

W8072 Suspicious pointer arithmetic Compiler warning

This message indicates an unintended side effect to the pointer arithmetic (or array indexing) found in an expression.

Example:

```
#pragma warn +spa

int array[10];

int foo(__int64 index)
{
  return array[index];
}
```

The value of index is 64 bits wide while the address of array is only 32 bits wide.

W8050 No type OBJ file present. Disabling external types option. Compiler warning

(Command-line option to suppress warning: **-w-nto**)

A precompiled header file references a type object file, but the type object file cannot be found. This is not a fatal problem but will make your object files larger than necessary.

W8012 Comparing signed and unsigned values Compiler warning

(Command-line option to suppress warning: **-w-csu**)

Since the ranges of signed and unsigned types are different the result of an ordered comparison of an unsigned and a signed value might have an unexpected result.

Example:

```
#pragma warn +csu

bool foo(unsigned u, int i)
{
  return u < i;
}
```

W8041 Negating unsigned value

Compiler warning

(Command-line option to suppress warning: **-w-ngu**)

Basically, it makes no sense to negate an unsigned value because the result will still be unsigned.

Example:

```
#pragma warn +ngu

unsigned foo(unsigned u)
{
  return -u;
}
```

W8078 Throw expression violates exception specification

Compiler warning

(Command-line option to suppress warning: **-w-thr**)

This warning happens when you add an exception specification to a function definition and you throw a type in your function body that is not mentioned in your exception specification.

The following program would generate this warning:

```
int foo() throw(char*)  // I promise to only throw char*s
{
    throw 5;  // Oops, I threw an integer
    return 0;
}
```

E2370 Simple type name expected Compiler error

To ensure interoperability between Delphi and C++Builder, there are restrictions on the type names mentioned in the parameter lists of published closure types. The parameter types have to be simple type names with optional const modifier and pointer or reference notation.

So when declaring a closure type, the arguments passed to that closure must be of a simple type. For example, templates are not accepted. To pass a reference to an object of template type to a closure, you must declare a typedef, which counts as a simple type name.

Example:

```
struct __declspec(delphiclass) foo
{
     typedef void __fastcall (__closure *foo1)(SomeTemplateType<int> *);

     typedef SomeTemplateType<int> SimpleTypeName;
     typedef void __fastcall (__closure *foo2)(SimpleTypeName *);

published:

     __property foo1 prop1;  // Error
     __property foo2 prop2;  // OK
};
```

E2122 Function call terminated by unhandled exception 'value' at address 'addr'

Compiler error

This message is emitted when an expression you are evaluating while debugging includes a function call that terminates with an unhandled exception. For example, if in the debugger's evaluate dialog, you request an evaluation of the expression `foo()+1` and the execution of the function `foo()` causes a GP fault, this evaluation produces the above error message.

You may also see this message in the watches window because it also displays the results of evaluating an expression.

E2241 VCL style classes need virtual destructors Compiler error

Destructors defined in VCL style classes have to be virtual.

Example:

```
struct __declspec(delphiclass) vclclass1
{
  ~vclclass1() {}           // Error
};

struct __declspec(delphiclass) vclclass2
{
  virtual ~vclclass2() {}   // OK
};
```

E2249 = expected Compiler error

The compiler expected an equal sign in the position where the error was reported but there was none. This is usually a syntax error or typo.

E2008 Published property access functions must use __fastcall calling convention Compiler error

The calling convention for access functions of a property (read, write, and stored) declared in a __published section must be __fastcall. This also applies to hoisted properties.

Example:

```
struct __declspec(delphiclass) vclclass
  {
    int  __fastcall Getter1(void);
    int  __cdecl    Getter2(void);
  __published:
    int __property ip1 = {read = Getter1};    // OK
    int __property ip2 = {read = Getter2};    // Error
  };
```

E2273 'main' cannot be declared as static or inline

Compiler error

You cannot make main static or inline. For example, you cannot use `static int main()` or `inline int main()`.

E2112 Unknown unit directive: 'directive'

Compiler error

You cannot use this name as a unit directive. Instead use one of the following unit directives: weak, smart_init, or deny.

E2177 Redeclaration of #pragma package with different arguments Compiler error

You can have multiple #pragma package statements in a source file as long as they have the same arguments. This error occurs if the compiler encounters more than one #pragma package with different arguments in each.

E2032 Illegal use of closure pointer Compiler error

A closure pointer variable is used incorrectly. Closure variables have limited usage. For instance, you can assign a function to a closure variable, and execute that function through the closure variable, but you cannot use a closure variable like a pointer variable.

E2093 Operator 'operator' not implemented in type 'type' for arguments of the same type Compiler error

The operator you are calling is not defined in this class. When you have an expression: x + x, where x is of type class X, the operator + has to be defined in class X and be accessible.

E2094 Operator 'operator' not implemented in type 'type' for arguments of type 'type'** Compiler error

The operator you are calling is not defined in this class. When you have an expression: x + x, where x is of type class X, the operator + has to be defined in class X and be accessible.

E2419 Error while instantiating template 'template'

Compiler error

An error occurred during the instantiation of a particular template. This message always follows some other error message that indicates what actually went wrong. This message is displayed to help track down which template instantiation introduced the problem.

E2106 Explicit specialization must be used with a template class or function

Compiler error

The explicit specialization operator template<> can only be used in front of a template class or function. Using it with a normal class means nothing, and hence generates an error.

E2099 Explicit specialization only allowed at file or namespace scope

Compiler error

The explicit specialization operator template<> can only be used within global or namespace scope. It cannot be used to qualify a local class or a class member, for example.

E2098 Explicit specialization declarator "template<>" now required Compiler error

When specializing a function, such as providing the definition for "foo<int>", so that foo behaves specially which called for the "int" argument, now requires that the declaration begin with an explicit specialization operator.

E2103 Explicit instantiation must be used with a template class or function

Compiler error

The explicit instantiation operator "template" can only be used to refer to templates. It cannot be used with non-templates.

E2097 Explicit instantiation only allowed at file or namespace scope Compiler error

The explicit instantiation operator "template" can only be used within global or namespace scope. It cannot be used to qualify a local class or a class member, for example.

E2101 'export' keyword must precede a template declaration Compiler error

The 'export' keyword can only occur before the keyword "template" ina template declaration. It cannot be used anywhere else.

E2049 Class type 'type' cannot be marked as __declspec(delphireturn)

Compiler error

Classes marked as delphireturn are special classes that the compiler needs to recognize by name. These classes are predefined in the headers.

Some of the delphireturn classes are Variant, AnsiString, and Currency.

You cannot mark user-defined classes as delphireturn.

E2065 Using namespace symbol 'symbol' conflicts with intrinsic of the same name Compiler error

If you define a function in a namespace, which has a name that might be replaced by a call to an intrinsic when -Oi is on, it is not permitted to have a "using" declaration which refers to that member.

For example, calls to "strcmp" are replaced by the intrinsic "__strcmp__" when -Oi is on. This means that the declaration "using N::strcmp;" would become "using N::__strcmp__", since the token replacement happens before the compiler's parser ever sees the tokens.

An error displays in this case, because the compiler doesn't know how to process "N::__strcmp__".

E2052 Dynamic function 'function' conflicts with base class 'class'

Compiler error

Some of the modifiers of this dynamic function conflict with the definition of the same function in the base class. The two functions should have the same modifiers. The following modifiers (among others) can cause conflicts:

- `__export`
- `__import`
- `declspec(naked)`
- `declspec(package)`
- `__fastcall`

E2369 Cannot use the result of a property assignment as an rvalue' Compiler error

The result of a property assignment is an lvalue. This implies for instance that chained assignments of properties is not allowed; for example, x = y = 5, where both x and y are properties. Certain embedded assignments of properties can also produce errors; for example, x != (y = z), where y is a property.

E2157 Deleting an object requires exactly one conversion to pointer operator

Compiler error

If a person uses the 'delete' operator on an object (note: not a pointer to an object, but an object itself), the standard requires that object to define exactly one "conversion to pointer operator" which will yield the pointer that gets freed. For example:

```
char *a = new char[10];
class foo {
public:
    operator char *() { return a; }
};

int main() {
    delete[] x;
}
```

Since 'x' is not a pointer, but an object, the compiler will delete 'a', because that is what the pointer conversion operator for the object yields. Having more than one conversion to pointer operator is illegal, because the compiler would not know which one to call.

E2315 'Member' is not a member of 'class', because the type is not yet defined

Compiler error

The member is being referenced while the class has not been fully defined yet. This can happen if you forward declare class X, declare a pointer variable to X, and reference a member through that pointer; for example:

```
class X;
X * oneX;
int test() { return oneX->i; }
```

E2261 Use of dispid with a property requires a getter or setter Compiler error

This property needs either a getter or a setter.

E2301 Cannot use templates in closure arguments -- use a typedef

Compiler error

When declaring a closure type, the arguments passed to that closure must be of a simple type. Templates are not accepted. To pass a reference to an object of template type to a closure, you must declare a typedef, which counts as a simple type name.

Example:

```
typedef my_class<int> mci;

typedef void (__closure * func) (const mci& object);
```

E2253 Calling convention must be attributed to the function type, not the closure

Compiler error

The calling convention is in the wrong place in the closure declaration. For example,

```
int __fastcall (__closure * x)()
```

will compile, but

```
int (__fastcall __closure * x)()
```

will not.

E2073 Nothing allowed after pragma option pop Compiler error

The #pragma option pop can only be followed by comments, blanks, or end of line.

E2181 Cannot override a 'dynamic/virtual' with a 'dynamic/virtual' function

Compiler error

When you declare a function dynamic, you cannot override this function in a derived class with a virtual function of the same name and type. Similarly when the function is declared virtual, you cannot override it with a dynamic one in a derived class.

E2326 Use __declspec(spec1[, spec2]) to combine multiple __declspecs

Compiler error

When you want to use several __declspec modifiers, the compiler will complain if you don't combine them into one __declspec. For example:

```
int __declspec(__import) __declspec(delphiclass) X
```

will give an error. Use the following instead:

```
int __declspec(__import, delphiclass) X
```

W8062 Previous options and warnings not restored

Compiler warning

The compiler didn't encounter a #pragma option pop after a previous #pragma option push, or in the case of nesting, there are more occurrences of #pragma option push than of #pragma option pop.

W8046 Pragma option pop with no matching option push

Compiler warning

The compiler encountered a #pragma option pop before a previous #pragma option push, or in the case of nesting, there are more occurrences of #pragma option pop than of #pragma option push.

W8056 Integer arithmetic overflow

Compiler warning

The compiler detected an overflow condition in an integer math expression.

For example:

```
int X = 0xFFFF * 0xFFFF;
```

W8055 Possible overflow in shift operation Compiler warning

The compiler detects cases where the number of bits shifted over is larger than the number of bits in the affected variable; for example:

```
char c;  c >> 16;
```

E2331 Number of allowable option contexts exceeded Compiler error

You have interspersed too many source-code option changes (using #pragma option) between template declarations. For example:

```
#pragma option -x
template<class T> class foo1 { };
#pragma option -a3
template<class T> class foo2 { };
#pragma option -b
template<class T> class foo3 { };
#pragma option -k-
```

You need to break your source code into smaller files.

E2328 Classes with properties cannot be copied by value

Compiler error

This error occurs if you attempt to use the default assignment operator. For example, the following code generates this error given two labels on a form:

```
*Label1->Font = *Label2->Font;
```

E2066 Invalid MOM inheritance Compiler error

The compiler issues this error if the currently compiled class doesn't have the same MOM (Microsoft Object Model) related flags set as its direct parent.

E2476 Cannot overload 'function'

Compiler error

You cannot overload the specified function. This error is displayed if you tried to declare a function with the same name as another function, but the redeclaration is not legal. For example, if both functions have the 'extern "C"' linkage type, only one 'extern "C"' function can have a given name.

E2050 __declspec(delphireturn) class 'class' must have exactly one data member

Compiler error

This is an internal compiler error. A class marked as a delphireturn class has more than one non-static data member.

E2074 Value after -g or -j should be between 0 and 255 inclusive

Compiler error

Both the -g and the -j command line options can be followed by an optional number. The compiler expects this number to be between 0 and 255 inclusive.

E2191 '__far16' may only be used with '__pascal' or '__cdecl' Compiler error

This is an internal compiler error. The compiler emits this message if the keyword __far16 is mixed with one of the keywords __pascal or __cdecl, all in the same declaration.

E2230 In-line data member initialization requires an integral constant expression

Compiler error

Static const class members, which are initialized in the body of the class, have to be initialized with a constant expression of integral type.

E2246 x is not abstract public single inheritance class hierarchy with no data

Compiler error

Internal compiler error. In some cases, the compiler will enforce restrictions on a class hierarchy. In this case, the restrictions would be that all classes are abstract classes, and all classes only have one parent.

E2267 First base must be VCL class Compiler error

Internal compiler error. In some cases, the compiler will enforce restrictions on a class hierarchy. In this case, the restrictions would be that the first parent of a class is a Delphi style class.

E2427 'main' cannot be a template function Compiler error

'main' cannot be declared as a template function. 'main' is the entry point of a console application, and it should be declared as a regular __cdecl function.

This error message should not occur because it has been replaced with another one (E2475).

2462 'virtual' can only be used with non-template member functions Compiler error

The 'virtual' keyword can only be applied to regular member functions, not to member template functions.

Consider a test case with the following code:

```
template <class T>
class myTemplateClass
{
  virtual int func1();                    // This is fine
  template <class T> virtual int func2(); // This causes an error
};

class myClass
{
  virtual int func1();                    // This is fine
  template <class T> virtual int func2(); // This causes an error
};
```

E2470 Need to include header <typeinfo> to use typeid Compiler error

When you use the 'typeid' function, you have to include the <typeinfo> header, otherwise you will get syntax errors.

For example, consider a test case with the following code:

```
int func()
{
  char * name = typeid(int).name();  // This causes an error
}
```

E2471 pragma checkoption failed: options are not as expected Compiler error

You can use #pragma checkoption to check that certain switches are in the state that you expect. If #pragma checkoption detects that a switch is not in the expected state, the compiler displays this error.

You can use the following syntax:

```
#pragma checkoption <options>
```

For example:

```
#pragma checkoption -O2
```

OR

```
#pragma checkoption -C -O2
```

The compiler will check if the option(s) are turned on. If all are turned on, nothing happens. If at least one is not turned on, this error is displayed.

E2474 'function' cannot be declared as static or inline Compiler error

You attempted to declare a symbol as static or inline and this type of symbol cannot be defined as static or inline. Certain functions, like 'main' and 'WinMain' cannot be declared static or inline. 'main' is the entrypoint of console applications, and 'WinMain' is the entry point of Windows applications.

For example, this error is displayed in the following cases:

```
static int main()  // This causes an error
{}
```

or

```
inline int main() { return 0; }
```

E2475 'function' cannot be a template function

Compiler error

Certain functions, like 'main' and 'WinMain' cannot be declared as a template function. 'main' is the entrypoint of console applications, and 'WinMain' is the entry point of Windows applications.

For example:

```
template <class T> int main()  // This causes an error
{}
```

See Function Templates for more information.

W8083 Pragma pack pop with no matching pack push

Compiler warning

Each #pragma pack(pop) should have a matching preceding #pragma pack(push) in the same translation unit. Pairs of 'push' and 'pop' can be nested.

For example:

```
#pragma pack( push )
#pragma pack( push )
#pragma pack( pop )
#pragma pack( pop )
#pragma pack( pop )   // This causes an error
```

W8084 Suggest parentheses to clarify precedence

Compiler warning

This warning indicates that several operators used in one expression might cause confusion about the applicable operator precedence rules. The warning helps create code that is more easy to understand and potentially less ambiguous.

For example, compile the following code using the **–w** command line option:

```
int j, k, l;
int main()
{

  return j < k & l;  // This causes an error
}
//
```

E2100 Invalid template declarator list Compiler error

It is illegal for a declarator list to follow a template class declaration. For example:

```
template<class T>
class foo {
} object_name;  // This causes an error
```

E2102 Cannot use template 'template' without specifying specialization parameters Compiler error

The generic form of a template must be referenced using specialization parameters. For example, for a template class named foo, taking two template parameters, then a legal reference might have the form

```
foo<int, char>
```

Referring to the template as just foo is legal in only two circumstances:

- When passing the template name as a template template argument
- While declaring the members of that class template, to refer to the enclosing template type

For example:

```
template<class T>
class foo
{
public:
    foo();       // legal use of bare template name
    foo& operator=(const foo&);
};

foo<foo> x;   // error: not a template template argument

foo y;    // error: needs specialization parameters
```

E2105 'template' qualifier must specify a member template name Compiler error

When parsing code that depends in some way upon a template parameter, it is sometimes impossible to know whether a member name will resolve to a template function name, or a regular parameter. In the following code, a ‘template’ qualifier is required in order to know if the ‘<’ (less-then) operator should be parsed as the beginning character of a template argument list, or as a regular less-than operator:

```
template<class T>
void foo(T a)
{
  a.member<10>();
}
```

Although it may be apparent to the reader what is meant, the compiler does not know that “member” refers to a member template function, and it will parse the line of code as follows:

```
a.member < (10>());
```

In order to tell the compiler that the less-than character begins a template argument list, the ‘template’ qualifier is needed:

```
a.template member<10>();   // "member" must be a member template
```

If the ‘template’ qualifier is used in a situation where “member” does not resolve to a member template, the above error will result.

E2107 Invalid use of template 'template' Compiler error

This error results when attempting to use a template template parameter in any way other than to reference a template specialization, or to pass that parameter in turn as a template template argument to another template. For example:

```
template<template<class T> class U>
class foo;

template<template<class T> class U>
class bar
{
  U x;      // error: not a specialization
  U<U> y;   // ok: used as a specialization, and as a
            // template template argument
  U<bar> z; // ok: used to reference a specialization
};
```

E2295 Too many candidate template specializations from 'specifier' Compiler error

When reference a class template specialization, it is possible that more than one possible candidate might result from a single reference. This can only really happen among class partial specializations, when more than one partial specialization is contending for a possible match:

```
template<class T, class U>
class foo;
template<class T>
class foo<T, T *>;
template<class T>
class foo<T *, T>;

foo<int *, int *> x;  // error: which partial specialization to use?
```

In this example, both partial specializations are equally valid, and neither is more specialized than the other, so an error would result.

E2299 Cannot generate template specialization from 'specifier' Compiler error

This error will result if an attempt is made to reference a template class or function in a manner which yields no possible candidate specializations. For example:

```
template<class T>
class foo;

foo<10> x;       // error: arguments aren't valid for 'foo'
```

E2300 Could not generate a specialization matching type for 'specifier'

Compiler error

This error is no longer generated by the compiler.

E2386 Cannot involve parameter 'parameter' in a complex partial specialization expression Compiler error

When declaring or defining a template class partial specialization, it is illegal to involve any of the non-type template parameters in complex expressions. They may only be referenced by name. For example:

```
template<class T, int U>
class foo;

template<int U>
class foo<char, U * 3>; // error: "U * 3" is a complex expression
template<int U>
class foo<char, U>;     // OK: "U" is a simple, by-name expression
```

E2387 Partial specializations may not specialize dependent non-type parameters ('parameter')

Compiler error

A partial specialization may not use a template parameter in its specialization argument list which is dependent on another type parameter. For example:

```
template<class T, int U>
class foo;

template<class T, T U>
class foo<T *, U>;      // error: ‘U’ is type-dependent
```

E2388 Argument list of specialization cannot be identical to the parameter list of primary template Compiler error

When declaring a partial specialization, its specialization argument list must differ in some way from its basic parameter list. For example:

```
template<class T>
class foo;

template<class T>
class foo<T *>;     // OK: is more specialized than primary template

template<class T>
class foo<T>;       // error: identical to primary template
```

E2389 Mismatch in kind of substitution argument and template parameter 'parameter' Compiler error

When referencing a template specialization, all type parameters must be satisfied using type arguments, all non-type parameters require non-type arguments, and all template template parameters require either a template name, or another template template argument. Mismatching these requirements in any way will trigger the above error. For example:

```
template<class T, int U, template<class V> class W>
class foo;

foo<char, 10, foo> x;    // OK: all parameter kinds match
foo<10, char, int> y;    // error: no parameter kinds match at all!
```

E2392 Template instance 'template' is already instantiated

Compiler error

There are two ways to trigger this error. If –A is enabled (ANSI compliant mode), then attempting to explicitly instantiate a template specialization which has already been instantiated (either implicitly or explicitly) will cause this error. Regardless of –A, attempting to explicitly specialize a template specialization which has already been either implicit or explicitly instantiated will always trigger this error. For example:

```
template<class T>
class foo;

foo<char> x;     // causes implicit instantiation of "foo<char>"

template<>
class foo<char> { };   // error: "foo<char>" already instantiated

template class foo<char>;   // error in –A mode, otherwise a warning
```

E2393 Cannot take the address of non-type, non-reference template parameter 'parameter' Compiler error

A template parameter has no address, and is not associated with any real “object”. Therefore, to take its address, or attempt to assign to it, has no meaning. For example:

```
template<int U>
void foo()
{
  int *x = &U;    // error: cannot take address of parameter
}
```

E2397 Template argument cannot have static or local linkage Compiler error

Only integral constant expressions, and the address of global variables with external linkage, may be used as template arguments. For example:

```
template<char *x>
class foo;

const char *p = "Hello";
extern char *q;

foo<p> x;   // OK: "p" is visible to the outside
foo<q> y;   // OK: “q" is also globally visible
foo<"Hello"> z;   // error: string literal has static linkage
```

E2399 Cannot reference template argument 'arg' in template class 'class' this way

Compiler error

The compiler no longer generates this error.

E2402 Illegal base class type: formal type 'type' resolves to 'type'

Compiler error

When instantiating a template class definition, if it is found that a declared base class does not resolve to an accessible class type, this error will result. For example:

```
template<class T>
class foo : public T { };

foo<int> x; // error: "int" is not a valid base class
foo<bar> y; // error: "bar" is an unknown type
```

E2403 Dependent call specifier yields non-function 'name'

Compiler error

The compiler no longer generates this error.

E2404 Dependent type qualifier 'qualifier' has no member type named 'name'

Compiler error

If a template declaration references a member of a dependent type, it is only possible to alert the user to the non-existence of this member during type instantiation for a given set of template arguments. For example:

```
template<class T>
class foo
{
    typename T::A x;     // we expect that "A" is a member type
};

struct bar { };

foo<bar> y; // error: "bar" has no member type named "A"
```

E2405 Dependent template reference 'identifier' yields non-template symbol

Compiler error

If a template specialization reference within a template declaration yields a reference to a non-template during type instantiation, the above error will result. For example:

```
template<class T>
class foo
{
    typename T::template A<int> x;  // "A" must be a member template
};

struct bar {
    struct A { };
};

foo<bar> y; // error: bar::A is a non-template class!
```

E2406 Dependent type qualifier 'qualifier' is not a class or struct type Compiler error

If a dependent name reference within a template declaration results in a non-struct member qualification at instantiation time, the above error will result. For example:

```
template<class T>
class foo
{
    typename T::A x;     // we expect that "A" is a member type
};

foo<int> y; // error: "int" cannot be qualified; not a class
```

E2407 Dependent type qualifier 'qualifier' has no member symbol named 'name'

Compiler error

If a template declaration references a member of a dependent type, it is only possible to alert the user to the non-existence of this member during type instantiation for a given set of template arguments. For example:

```
template<class T>
class foo
{
    foo(int *a = T::A);     // we expect that "A" is a data member
};

struct bar { };

foo<bar> y; // error: "bar" has no member named "A"
```

E2408 Default values may be specified only in primary class template declarations

Compiler error

Template functions, and class partial specializations, may not use default expressions in their template parameter lists. Only primary template declarations may do this. For example:

```
template<class T = int>
class foo;              // OK: primary class template

template<class T = int>
void bar();             // error: template function

template<class T = int>
class foo<T *>;              // error: partial specialization
```

E2409 Cannot find a valid specialization for 'specifier'

Compiler error

This error is no longer generated by the compiler.

E2410 Missing template parameters for friend template 'template'

Compiler error

If a friend template is declared, but no template parameters are specified, this error will result. For example:

```
template<class T>
class foo;

class bar {
    friend class foo;        // error: forgot template parameters!
};
```

E2411 Declaration of member function default parameters after a specialization has already been expanded Compiler error

If a member function of a class template is declared, and then a specialization of that class implicitly instantiated, and later that member function defined with default parameters specified, the above error will result. For example:

```
template<int i>
class foo {
    void method(int a, int b = i);
};

foo<10> x;

template<int i>
void foo<i>::method(int a = i, int b);  // error!
```

E2412 Attempting to bind a member reference to a dependent type

Compiler error

The compiler no longer generates this error.

E2414 Destructors cannot be declared as template functions Compiler error

Destructors cannot be templates. For example:

```
class foo {
    template<class T>
    virtual ~foo();          // error: don't try this at home!
};
```

E2416 Invalid template function declaration Compiler error

The compiler no longer generates this error.

E2417 Cannot specify template parameters in explicit specialization of 'specifier'

Compiler error

The compiler no longer generates this error.

E2418 Maximum instantiation depth exceeded; check for recursion Compiler error

The compiler only supports 256 levels of instantiation before it will trigger this error. The main problem is in controlling stack depth, because the parser uses recursive functions to manage type instantiation. Here is an example that would produce such an error:

```
template<int T>
class foo {
public:
    static const int x = foo<T – 1>::x;
};

template<int T>
class foo<1> {
public:
    static const int x = 1;
};

int main() {
    int y = foo<100000>::x;  // error: instantiation depth exceeded
}
```

E2420 Explicit instantiation can only be used at global scope Compiler error

Explicit instantiation cannot be specified at any level other than namespace or global scope. For example:

```
template<class T>
class foo { };

template class foo<char>;     // OK: at global scope

int main() {
    template class foo<int>: // error: local scope
}
```

E2422 Argument kind mismatch in redeclaration of template parameter 'parameter' Compiler error

If a template is declared at one point in the translation unit, and then redeclared with template parameters of a different kind at another location, this error will result. For example:

```
template<class T>
class foo;

// … time passes …

template<int T>
class foo;        // error: type vs. non-type parameter
```

E2423 Explicit specialization or instantiation of non-existing template 'template'

Compiler error

Attempting to explicit specialize or instantiate a template which does not exist is clearly illegal. For example:

```
template<class T>
class foo;

template class bar<char>;     // error: what is "bar"??

template<>
class bar<int> { };           // error: there's that "bar" again…
```

E2426 Explicit specialization of 'specifier' requires 'template<>' declaration

Compiler error

According to the standard, explicit specialization of any template now always require the “template<>” declarator syntax. For example:

```
template<class T>
class foo;

template<>
class foo<char>;        // OK: "template<>" was provided

class foo<int>;         // error: "template<>" required
```

E2429 Not a valid partial specialization of 'specifier' Compiler error

Internal compiler error.

E2430 Number of template parameters does not match in redeclaration of 'specifier' Compiler error

If a template is redeclared with a different number of template parameters, this error will result. For example:

```
template<class T>
class foo;

template<class T, int U>
class foo;              // error: parameter count mismatch!
```

E2431 Non-type template parameters cannot be of floating point, class, or void type Compiler error

Non-type template parameters are restricted as to what type they may be. Floating point, class and void types are illegal. For example:

```
template<float U>
class foo;              // error: "U" cannot be of "float" type
```

E2434 Template declaration missing template parameters ('template<...>')

Compiler error

In a context where at least one template parameter is clearly required, if none are found this error will result. For example:

```
template<class T, template<> class U>
class foo;  // error: template template parameters require
            // at least one actual parameter to be declared
```

E2435 Too many template parameter sets were specified

Compiler error

If a member template is being defined outside of its parent class, and too many template parameter sets are declared, this error will result. For example:

```
template<class T>
class foo
{
    template<class U>
    void method(U a);
};

template<class T> template<class U> template<class V>
void foo<T>::method(U a);     // error: too many parameter sets!
```

E2436 Default type for template template argument 'arg' does not name a primary template class Compiler error

If a template template parameter is to have a default type, that type must either be a generic template class name, or another template template parameter.

```
template<class T>
class foo;

template<template<class T> class U = foo>
class bar; // OK: "foo" is a qualifying primary template

template<template<class T> class U = int>
class baz; // error: "int" is not a template class
```

E2437 'typename' should be followed by a qualified, dependent type name

Compiler error

Whenever the “typename” keyword is used in a template declaration or definition, it should always name a dependent type. For example:

```
struct bar { };

template<class T>
class foo {
    typename T::A *x;        // OK: names a qualified type
    typename T y;            // error: not a qualified type
    typename bar z;          // error: not a dependent type
};
```

E2438 Template template arguments must name a class Compiler error

A template template parameter must always declare a new class name. For example:

```
template<template<class T> int U>
class foo;  // error: "U" is not a class tag name

template<template<class T> class V>
class bar;  // OK: "V" is a class tag name
```

E2439 'typename' is only allowed in template declarations

Compiler error

The “typename” keyword must only be used within template declarations and definitions.

E2440 Cannot generate specialization from 'specifier' because that type is not yet defined Compiler error

The compiler no longer generates this error.

E2441 Instantiating 'specifier' Compiler error

Whenever a compiler error occurs while instantiating a template type, the context of what was being instantiated at that point in time will be reported to the user, in order to aid in detection of the problem.

E2473 Invalid explicit specialization of 'specifier' Compiler error

Attempting to explicitly specialize a static data member or any non-template will cause this error.

E2478 Too many template parameters were declared for template 'template'

Compiler error

If a member declaration or definition occurs outside of a template class, and that outer declaration uses a different number of template parameters than the parent class, this error will result. For example:

```
template<class T, class U>
class foo {
    void method();
};

template<class T, class U, class V>
void foo<T, U, V>::method() { }    // error: too many parameters!
```

E2479 Cannot have both a template class and function named 'name'

Compiler error

No other function or type may have the same name as a template class. For example:

```
void foo(); // error: there is a template class named "foo"

template<class T>
class foo;
```

E2480 Cannot involve template parameters in complex partial specialization arguments Compiler error

A partial specialization cannot reference other template parameters in a nonvalue argument expression, unless it is simply a direct reference to the template parameter. For example:

```
template<int A, int B, int C> class foo;
template<int A> class foo<A+5, A, A+10>;
```

The partial specialization has two illegal arguments. 'A+5' is a complex expression because it uses 'A' in a manner other than as merely a direct argument. The reference to plain 'A' in the second argument is fine, but the third argument is also illegal because it references 'A' in a complex manner as well.

E2481 Unexpected string constant Compiler error

There are times when the compiler does not expect a string constant to appear in the source input. For example:

```
class foo { "Hello"; };
```

E2482 String constant expected

Compiler error

The compiler expected a string constant at this location but did not receive one.

This error is no longer generated by the compiler.

E2483 Array dimension 'specifier' could not be determined

Compiler error

If, during instantiation of a type, an array dimension cannot be computed—usually this is due to some other error which would be reported—then this error will result.

For example, if an array dimension is dependent upon a template parameter but an error occurs while it is being parsed and the template argument being substituted does not yield a legal constant expression, this error is displayed.

E2484 The name of template class 'class' cannot be overloaded

Compiler error

Attempting to declare a function that overrides the name of a template class will cause this error. For example:

```
template<class T>
class foo;

void foo(); // error: there is a template class named "foo"
```

E2485 Cannot use address of array element as non-type template argument

Compiler error

Non-type template arguments may only be of integral type, or the address of a global variable. They cannot be the address of an array element. For example:

```
int x[100];

template<int T>
class foo;

foo<&x[0]> y;          // error: not an integral or global address
```

E2486 Cannot use address of class member as non-type template argument

Compiler error

Non-type template arguments may only be of integral type, or the address of a global variable. They cannot be the address of a class member. For example:

```
struct bar {
    int x;
} y;

template<int T>
class foo;

foo<&y.x> z;            // error: not an integral or global address
```

W8076 Template instance 'specifier' is already instantiated

Compiler warning

You are trying to explicitly instantiate a template that was already implicitly instantiated.

If –A is not enabled and an attempt is made to explicitly instantiate a specialization which has already been either implicitly or explicitly instantiated, this error will result.

W8077 Explicitly specializing an explicitly specialized class member makes no sense

Compiler warning

Internal error. This warning is no longer generated by the compiler.

The following code is illegal:

```
template<class T> class foo { int x; }
template<> class foo<int> { int y; }
template<> int foo<int>::y;   // error: cannot explicitly specialize of
  'foo<int>::x'
```

W8085 Function 'function' redefined as non-inline Compiler warning

This warning is used to indicate when a certain function, which has been declared inline in one location, is redefined in another location to be non-inline.

E2472 Cannot declare a member function via instantiation

Compiler error

If a declaration within a template class acquires a function type through a type dependent on a template-parameter and this results in a declaration that does not use the syntactic form of a function declarator to have function type, the program is ill-formed. For example:

```
template<class T>
struct A {
    static T t;
};

typedef int function();

A<function> a;          // error: would declare A<function>::t
                  // as a static member function
```

Another example:

In the example below, the template member 'a' has type 'T'. If the template is instantiated with T as a function type, it implies that 'a' is therefore a member function. This is not allowed and the error message is displayed.

```
template<T& x> class foo { T a; }
int func(int);
template class foo<func>;
```

E2477 Too few template parameters were declared for template 'template'

Compiler error

If a member declaration or definition occurs outside of a template class, and that outer declaration uses a different number of template parameters than the parent class, this error will result. For example:

```
template<class T, class U>
class foo {
    void method();
};

template<class T>
void foo<T>::method() { }     // error: too few template parameters!
```

E2487 Cannot specify default function arguments for explicit specializations

Compiler error

An explicit specialization of a function may not declare default function arguments. For example:

```
template<class T>
void foo(T a);

template<>
void foo<int>(int a = 10);    // error: default value not allowed
```

E2488 Maximum token reply depth exceeded; check for recursion

Compiler error

If this error is triggered, it means that recursive template instantiation has gone too deep. Check for compile-time recursion in your program, and limit it to no more than 256 levels.

E2489 Maximum option context replay depth exceeded; check for recursion

Compiler error

If this error is triggered, it means that recursive template instantiation has gone too deep. Check for compile-time recursion in your program, and limit it to no more than 256 levels.

E2490 Specialization within template classes not yet implemented Compiler error

Explicit and partial specialization of member template classes and functions within template classes and nested template classes, is not supported.

E2491 Maximum VIRDEF count exceeded; check for recursion Compiler error

Too many VIRDEF symbols were allocated. The compiler imposes a limit to the number of VIRDEFs allowed per translation unit. Currently this limit is in the order of 16384 VIRDEFs.

One way this could happen is if a program has more than 16384 functions.

E2492 Properties may only be assigned using a simple statement, e.g. \"prop = value;\"

Compiler error

Assignments to properties should be made in simple assignment statements. If property assignments could become Lvalues, which happens when property assignments are embedded in larger statements, the getter is called to create the Lvalue, with all the side effects that getter causes. The compiler allows only one call to either the getter or the setter in a statement.

For example:

```
class myClass
{
  int X;
  public:
  int __property x = { read=getx, write=putx };
  int getx() { return X; }
  void putx(int val) { X = val; }
} OneClass;

int value(int);

int main()
{
  return value(OneClass.x = 4);  // This causes an error
}
```

E2493 Invalid GUID string Compiler error

The GUID string does not have the form of a Globally Unique Identifier.

E2494 Unrecognized __declspec modifier

Compiler error

A `__declspec` modifier was given that is not valid.

E2495 Redefinition of uuid is not identical Compiler error

GUID's attached to structs have to be the same across multiple declarations and definitions of the same struct. So the following example would cause this error:

```
class __declspec(uuid("19a76fe0-7494-11d0-8816-00a0c903b83c")) foo;
class __declspec(uuid("00000000-7494-11d0-8816-00a0c903b83c")) foo{}
```

E2496 Invalid call to uuidof(struct type|variable)

Compiler error

The uuidof operator was given an incorrect argument.

E2497 No GUID associated with type:'type' Compiler error

A variable or type was used in a context requiring a GUID, but the type does not have a GUID associated with it. GUIDs are associated with types using `__declspec(uuid(GUID))`.

E2498 Need previously defined struct GUID Compiler error

This happens when you use the __uuidof operator without including a header that defines the GUID struct. So the following program code would display this error:

```
class __declspec(uuid("19a76fe0-7494-11d0-8816-00a0c903b83c")) foo{};

int main()
{
    __uuidof(foo);
    return 0;
}
```

And you would fix it as follows:

```
#include <windows.h>    // Will pull in struct GUID
class __declspec(uuid("19a76fe0-7494-11d0-8816-00a0c903b83c")) foo{};

int main()
{
    __uuidof(foo);
    return 0;
}
```

E2499 Invalid __declspec(uuid(GuidString)) format

Compiler error

This error happens when you used the wrong format to define your GuidString. GUIDs are defined for structs in the following way:

```
class __declspec(uuid("19a76fe0-7494-11d0-8816-00a0c903b83c")) foo{};
```

You would get the above mentioned error for instance from:

```
class __declspec(uuid(19a76fe0-7494-11d0-8816-00a0c903b83c)) foo{};
  //Missing quotes
```

or

```
class __declspec(uuid"7494-11d0-8816-00a0c903b83c")) foo{}; // Missing left
  parentheses
```

E2500 __declspec(selectany) is only for initialized and externally visible variables Compiler error

You cannot use `__declspec(selectany)` with static variables, unitialized variables, etc.

E2501 Unable to open import file 'filename' Compiler error

This error occurs when you use:

```
#import "somefile.h"
```

and the file you are trying to import doesn't exist or can't be found by the compiler.

E2502 Error resolving #import: problem Compiler error

Where **problem** can be any of the following relating to problems with the various attributes of the **#import** directive:

`unexpected import directive value attribute ‘attribute’`
A value was supplied for the indicated attribute. None was expected.

`missing ')' in import directive attribute ‘attribute’`
The value for the indicated attribute was incorrectly specified : a closing parenthesis is missing.

`unrecognized import directive attribute ‘attribute’`
The indicated token is not a legitimate attribute for the #import directive.

`invalid values for raw_property_prefixes attribute`
The values for the raw_property_prefixes attribute were incorrectly specified.

`unexpected duplicate property ‘property’`
The indicated #import attribute was specified more than once -- this is an error.

`unexpected duplicate get method for property ‘property’`
The get method for the indicated property was specified more than once.

`unexpected duplicate put method for property ‘property’`
The put method for the indicated property was specified more than once.

`unexpected duplicate put-reference method for property ‘property’`
The put-reference method for the indicated property was specified more than once.

`no return value specified for property get method ‘method’`
The indicated property get method does not supply the correct return type.

`no return value specified for property put method ‘method’`
The indicated property put method does not supply the correct return type.

`could not load type library in ‘filename’`
The indicated type library could not be opened.

`could not obtain type library name`
The compiler could not obtain obtain a library name for the current type library

E2503 Missing or incorrect version of TypeLibImport.dll Compiler error

This error occurs when the compiler is trying to access TypeLibImport.dll but it either can’t find it, it was corrupted, or you have the wrong version of it installed on your computer. You can reinstall it from the C++Builder CD.

E2504 'dynamic' can only be used with non-template member functions

Compiler error

You tried to use dynamic with a template member function. Dynamic functions are allowed for classes derived from TObject. Dynamic functions occupy a slot in every object that defines them, not in any descendants. That is, dynamic functions are virtual functions stored in sparse virtual tables. If you call a dynamic function, and that function is not defined in your object, the virtual tables of its ancestors are searched until the function is found.

E2505 Explicit instantiation requires an elaborated type specifier (i.e.,"class foo<int>") Compiler error

The following code is illegal:

```
template<class T> class foo;
template foo<int>;            // missing `class' keyword
```

See Implicit and Explicit Template Functions for more information.

E2506 Explicit specialization of 'specifier' is ambiguous: must specify template arguments Compiler error

In the following code, explicit template arguments are necessary:

```
template<class T> void foo(T);
template<class T> void foo(T *);
template<> void foo(int *); // error, must say 'foo<int>' or 'foo<int *>'
```

E2507 'class' is not a direct base class of 'class' Compiler error

The first type is not a direct base class of the second type. A direct base class refers to the immediate derivations of that class, and not the derivations of its subclasses.

W8086 Incorrect use of #pragma alias "aliasName"="substituteName"

Compiler warning

The directive #pragma alias is used to tell the linker that two identifier names are equivalent. You must put the two names in quotes.

You will receive this warning if you don't use pragma alias correctly. For example, the following two lines both generate this warning:

```
#pragma alias foo=bar
#pragma alias "foo" = bar
```

W8087 'operator::operator==' must be publicly visible to be contained by a 'type'

Compiler warning

A type that is being used with an STL container has a private 'operator=='.

W8089 'type::operator<' must be publicly visible to be contained by a 'type'

Compiler warning

The type that is being used for an STL container has a private 'operator<'. The type that is being contained (type::operator) must be a public type.

For example, if you were trying to instantiate a class type "vector<blah>", the error would be:

```
'blah::operator<' must be publicly visible to be contained by a 'vector'
```

W8090 'type::operator<' must be publicly visible to be used with 'type'

Compiler warning

A type that is being used with an STL container has a private 'operator<'. The type you're trying to use must be made public.

W8091 ‘type’ argument ‘specifier’ passed to 'function' is a ‘iterator category’ iterator: ‘iterator category’ iterator required Compiler warning

An incorrect iterator category is being used with an STL algorithm.

W8092 ‘type’ argument ‘specifier’ passed to 'function' is not an iterator: ‘type’ iterator required Compiler warning

An argument that is not an iterator is being used with an STL algorithm that requires an iterator.

W8093 Incorrect use of #pragma codeseg [seg_name] ["seg_class"] [group]

Compiler warning

The #pragma codeseg directive can be used to set or reset the name, class, and group of a segment. You have to follow the exact syntax mentioned in the warning message, and all names are optional.

So these are all legal:

```
#pragma codeseg
#pragma codeseg foo
#pragma codeseg foo "bar"
#pragma codeseg foo "bar" foobar
```

But these are not:

```
#pragma codeseg ###
#pragma codeseg "foo" "bar"
#pragma codeseg foo "bar" foobar morefoobar
```

W8094 Incorrect use of #pragma comment(<type> [,"string"]) Compiler warning

The directive #pragma comment can be used to emit linker comment records.

In this message, <type> can be any of the following:

- user
- lib
- exestr
- linker

The type should be there but the string is optional.

W8095 Incorrect use of #pragma message("string")

Compiler warning

You can use pragma message to emit a message to the command line or to the message window. You would get this warning if you use the incorrect syntax, so

```
#pragma message( "hi there" )
```

is correct, but the following code would generate the warning:

```
#pragma message( badly formed )
```

W8096 Incorrect use of #pragma code_seg(["seg_name"[,"seg_class"]])

Compiler warning

Pragma code_seg is similar to pragma codeseg, but with this one you can only modify the name and the class of a code segment. If you use the wrong syntax, you get this warning.

The following examples show the correct usage:

```
#pragma code_seg()
#pragma code_seg("foo")
#pragma code_seg("foo", "bar")
```

However, the following incorrect examples will all produce the warning:

```
#pragma code_seg(foo)
#pragma code_seg(foo, bar)
```

W8097 Not all options can be restored at this time Compiler warning

Your program has a #pragma pop at a place where it can't restore options.

For example:

```
#pragma option push -v
int main()
{
  int i;
  i = 1;
#pragma option pop
  return i;
}
```

For this example, compile with -v-. The message happens because the first #pragma causes debug info to change state (turns it on). Then, in the middle of the function where it is useless to toggle the debug info state, the #pragma pop attempts to return to the former state.

Exceeded memory limit for block *address* in module *module*

The linker has run out of memory.

Solutions

You can try reducing size of active RAM disks and/or disk caches.

Close one or more applications to free memory.

Malloc of *number* bytes failed in *module*, line *number*

The linker has run out of memory.

Solutions

You can try reducing size of active RAM disks and/or disk caches.

Close one or more applications to free memory.

Realloc of *number* bytes failed in *module*, line *number*

The linker has run out of memory.

Solutions

You can try reducing size of active RAM disks and/or disk caches.

Close one or more applications to free memory.

Out of memory

The linker has run out of dynamically allocated memory needed during the link process. The total working storage is exhausted.

This error is a catchall for running into a limit on memory usage. This usually means that the object files being linked together have defined too many modules, externals, groups, or segments.

Solutions

You can try reducing size of active RAM disks and/or disk caches.

Close one or more applications to free memory.

Assertion failed: *module* at "*address*\", line *number*

This is an internal error. You should note the circumstances under which this message occurred and inform Inprise Technical Support.

Access violation. Link terminated.

You should note the circumstances under which this message occurred and inform Inprise Technical Support.

Illegal *type* fixup index in module *module*

The object file contains an invalid fixup index. This can result if the compiler emits the wrong the index. It can also happen if the object file is corrupted. Try to recompile that object file. If this message still persists, contact Inprise Technical Support.

Illegal fixup type *type* at offset *address* in module *module*

The fixup in the object file has an invalid type. This can happen if the object file is corrupted. Try to recompile that object file. If this message still persists, contact Inprise Technical Support.

User break. Link aborted.

You typed Ctrl+Break while in the linker. The link was aborted. (This is not an error, just a confirmation.)

Out of disk space

Your disk is full. Check to ensure that you have write access to the disk.

Cannot write to disk

Writing to the specified disk failed. Check to see if the disk is full.

Additional segments need to be defined in a .def file

A module processed by ILINK32 contained multiple user-defined code segments or user-defined data segments of different classes. Multiple segments must be defined in a .def file.

Invalid object file

The contents of the object file is not valid OMF format.

Hints:

- Check that the file is a .OBJ file.
- Check that the file is not a COFF format .OBJ file.
- The file may have become corrupted. Try rebuilding it.

Unresolved external *symbol* referenced from *module*

The named symbol is referenced in the given module but is not defined anywhere in the set of object files and libraries included in the link. Check to make sure the symbol is spelled correctly.

You will usually see this error from the linker for C or C++ symbols if any of the following occur:

- You did not properly match a symbol's declarations of **__pascal** and **__cdecl** types in different source files.
- You have omitted the name of an .OBJ file your program needs.
- You did not link in the emulation library.

If you are linking C++ code with C modules, you might have forgotten to wrap C external declarations in extern "C".

You could also have a case mismatch between two symbols.

Attempt to export non-public symbol *symbol*

A symbol name was listed in the EXPORTS section of the module definition file, but no symbol of this name was found as public in the modules linked.

If compiling in C++Builder, this is usually caused by the name mangling that occurs as a result of C++Builder type safe linkage. Inserting the **_export** keyword in the function prototype and function definition is required for all Windows callback functions.

Language-independent causes result from a mistake in spelling or case, case-sensitive exports, or a procedure with this name that was not defined.

If you are using case-sensitive exports, the Pascal calling convention used by Windows requires these symbols to be all uppercase characters.

Public symbol *symbol* defined in both module *module1* and *module2*

There is a conflict between two public symbols. This means that a symbol is defined in two modules.

Error processing module *module*

The incremental linker is unable to process the module named in the error message.

Unable to open file *filename*

The named file can't be found, possibly because it does not exist or is misspelled, or it resides in a different directory than those being searched.

If you are using the IDE, make sure you have set the appropriate directory paths in the Options | Directories dialog box.

RLINK32 was not initialized

The RLINK32.DLL failed to initialize. The .DLL is either corrupted or missing.

Could not load RLINK32.DLL

The linker was not able to load the specified DLL. Check to make sure that the DLL is in a directory that is in your path.

Could not get procedure address from RLINK32.DLL

A procedural export is missing from RLINK32.DLL. You don't have the right version of RLINK32.DLL in the C++Builder BIN directory. You need to delete that version of the DLL and replace it with the correct version from the C++Builder source. You can either copy it from another system or reinstall C++Builder.

Incompatible version of RLINK32.DLL

You don't have the right version of RLINK32.DLL in the C++Builder BIN directory. You need to delete that version of the DLL and replace it with the correct version from the C++Builder source. You can either copy it from another system or reinstall C++Builder.

Unknown RLINK32 error

Error processing resources (general error).Try using Build All to recompile the application. If you still receive this message, note the circumstances under which this message occurred and inform Inprise Technical Support.

Could not open *filename* (error code *number*)

The incremental linker was unable to open *filename*.

Failed to read from *filename* at offset *offset* for *n* bytes

The linker could not read the file *filename.* The file is probably corrupt. Try rebuilding it.

Could not create *filename* (error code *number*)

The operating system failed to create the file *filename.* Check that there is not an existing file of that name.

Could not write to *filename* (error code *number*)

The operating system was unable to write to the file *filename.* Check that there is not an existing file of that name with read-only attributes.

Could not open *filename* (program still running?)

The operating system was unable to open *filename.* Is the program you are trying to build still running?

Could not open *filename* (project already open in IDE?)

The operating system was unable to open *filename.* The program you are trying to build is open in the IDE.

Failed to create state file *filename* (error code *number*)

The operating system was unable to create the state file *filename* in which the linker saves information from previous links.

Symbol *symbol* marked as __import in *module* is public in *module*

Incremental linker message not used anymore.

This happens if you compile one object module using imports (-D_RTLDLL, for example), and another module using static binding (no such compilation flag). The result is that one object will expect global variables within the module to require an indirection in the assembly code (because it is an import), while the other object will expect to reference the data directly.

The solution is to either compile all object modules to using imports, or compile them all to link statically.

Unable to perform incremental linkperforming full link...

The linker detected an error in one of its state files (*projectname*.IL*) so it started a new link and created a new set of state files.

State invalid: *module* at "*address*\", line line

Internal linker error. Call Inprise Technical Support if you receive this error.

module contains invalid OMF record, type 0x*HH*

The object file is corrupt. Regenerate it, or contact Inprise Technical Support.

General error in link set

An unhandled exception occurred in the linker. Inform Inprise Technical Support of the circumstances.

Error parsing .DEF file

An error occurred while parsing the DEF file. Check the DEF file syntax.

Unsupported 16-bit segment(s) in module *module*

You cannot link 16-bit segments into 32-bit applications. The only way you can code 16-bit segments is using the assembler (tasm32.exe).

No object file or .EXE name was given

You have not specified an object file or executable file name for the incremental linker.

If you are working at the command line, check that a comma is not missing in the call to the linker.

Type index *number* bad in module *module*

Debug information or type information could be corrupt. Try deleting the *projectname*.TDS, and the compiler-generated type index files (*.x??) by default located in the LIB directory.

LIN%s: Last line *lineno* (*hex*) is less than first line *lineno* (*hex*) for symbol \

Inconsistent line numbers in debug information. This warning indicates an internal incremental linker problem. Contact Inprise Technical Support.

LIN%s: First line *lineno* (*hex*) is less than previous last line *lineno* (*hex*) for symbol \

Inconsistent line numbers in debug information. This warning indicates an internal incremental linker problem. Contact Inprise Technical Support.

Failed to create resource file *filename*

The operating system could not create the resource file. Check that there is not an existing read-only file of the same name.

Weak package unit *module* cannot contain inits

Incremental linker error that is not used at this time.

The unit name *unit* is redefined by module *module* (original definition in *module*)

Incremental linker error that is not used at this time.

Creating a package without any units

You are attempting to create a package that contains no units. The Contains list identifies the unit files to be bound into the package. Check the Contains list to be sure the associated units are listed.

Packages must be linked with the startup code in C0PKG32.OBJ

You've included the wrong startup code for a package.

Could not find a 'main unit'; initialization order will follow link order

Incremental linker error that is not used at this time.

There can only be one 'main unit' per project

Incremental linker error that is not used at this time.

Cannot reserve virtual memory at addr *address* for *n* bytes (errcode *errnumber*)

Incremental linker failed to allocate virtual memory. Try rebuilding the project.

Cannot release virtual memory at addr *address* for *n* bytes (errcode *errnumber*)

Incremental linker failed to release virtual memory. Try rebuilding the project.

Unit *unit* (defined by *module*) depends on unit *unit*, but no implementation was found

Incremental linker error that is not used at this time.

Too many section/segment definitions found in .def file

The incremental linker currently allows up to 8 extra sections/segments to be defined. You need to reduce the amount of section/segments to a maximum of 8.

Section *section* defined in .def file is empty

The segment/section specified in the .DEF file is void of code or data. Remove this segment/section declaration from the .DEF file.

Cannot have In-memory executable without state files

You have selected two mutually exclusive options: In-memory EXE and Don’t generate state files. Uncheck one of these options (choose Project|Options, click the Linker tab and uncheck one option.)

module contains invalid OMF record, type 0x%02x (possibly COFF)

The object file is corrupt or is not an OMF object file. If linking a COFF import library, try running the utility COFF2OMF.EXE, located in the BIN directory.

Error detected

The linker detected an unknown error.

Could not load imagehlp.dll. The –Gz option requires imagehlp

The linker could not load imagehlp.dll, which is required for using the –Gz option. The imagehlp.dll is a helper DLL from Microsoft that comes with Windows NT. If you cannot get the DLL, try using different command-line linker options.

Unit *unit* is marked with deny package

The unit cannot be linked into a package. Change the source code so it can be linked or eliminate this unit from the package.

Cannot build DLL or package with in-memory executable option

The linker could not build a DLL or package with the in-memory EXE option specified on the Linker Project Options page. Choose Project|Options, click the Linker tab and uncheck that option.

Cannot delay load of module

You tried to delay load a Kernel32.DLL. You cannot delay load system DLLs.

Export *symbol* in module *module* references *symbol* in unit *unit*

You are attempting to export a symbol from a module that is not a unit (does not contain the #pragma package(smart_init) directive) and it references a symbol in a unit. This is not allowed because if you have such a symbol, then someone can link to its import; and when the import is called, it calls into the unit code. If the client of the exported non-unit function did not reference anything from the unit, it will never be initialized. To fix this error, include the #pragma package(smart_init) directive in the non-unit.

Symbol *symbol* from delayed load module *module* is a data reference

You cannot delay load a module (DLL) from which you are also importing data.

Exports *symbol* and *symbol* both have the same ordinal: *number*

You are trying to export two symbols using the same ordinal number. Check the DEF file for duplicate ordinals.

Could not delete *item* (project already open in IDE?)

The linker could not delete the specified item.

Fixup to empty segment in module *module*

A fixup was detected to a segment that is of zero length. Segment length cannot be zero.

Heap reserve size is less than the commit size

You need more space than you have allocated for your application.

Symbol ‘symbol1’ is aliased to ‘symbol2,’ which is already aliased

The linker does not support alias chains. You are restricted to a single alias hop. If you attempt to chain, the linker displays this error message indicating that a given symbol you are using as the alias target is already aliased.

Note This error message only appears the first time you attempt to double alias a symbol. On subsequent incremental links, the error will not be detected, potentially leaving you with unresolved external messages, should you reference the second aliased symbol.

Delayed load module *module* was not found

A module specified as a delayed module was not found.

Public symbol for EXPDEF *symbol* not found in module *module*

An EXPDEF without a corresponding public was encountered in the specified module.

RTL helper function *function* not found

The linker could not locate specified the helper function required to use the RTL.

Stack reserve size is less than the commit size

You tried to commit more stack space than you've allocated. Increase the stack reserve space or decrease the commit size.

Could not strip resources from ‘target’

The linker was unable to strip resources from the specified target. The file may be open.

Too many exports; only 65535 permitted

Your EXE, DLL, or package is trying to export more than 65535 items. Don’t export so many items.

Bad alignment factor: *symbol*

Fatal error. The specified symbol has a bad alignment factor. The alignment specified must be a power of 2.

Illegal number format: *symbol*

Fatal error. The specified symbol uses an illegal number format.

Illegal option: *option*

Fatal error. You tried to use a linker option that is not applicable to your project or does not exist. You have specified an illegal switch or subswitch most likely on the command line.

Comma not allowed here: *location*

Fatal error. There's a problem with your linker command specifications on the command line, ILINK32.CFG file, or response file. There's a comma in the wrong place. Commas are not allowed in the response file.

Expected a file name: *identifier*

Fatal error. There's a problem with your linker command specifications on the command line, ILINK32.CFG file, or response file.

Expected an option: *identifier*

Fatal error. There's a problem with your linker command specifications on the command line, ILINK32.CFG file, or response file.

Expected a ':' or '=': *identifier*

Fatal error. There's a problem with your linker command specifications on the command line, ILINK32.CFG file, or response file. Switches that take arguments must use a : or = as a separator. Check options with arguments in the ILINK32.CFG file or response file or on the command line.

File alignment value is bigger than section alignment

Fatal error. You cannot set the file alignment so that it is larger than the section alignment. Use the **–Af** switch to set the file alignment smaller or the **–Ao** switch to set the section alignment larger.

Too many commas on command line: *identifier*

Fatal error. There's a problem with your linker command specifications on the command line. You've got too many commas or options specified.

Too many DEF file names: *identifier*

Fatal error. There's a problem with your linker command specifications. You probably forgot a comma and the linker is interpreting the result as specifying more than one DEF file name.

Too many errors; stopping link

Fatal error. Your program has exceeded the maximum number of errors. By default, the maximum number of errors is 0. You can change it on the Linker page of the Project Options to any number between 0 and 255.

Too many EXE file names: *identifier*

Fatal error. There's a problem with your linker command specifications. You probably forgot a comma.

Too many MAP file names: *identifier*

Fatal error. There's a problem with your linker command specifications. You probably forgot a comma.

Too many returns in response file: *filename*

Fatal error. There's a problem with the linker command specifications in your response file. Either edit it or regenerate it using the Make utility.

Unable to load DLL *filename*

-wuld command-line equivalent

Displayed by default. This warning occurs if the linker encounters the /B (image base) switch when linking a DLL. In almost every case, this error will prevent the application from running.

Image linked as an executable, but with a .DLL or .BPL extension

-wdee command-line equivalent

Displayed by default. The linker generates this warning when an executable file has been generated and stored in a file with a .DLL or .BPL extension. This usually occurs when you intended to build a .DLL or .BPL but forgot to specify a .DLL or .BPL target.

Public symbol *symbol* defined in both library module *module1* and *module2*

-wdup or **-wdpl** command-line equivalents

-wdup is displayed by default; **-wdpl** is not displayed by default. This warning is displayed if you have duplicate symbols in two separate libraries.

There is special handling for **-wdup** and **-wdpl**. **-wdup** can be considered the master control for warnings about duplicate publics. If this warning is disabled, no message about duplicate publics is emitted. **-wdpl** controls whether or not warnings are emitted for duplicate publics that are being linked from library modules. If **-wdpl** is disabled, and -wdup is enabled, only warnings for duplicate publics from OBJ files on the linker command line are emitted. If **-wdpl** and **-wdup** are both enabled, the linker flags duplicate publics that are in modules being linked from LIB files.

Currently, **-wdpl** is the only warning that is disabled by default. So specifying no warning options on the command line is the same as the following:

```
-w+exp -w+rty -w+dup -w-dpl -w+nou -w+srd -w+dee -w+dli -w+snf
```

Delayed loading of DLLs is not supported for in-memory images

Warning. Does not occur. No support for this feature.

Section *section* not found

-wsnf command-line equivalent

Displayed by default. This warning occurs when a named section is not found.

Stripping relocations from a DLL may cause it to malfunction

-wsrd command-line equivalent

Displayed by default. This warning occurs if the linker encounters the /B (image base) switch when linking a DLL.

Bad OMF record type 'type' encountered in module 'module'

Librarian message

The librarian encountered a bad Object Module Format (OMF) record while reading through the object module.

Because the librarian has already read and verified the header records in 'module', the object module is probably corrupt. Recreate it.

Could not allocate memory for per module data

Librarian message

The librarian has run out of memory.

Could not create list file 'filename' Librarian message

The librarian could not create a list file for the library. This could be due to lack of disk space.

Could not write output Librarian message

The librarian could not write the output file.

Error opening 'filename' Librarian message

The librarian cannot open the specified file.

Error opening 'filename' for output Librarian message

The librarian cannot open the specified file for output.

Error renaming 'filename' to 'filename'

Librarian message

This error occurs when the librarian is building a temporary library file and renaming the temporary file to the target library file name.

The error indicates that the target file is read only.

Library was too large for page size, rebuilt with page size 'size'

Librarian message

The library being created could not be built with the specified library page size. The librarian automatically increased the size and told you the new page size in a warning message.

Not enough memory for command-line buffer Librarian message

This error occurs when the librarian runs out of memory.

Bad header in input LIB Librarian message

The librarian could not understand the header record of the object module being added to the library. The librarian assumes that it is an invalid module.

Out of memory

Librarian message

For any number of reasons, the librarian or the IDE ran out of memory while building the library. For many specific cases, a more detailed message is reported.

Close one or more applications.

Out of memory creating extended dictionary

Librarian message

The librarian ran out of memory while creating an extended dictionary for a library.

The library is created but will not have an extended dictionary.

Out of memory reading LE/LIDATA record from object module Librarian message

The librarian is attempting to read a record of data from the object module, but it cannot get a large enough block of memory.

If the module being added has a large data segment or segments, try adding this module before other modules.

Out of space allocating per module debug struct Librarian message

The librarian ran out of memory while allocating space for the debug information associated with a particular object module.

Try removing debugging information from the modules being added to the library to resolve the problem.

Output device is full Librarian message

The output device is full. This error usually means that there is no space left on the disk.

Ignored 'path' - path is too long Librarian message

This error occurs when the length of any of the library file or module file's 'path' is greater then 64.

Public 'symbol' in module 'module1' clashes with prior module 'module2'

Librarian message

A public symbol can only appear once in a library file. A module, which is being added to the library, contains a public 'symbol' that is already in a module of the library and cannot be added.

The command-line message reports the module2 name.

Record kind 'num' found, expected theadr or lheadr in module 'filename'

Librarian message

The librarian could not understand the header record of the object module being added to the library and has assumed that it is an invalid module.

Record length 'len' exceeds available buffer in module 'module'

Librarian message

This error occurs when the record length 'len' exceeds the available buffer to load the buffer in module 'module'.

This occurs when the librarian runs out of dynamic memory.

The combinations '+*' or '*+' are not allowed Librarian message

It is not legal to add and extract an object module from a library in one action.

User break, library aborted Librarian message

You pressed Cancel while compiling in the IDE. The library was not created.

(This is not an error, just a confirmation.)

Added file 'filename' does not begin correctly, ignored Librarian message

The librarian has decided that the file being added is not an object module. It will not try to add it to the library.

The library is created anyway.

Cannot write GRPDEF list, extended dictionary aborted Librarian message

The librarian cannot write the extended dictionary to the end of the library file. There may not be enough space on the disk.

'filename' couldn't be created, original won't be changed

Librarian message

You tried to extract an object, but the librarian cannot create the object file into which to extract the module.

Either the object already exists and is read only, or the disk is full.

'reason' - extended dictionary not created Librarian message

OR Library contains COMDEF records - extended dictionary not created

If the Library contains COMDEF records message is displayed, an object record being added to a library contains a COMDEF record. This is not compatible with the extended dictionary option.

'filename' file not found Librarian message

The IDE creates the library by removing the existing library and rebuilding it. If any of the specified objects do not exist, the library is incomplete and this error is generated.

If the IDE reports that an object does not exist, either the source module has not been compiled or there were errors during compilation.

Invalid page size value ignored

Librarian message

The librarian encountered an invalid page size.

The page size must be an integer that is a power of 2.

Memory full listing truncated! Librarian message

The librarian ran out of memory while creating a library listing file. An incomplete list file will be created.

Results are safe in file 'filename' Librarian message

The librarian has successfully built the library into a temporary file, but it cannot rename the file to the desired library name.

The temporary file will not be removed (so that the library can be preserved).

Unknown command line switch 'X' ignored Librarian message

A forward slash character (/) was encountered on the command line or in a response file without being followed by one of the allowed options.

@ seen, expected a response-file's name Librarian message

The response file name is not given immediately after @.

Import 'symbol' in module 'module' clashes with prior module Librarian message

An import symbol can appear only once in a library file. A module that is being added to the library contains an import that is already in a module of the library and it cannot be added again.

Unexpected char X in command line Librarian message

The librarian encountered a syntactic error while parsing the command line.

Changing file buffer size

Librarian message

The librarian had to change the file buffer size to complete the operation.

'module' already in LIB, not changed! Librarian message

This warning indicates that you attempted to use the + action on the library, but an object with the same name already exists in the library. To update the module, the action should be +–. The library was not modified.

Bad GRPDEF type encountered, extended dictionary aborted Librarian message

The librarian has encountered an invalid entry in a group definition (GRPDEF) record in an object module while creating an extended dictionary.

The only type of GRPDEF record that the librarian and the linker support is segment index type. If any other type of GRPDEF is encountered, the librarian can't create an extended dictionary. It is possible that an object module created by products other than Borland tools can create GRPDEF records of other types. A corrupt object module can also generate this warning.

Bad header in input LIB Librarian message

When adding object modules to an existing library, the librarian found a bad library header. Rebuild the library.

Can't grow LE/LIDATA record buffer Librarian message

Command-line error.

The librarian is attempting to read a record of data from the object module, but it cannot get a large enough block of memory.

If the module being added has a large data segment or segments, try adding this module before other modules.

Couldn't get LE/LIDATA record buffer

Librarian message

Command-line error.

The librarian is attempting to read a record of data from the object module, but it cannot get a large enough block of memory.

If the module being added has a large data segment or segments, try adding this module before other modules.

Duplicate file 'filename' in list, not added! Librarian message

When building a library module, you specified an object file more than once.

Error changing file buffer size

Error changing file buffer size Librarian message

The librarian is attempting to adjust the size of a buffer used while reading or writing a file, but there is not enough memory. You'll need to free up a lot of system memory to resolve this error.

'filename' file not found Librarian message

Command-line error.

The command-line librarian attempted to add a nonexisting object but created the library anyway.

Ignored 'module', path is too long Librarian message

The path to a specified .OBJ or .LIB file is greater than 64 characters. The maximum path to a file for the librarian is 64 characters.

'module' not found in library Librarian message

An attempt to perform either a remove (-) or an extract (*) on a library has occurred and the indicated object does not exist in the library.

Record type 'type' found, expected theadr or lheadr in 'module' Librarian message

The librarian encountered an unexpected type instead of the expected THEADR or LHEADER record in the specified module.

Unable to open 'filename' for output Librarian message

The librarian cannot open the specified output file. This is usually due to lack of disk space for the target library, or a listing file.

Unable to rename 'filename1' to 'filename2' Librarian message

The librarian builds a library into a temporary file (filename1) and then renames the temporary file to the target library file name (filename2). This message appears if an error occurs during the renaming process, such as insufficient disk space.

Unexpected char X in command line Librarian message

The librarian encountered a syntactic error while parsing the command line.

Use /e with TLINK to obtain debug information from library

Librarian message

The library was built with an extended dictionary and also includes debugging information. TLINK cannot extract debugging information if it links using an extended dictionary.

To obtain debugging information in an executable from this library, use the /e switch to cause the linker to ignore the extended dictionary.

Note: The IDE linker does not support extended dictionaries; therefore, no settings need to be altered in the IDE.

Resource Linker error messages and warnings

Duplicate resource: Type ‘type’, ID ‘identifier’; File ‘file’ resource kept; file ‘file’ resource discarded
Duplicate string: ID ‘identifier’; File ‘file’ string kept: ‘string’; File ‘file’ resource discarded: string: ‘string’
Error creating file
Error creating temporary file
Error deleting file
Error getting size of file
Error in CURSDIR. Cannot find CURSOR
Error in FONTDIR. Cannot find FONT
Error in ICONDIR. Cannot find ICON
Error opening file
Error reading file
Error renaming file
Error seeking point in file
Error writing file
Executable format not recognized in file
FONTDIR too large to handle
Internal software error!
No resources
Out of memory!
Resource format not recognized in file
Too many files to open
Too many resources to handle

Duplicate resource: Type ‘type’, ID ‘identifier’; File ‘file’ resource kept; file ‘file’ resource discarded Resource Linker message

The resource linker encountered two resources (such as strings, menus, or icons) with the same name in two files. This message is telling you which one it used and which was discarded. You need to look at the discarded resource and if you want to retain it, you’ll need to rename it in the specified file.

Duplicate string: ID ‘identifier’; File ‘file’ string kept: ‘string’; file ‘file’ resource discarded: string: ‘string’

Resource Linker message

The resource linker encountered duplicate resource strings in two files. This message is telling you which one it used and which was discarded. You need to look at the discarded resource string and if you want to retain it, you’ll need to rename it in the specified file.

Duplicate string: ID ‘identifier’; File ‘file’ string kept: ‘string’; File ‘file’ resource discarded: string: ‘string’LCTX_RL_DUPSTR_WMSG

Error creating file. Resource Linker message

An error occurred when the resource linker tried to create a file. This error occurs if the work disk is full or write-protected. It can also occur if the output directory does not exist.

Solutions

- If the disk is full, try deleting unneeded files and restarting the resource link.
- If the disk is write-protected, direct the output to a writeable disk and restart the resource link.

Error creating temporary file

Resource Linker message

An error occurred when the resource linker tried to create a temporary file. This error occurs if the work disk is full or write-protected. It can also occur if the output directory does not exist.

Solutions

- If the disk is full, try deleting unneeded files and restarting the resource link.
- If the disk is write-protected, direct the output to a writeable disk and restart the resource link.

Error deleting file Resource Linker message

An error occurred when the resource linker tried to delete a file. This error occurs if the file is marked as read-only or does not exist.

Solutions

- If the disk is read-only, change its attributes so that it can be deleted.

Error getting size of file Resource Linker message

A disk error occurred when trying to determine the file size.

Error in CURSDIR. Cannot find CURS. Resource Linker message

An entry was found in the cursor directory that had no corresponding cursor resource. The resource file is probably corrupted.

Error in FONTDIR. Cannot find FONT. Resource Linker message

An entry was found in the font directory that had no corresponding font resource. The resource file is probably corrupted.

Error in ICONDIR. Cannot find ICON. Resource Linker message

An entry was found in the icon directory that had no corresponding icon resource. The resource file is probably corrupted.

Error opening file. Resource Linker message

An error occurred when the resource linker tried to open a file. This error occurs if the file does not exist, another process has denied access to the file, the path or filename is incorrect, or there are no more available file handles.

Error reading file. Resource Linker message

An error occurred when the resource linker tried to read a file. This error typically occurs when there is a disk error while the file is being read.

Error renaming file. Resource Linker message

An error occurred when the resource linker tried to rename a file. This error occurs if the file is marked as read-only or does not exist or a file already exists having the name that the resource linker is trying to use.

Solutions

- If the disk is read-only, change its attributes so that it can be deleted.
- If a file having the name already exists you can either delete that file or choose another name.

Error seeking point in file. Resource Linker message

An error occurred trying to seek to a location in a file. This file could be truncated or corrupted. Try compiling the resource files and restart the resource link.

Error writing file. Resource Linker message

An error occurred when the resource linker tried to write to a file. This error occurs if the work disk is full or write-protected.

Solutions

- If the disk is full, try deleting unneeded files and restarting the resource link.
- If the disk is write-protected, direct the output to a writeable disk and restart the resource link.

Executable format not recognized in file Resource Linker message

The executable file contained invalid information in its header. The file might not be a valid executable or might contain corrupted data.

FONTDIR too large to handle Resource Linker message

The directory of fonts table size has been exceeded. Try splitting your fonts into multiple FON files.

Internal software error! Resource Linker message

The resource linker encountered unexpected data. Restart the resource link. If the error persists, contact Inprise Technical Support.

No resources. Resource Linker message

This warning message occurs if the resource linker is given a resource file that contains not resources.

Out of memory! **Resource Linker message**

Not enough memory is available for compiling a particular file. In this case, shut down any other concurrent applications. You may also try to reconfigure your machine for more available memory, or break up the source file being compiled into smaller separate components. You can also compile the file on a system with more available RAM.

Resource format not recognized in file. Resource Linker message

The format of a .RES file that you are attempting to use contains a resource with an unknown format. This is normally due to a corrupt resource file. Make sure that you are binding a legitimate resource file, and rebuild the .RES file, if necessary.

Too many files to open. Resource Linker message

You have exceeded the resource linkers limit on files. Try combining some of your individual resource files into a single resource.

Too many resources to handle. Resource Linker message

You have too many resources for the resource linker to handle. Try reducing the total number of resources you are trying to link.

'filename' does not exist: don't know how to make it MAKE message

The build sequence includes a nonexistent file name, and no rule exists that would allow the file name to be built.

Unable to execute command: 'command' MAKE message

A command failed to execute. This might be because the command file could not be found or was misspelled, there was no disk space left in the specified swap directory, the swap directory does not exist, or (less likely) the command itself exists but has been corrupted.

Incorrect command line argument: 'argument' MAKE message

You've used incorrect command-line arguments. Reenter the command and arguments.

Unable to open makefile MAKE message

The current directory does not contain a file named MAKEFILE or MAKEFILE.MAK, or it does not contain the file you specified with –f.

Not enough memory MAKE message

All of your working storage has been exhausted.

Error directive: 'message' MAKE message

MAKE has processed an #error directive in the source file, and the text of the directive is displayed in the message.

Unable to redirect input or output MAKE message

MAKE was unable to open the temporary files necessary to redirect input or output. If you are on a network, make sure you have access rights to the current directory.

No terminator specified for in-line file operator MAKE message

The makefile contains either the && or << command-line operators to start an inline file, but the file is not terminated.

Circular dependency exists in makefile MAKE message

The makefile indicates that a file needs to be up-to-date before it can be built. Take, for example, the explicit rules

```
filea:  fileb
fileb:  filec
filec:  filea
```

This implies that filea depends on fileb, which depends on filec, and filec depends on filea. This is illegal because a file cannot depend on itself, indirectly or directly.

Macro substitute text 'string' is too long MAKE message

The macro substitution text string overflowed MAKE's internal buffer of 512 bytes.

Macro replace text 'string' is too long MAKE message

The macro replacement text string overflowed MAKE's internal buffer of 512 bytes.

'macroname' - ')' missing in macro invocation MAKE message

The macro you entered is missing a right parenthesis.

Cycle in include files: 'filename' MAKE message

This error message is issued if a makefile includes itself in the make script.

FATAL ERROR: GP FAULT MAKE message

Your program caused a general protection fault and exited fatally. This type of error is caused by various reasons such as attempting to access or write to out of bound memory. For best results, use CodeGuard to locate the error.

Cannot write a string option MAKE message

The –W MAKE option writes a character option to MAKE.EXE. If there's any string option, this error message is generated. For example, the following string option generates this message:

```
-Dxxxx="My_foo" or -Uxxxxx
```

Cannot find MAKE.EXE MAKE message

The MAKE command-line tool cannot be found. Be sure that MAKE.EXE is in either the current directory or in a directory contained in your directory path.

Unable to open file 'filename' MAKE message

This error occurs if the named file does not exist or is misspelled.

'filename' not a MAKE MAKE message

The file you specified with the –f option is not a makefile.

Write error on file 'filename' MAKE message

MAKE couldn't open or write to the file specified in the makefile. Check to ensure that there's enough space left on your disk, and that you have write access to the disk.

Command arguments too long MAKE message

The arguments to a command exceeded the 511-character limit imposed by DOS.

Unexpected end of file in conditional started at line 'line number' MAKE message

The source file ended before MAKE encountered an !endif. The !endif was either missing or misspelled.

Unknown preprocessor statement MAKE message

A ! character was encountered at the beginning of a line, and the statement name following it was not error, undef, if, elif, include, else, or endif.

Bad filename format in include statement MAKE message

Include file names must be surrounded by quotes or angle brackets. The file name was missing the opening quote or angle bracket.

Filename too long MAKE message

The path name in an !include directive overflowed MAKE's internal buffer (512 bytes).

No filename ending MAKE message

The file name in an !include statement is missing the correct closing quote or angle bracket.

Unable to open include file 'filename' MAKE message

The compiler could not find the named file. This error can also be caused if an !include file included itself, or if you do not have FILES set in CONFIG.SYS on your root directory (try `FILES=20`). Check whether the named file exists.

Macro expansion too long

MAKE message

A macro cannot expand to more than 4,096 characters. This error often occurs if a macro recursively expands itself. A macro cannot legally expand to itself.

If statement too long MAKE message

An If statement has exceeded 4,096 characters.

Rule line too long MAKE message

An implicit or explicit rule was longer than 4,096 characters.

Bad undef statement syntax MAKE message

An !undef statement must contain a single identifier and nothing else as the body of the statement.

Misplaced else statement MAKE message

An !else directive is missing a matching !if directive.

Command syntax error MAKE message

This message occurs if

- The first rule line of the makefile contains any leading whitespace.
- An implicit rule does not consist of .ext.ext:.
- An explicit rule does not contain a name before the : character.
- A macro definition does not contain a name before the = character.

Misplaced elif statement MAKE message

An !elif directive is missing a matching !if directive.

Misplaced endif statement MAKE message

An !endif directive is missing a matching !if directive.

Illegal character in constant expression 'expression'

MAKE message

MAKE encountered a character not allowed in a constant expression. If the character is a letter, this probably indicates a misspelled identifier.

Expression syntax error in !if statement MAKE message

The expression in an !if statement is badly formed—it contains a mismatched parenthesis, an extra or missing operator, or a missing or extra constant.

Illegal octal digit MAKE message

An octal constant containing a digit of 8 or 9 was found.

Character constant too long MAKE message

A char constant in an expression is too long.

Division by zero MAKE message

A division or remainder operator in an !if statement has a zero divisor.

Redefinition of target 'filename' MAKE message

The named file occurs on the left side of more than one explicit rule.

Bad macro output translator MAKE message

Invalid syntax for substitution within macros.

Too many suffixes in .SUFFIXES list MAKE message

The suffixes list can include up to 255 suffixes.

Use of : and :: dependents for target 'target'

MAKE message

You tried to use the target in both single and multiple description blocks (using both the : and :: operators). Examples:

```
filea: fileb
filea:: filec
```

Ifdef statement too long MAKE message

An Ifdef statement has exceeded 4,096 characters.

Ifndef statement too long MAKE message

An Ifndef statement has exceeded 4,096 characters.

No match found for wildcard 'expression' MAKE message

No files match the wildcard expression that you want MAKE to expand. For example, the following causes MAKE to send this error message if there are no files with the extension .OBJ in the current directory:

```
prog.exe: *.obj
```

No closing quote MAKE message

A string expression is missing a closing quote in an !if or !elif expression.

String type not allowed with this operand MAKE message

You tried to use an operand that is not allowed for comparing string types. Valid operands are ==, !=, <, >, <=, and >=.

Int and string types compared MAKE message

You tried to compare an integer operand with a string operand in an !if or !elif expression.

Colon expected MAKE message

Your implicit rule is missing a colon at the end.

```
.c.obj:      # Correct
.c.obj       # Incorrect
```

Only <<KEEP or <<NOKEEP MAKE message

You specified something besides KEEP or NOKEEP when closing a temporary inline file.

Unexpected end of file MAKE message

The end of the makefile was reached before closing a temporary inline file.

Cannot have path list for target

MAKE message

You can only specify a path list for dependents of an explicit rule. For example, an invalid and a valid path list are shown here:

```
{path1;path2}prog.exe: prog.obj      # Invalid
prog.exe: {path1;path2}prog.obj      # Valid
```

Cannot have multiple paths for implicit rule MAKE message

You can have only one path for each of the extensions in an implicit rule; for example, `{path}.c.obj`. Multiple path lists are allowed only for dependents in an explicit rule.

No macro before = MAKE message

You must name a macro before you can assign it a value.

Too many rules for target 'target' MAKE message

MAKE can't determine which rules to follow when building a target because you've created too many rules for the target. For example, the following makefile generates this error message:

```
abc.exe : a.obj
  bcc -c a.c

abc.exe : b.obj

abc.exe : c.obj
  bcc -c b.c c.c
```

Illegal/invalid option in CMDSWITCHES directive 'option' MAKE message

The !CMDSWITCHES preprocessing directive turns on or off one or more command-line options. Specify an operator, either a plus sign (+) to turn options on, or a minus sign (-) to turn options off, followed by one or more letters specifying options. An invalid or illegal option is specified in the ! CMDSWITCHES directive.

Unknown CMDSWITCHES operator 'operator' MAKE message

The !CMDSWITCHES directive is supposed to turn on command-line options using a plus sign (+) followed by the option and turn them off using a minus sign (-) followed by the option. An operator other than + or - is specified in the !CMDSWITCHES directive.

MAKE errors

Following are the error messages that can occur while using MAKE to compile a program:

Bad filename format in include statement
Bad macro output translator
Bad undef statement syntax
Cannot find MAKE.EXE
Cannot have multiple paths for implicit rule
Cannot have path list for target
Cannot write a string option
Character constant too long
Circular dependency exists in makefile
Colon expected
Command arguments too long
Command syntax error
Cycle in include files: 'filename'
Division by zero
Error directive: 'message'
Expression syntax error in !if statement
FATAL ERROR: GP FAULT
filename' does not existdon't know how to make it
'filename' not a MAKE
Filename too long
If statement too long
Ifdef statement too long
Ifndef statement too long
Illegal character in constant expression 'expression'
Illegal octal digit
Illegal/invalid option in CMDSWITCHES directive 'option'
Incorrect command line argument:
Int and string types compared
Macro expansion too long
Macro replace text 'string' is too long
Macro substitute text 'string' is too long
'macroname'—')' missing in macro invocation
Misplaced elif statement
Misplaced else statement
Misplaced endif statement
No closing quote
No filename ending
No macro before =
No match found for wildcard 'expression'
No terminator specified for in-line file operator
Not enough memory
Only <<KEEP or <<NOKEEP
Redefinition of target 'filename'
Rule line too long
String type not allowed with this operand
Too many rules for target 'target'
Too many suffixes in .SUFFIXES list

Unable to execute command: 'command'
Unable to open file 'filename'
Unable to open include file 'filename'
Unable to open makefile
Unable to redirect input or output
Unexpected end of file in conditional started at line 'line number'
Unexpected end of file
Unknown CMDSWITCHES operator 'operator'
Unknown preprocessor statement
Use of : and :: dependents for target 'target'
Write error on file 'filename'

User Break

You typed a Ctrl+Break while compiling a program. (This is not an error, just a confirmation.)

Unknown error (# errornum)

Internal compiler error. Contact Inprise Technical Support.

Link terminated by user

You canceled the link. (This is not an error, just a confirmation.)

Fatal Error: Cannot Load Linker: linker

You are trying to load a version of the linker which is not compatible with this version of C++Builder. Try reinstalling C++Builder or contact Inprise Technical Support.

Fatal Error: Linker missing CREATE Entry Point

You are trying to load a version of the linker which is not compatible with this version of C++Builder. Try reinstalling C++Builder or contact Inprise Technical Support.

Fatal Error: Linker missing DESTROY Entry Point

You are trying to load a version of the linker which is not compatible with this version of C++Builder. Try reinstalling C++Builder or contact Inprise Technical Support.

Fatal Error: Linker CREATE Failed

You are trying to load a version of the linker which is not compatible with this version of C++Builder. Try reinstalling C++Builder or contact Inprise Technical Support.

Cannot find tasm program: tasm32.exe

You are trying to build code that includes assembly language directives but C++Builder cannot locate the assembler on your system. Borland's Turbo Assembler (tasm.32) is not provided with C++Builder. If you have the Turbo Assembler, make sure it is in your path.

The project's target could not be found in file 'file'.
The project's target is commonly WinMain or DllEntryPoint

You are trying to build a project that is missing the WinMain or DllEntryPoint. C++ Builder projects generally consist of a makefile (.BPR or .BPK) and a .CPP project source file. For Windows GUI applications, the project source file includes a WinMain as the Entry point. For a DLL, the entry point is DllEntryPoint and for a console application the entry point is main.

Unknown fatal error

Internal resource compiler error. C++Builder loaded the compiler, but it was unable to run. The compilation process was abruptly terminated. Call Inprise Technical Support.

Unable to load RW32CORE.DLL

Internal resource compiler error. C++Builder was unable to load the resource compiler DLL (RW32CORE.DLL) because it is not there, there is not enough memory, or the DLL is damaged in some way. Contact Inprise Technical Support.

Cannot locate file I/O hook functions for resource compiler

C++Builder loaded a version of RW32CORE.DLL, but for some reason, it is not the correct version. This could happen if you inadvertently loaded the Borland C++ 5.0 or 5.01 version of RW32CORE.DLL instead of the one provided with C++Builder. You may need to reinstall C++Builder.

Deprecated features and elements in C++Builder

The following program features and language elements are either no longer supported or no longer used in this release of C++Builder.

RTL elements

- **farfree**
- **faralloc**
- **farmalloc**
- **farheapcheck**
- **farrealloc**
- **inp**
- **inpw**
- **outp**
- **outpw**
- **inport**
- **inportb**
- **outport**
- **outportb**

farfree_ex

This feature or language element is no longer used or supported in this product.

faralloc_ex

This feature or language element is no longer used or supported in this product.

farmalloc_ex

This feature or language element is no longer used or supported in this product.

farheapcheck_ex

This feature or language element is no longer used or supported in this product.

farrealloc_ex

This feature or language element is no longer used or supported in this product.

inp_ex

This feature or language element is no longer used or supported in this product.

inpw_ex

This feature or language element is no longer used or supported in this product.

outp_ex

This feature or language element is no longer used or supported in this product.

outpw_ex

This feature or language element is no longer used or supported in this product.

inport_ex

This feature or language element is no longer used or supported in this product.

inportb_ex

This feature or language element is no longer used or supported in this product.

outport_ex

This feature or language element is no longer used or supported in this product.

outportb_ex

This feature or language element is no longer used or supported in this product.

Object Pascal compiler error messages

Complete list of compiler error messages

The most convenient way to get information on a message you receive in the Integrated Development Environment (IDE) is to highlight the message in the message window and press F1.

0. Ordinal type required

Complete list of compiler error messages

The compiler required an ordinal type at this point. Ordinal types are the predefined types Integer, Char, WideChar, Boolean, and declared enumerated types.

Ordinal types are required in several different situations:

- The index type of an array must be ordinal.
- The low and high bounds of a subrange type must be constant expressions of ordinal type.
- The element type of a set must be an ordinal type.
- The selection expression of a case statement must be of ordinal type.
- The first argument to the standard procedures Inc and Dec must be a variable of either ordinal or pointer type.

```
program Produce;
type
  TByteSet = set of 0..7;
var
  BitCount: array [TByteSet] of Integer;
begin
end.
```

The index type of an array must be an ordinal type - type TByteSet is a set, not an ordinal.

```
program Solve;
type
  TByteSet = set of 0..7;
var
  BitCount: array [Byte] of Integer;
begin
end.
```

Supply an ordinal type as the array index type.

1. File type not allowed here

Complete list of compiler error messages

File types are not allowed as value parameters and as the base type of a file type itself. They are also not allowed as function return types, and you cannot assign them - those errors will however produce a different error message.

```
program Produce;

procedure WriteInteger(T: Text; I: Integer);
begin
  Writeln(T, I);
end;

begin
end.
```

In this example, the problem is that T is value parameter of type Text, which is a file type. Recall that whatever gets written to a value parameter has no effect on the caller's copy of the variable - declaring a file as a value parameter therefore makes little sense.

```
program Solve;

procedure WriteInteger(var T: Text; I: Integer);
begin
  Writeln(T, I);
end;

begin
end.
```

Declaring the parameter as a var parameter solves the problem.

2. Undeclared identifier: '<name>'

Complete list of compiler error messages

The compiler could not find the given identifier - most likely it has been misspelled either at the point of declaration or the point of use. It might be from another unit that has not mentioned a uses clause.

```
program Produce;
var
  Counter: Integer;
begin
  Count := 0;
  Inc(Count);
  Writeln(Count);
end.
```

In the example the variable has been declared as "Counter", but used as "Count". The solution is to either change the declaration or the places where the variable is used.

```
program Solve;
var
  Count: Integer;
begin
  Count := 0;
  Inc(Count);
  Writeln(Count);
end.
```

In the example we have chosen to change the declaration - that was less work.

3. Identifier redeclared: '<name>'

Complete list of compiler error messages

The given identifier has already been declared in this scope - you are trying to reuse its name for something else.

```
program Tests;
var
  Tests: Integer;
begin
end.
```

Here the name of the program is the same as that of the variable - we need to change one of them to make the compiler happy.

```
program Tests;
var
  TestCnt: Integer;
begin
end.
```

4. '<name>' is not a type identifier

Complete list of compiler error messages

This error message occurs when the compiler expected the name of a type, but the name it found did not stand for a type.

```
program Produce;
type
  TMyClass = class
    Field: Integer;
  end;
var
  MyClass: TMyClass;

procedure Proc(C: MyClass);           (*<-- Error message here*)
begin
end;

begin
end.
```

The example erroneously uses the name of the variable, not the name of the type, as the type of the argument.

```
program Solve;
type
  TMyClass = class
    Field: Integer;
  end;
var
  MyClass: TMyClass;

procedure Proc(C: TMyClass);
begin
end;

begin
end.
```

Make sure the offending identifier is indeed a type - maybe it was misspelled, or another identifier of the same name hides the one you meant to refer to.

5. PACKED not allowed here

Complete list of compiler error messages

The packed keyword is only legal for set, array, record, object, class and file types. In contrast to the 16-bit version of Delphi, packed will affect the layout of record, object and class types.

```
program Produce;
type
  SmallReal = packed Real;
begin
end.
```

Packed can not be applied to a real type - if you want to conserve storage, you need to use the smallest real type, type Single.

```
program Solve;
type
  SmallReal = Single;
begin
end.
```

6. Constant or type identifier expected

Complete list of compiler error messages

This error message occurs when the compiler expects a type, but finds a symbol that is neither a constant (a constant could start a subrange type), nor a type identifier.

```
program Produce;
var
  c : ExceptionClass; (*ExceptionClass is a variable in System*)
begin
end.
```

Here, ExceptionClass is a variable, not a type.

```
program Solve;
program Produce;
var
  c : Exception; (*Exception is a type in SysUtils*)
begin
end.
```

You need to make sure you specify a type. Maybe the identifier is misspelled, or it is hidden by some other identifier, for example from another unit.

7. Incompatible types

Complete list of compiler error messages

This error message occurs when the compiler expected two types to be compatible (meaning very similar), but in fact, they turned out to be different. This error occurs in many different situations - for example when a read or write clause in a property mentions a method whose parameter list does not match the property, or when a parameter to a standard procedure or function is of the wrong type.

This error can also occur when two units both declare a type of the same name. When a procedure from an imported unit has a parameter of the same-named type, and a variable of the same-named type is passed to that procedure, the error could occur.

```
unit unit1;
interface
  type
    ExportedType = (alpha, beta, gamma);

implementation
begin
end.

unit unit2;
interface
  type
    ExportedType = (alpha, beta, gamma);

  procedure ExportedProcedure(v : ExportedType);

implementation
  procedure ExportedProcedure(v : ExportedType);
  begin
  end;

begin
end.

program Produce;
uses unit1, unit2;

var
  A: array [0..9] of char;
  I: Integer;
  V : ExportedType;
begin
  ExportedProcedure(v);
  I:= Hi(A);
end.
```

The standard function Hi expects an argument of type Integer or Word, but we supplied an array instead. In the call to ExportedProcedure, V actually is of type unit1.ExportedType since unit1 is imported prior to unit2, so an error will occur.

```
unit unit1;
interface
  type
    ExportedType = (alpha, beta, gamma);

implementation
begin
end.

unit unit2;
interface
  type
    ExportedType = (alpha, beta, gamma);

  procedure ExportedProcedure(v : ExportedType);

implementation
  procedure ExportedProcedure(v : ExportedType);
  begin
  end;

begin
end.

program Solve;
uses unit1, unit2;
var
  A: array [0..9] of char;
  I: Integer;
  V : unit2.ExportedType;
begin
  ExportedProcedure(v);
  I:= High(A);
end.
```

We really meant to use the standard function High, not Hi. For the ExportedProcedure call, there are two alternative solutions. First, you could alter the order of the uses clause, but it could also cause similar errors to occur. A more robust solution is to fully qualify the type name with the unit which declared the desired type, as has been done with the declaration for V above.

8. Incompatible types: <text>

Complete list of compiler error messages

The compiler has detected a difference between the declaration and use of a procedure.

```
program Produce;

  type
    ProcedureParm0 = procedure; stdcall;
    ProcedureParm1 = procedure(VAR x : Integer);

  procedure WrongConvention; register;
  begin
  end;

  procedure WrongParms(x, y, z : Integer);
  begin
  end;

  procedure TakesParm0(p : ProcedureParm0);
  begin
  end;

  procedure TakesParm1(p : ProcedureParm1);
  begin
  end;

begin
  TakesParm0(WrongConvention);
  TakesParm1(WrongParms);
end.
```

The call of 'TakesParm0' will elicit an error because the type 'ProcedureParm0' expects a 'stdcall' procedure, whereas 'WrongConvention' is declared with the 'register' calling convention. Similarly, the call of 'TakesParm1' will fail because the parameter lists do not match.

```
program Solve;

  type
    ProcedureParm0 = procedure; stdcall;
    ProcedureParm1 = procedure(VAR x : Integer);

  procedure RightConvention; stdcall;
  begin
  end;

  procedure RightParms(VAR x : Integer);
  begin
  end;

  procedure TakesParm0(p : ProcedureParm0);
  begin
  end;

  procedure TakesParm1(p : ProcedureParm1);
  begin
  end;

begin
  TakesParm0(RightConvention);
  TakesParm1(RightParms);
end.
```

The solution to both of these problems is to ensure that the calling convention or the parameter lists matches the declaration.

9. Incompatible types: '<name>' and '<name>'

Complete list of compiler error messages

This error message results when the compiler expected two types to be compatible (i.e. similar), but they turned out to be different.

```
program Produce;

procedure Proc(I: Integer);
begin
end;

begin
  Proc( 22 / 7 ); (*Result of / operator is Real*)
end.
```

Here a C++ programmer thought the division operator / would give him an integral result - not the case in Pascal.

```
program Solve;

procedure Proc(I: Integer);
begin
end;

begin
  Proc( 22 div 7 ); (*The div operator gives result type
Integer*)
end.
```

The solution in this case is to use the integral division operator div - in general, you have to look at your program very careful to decide how to resolve type incompatibilities.

10. Low bound exceeds high bound

Complete list of compiler error messages

This error message is given when either the low bound of a subrange type is greater than the high bound, or the low bound of a case label range is greater than the high bound.

```
program Produce;
type
  SubrangeType = 1..0;              (*Gets: Low bound exceeds
high bound *)
begin
  case True of
  True..False:                      (*Gets: Low bound exceeds
high bound *)
    Writeln('Expected result');
  else
    Writeln('Unexpected result');
  end;
end.
```

In the example above, the compiler gives an error rather than treating the ranges as empty. Most likely, the reversal of the bounds was not intentional.

```
program Solve;
type
  SubrangeType = 0..1;
begin
  case True of
  False..True:
    Writeln('Expected result');
  else
    Writeln('Unexpected result');
  end;
end.
```

Make sure you have specified the bounds in the correct order.

11. Type of expression must be BOOLEAN

Complete list of compiler error messages

This error message is output when an expression serves as a condition and must therefore be of Boolean type. This is the case for the controlling expression of the if, while and repeat statements, and for the expression that controls a conditional breakpoint.

```
program Produce;
var
  P: Pointer;
begin
  if P then
    Writeln('P <> nil');
end.
```

Here, a C++ programmer just used a pointer variable as the condition of an if statement.

```
program Solve;
var
  P: Pointer;
begin
  if P <> nil then
    Writeln('P <> nil');
end.
```

In Pascal, you need to be more explicit in this case.

12. Type of expression must be INTEGER

Complete list of compiler error messages

This error message is only given when the constant expression that specifies the number of characters in a string type is not of type integer.

```
program Produce;
type
  color = (red,green,blue);
var
  S3 : string[Succ(High(color))];
begin
end.
```

The example tries to specify the number of elements in a string as dependent on the maximum element of type color - unfortunately, the element count is of type color, which is illegal.

```
program Solve;
type
  color = (red,green,blue);
var
  S3 : string[ord(High(color))+1];
begin
end.
```

13. Statement expected, but expression of type '<type>' found

Complete list of compiler error messages

The compiler was expecting to find a statement, but instead it found an expression of the specified type.

```
program Produce;
  var
    a : Integer;
begin
   (3 + 4);
end.
```

In this example, the compiler is expecting to find a statement, such as an IF, WHILE, REPEAT, but instead it found the expression (3+4).

```
program Produce;
  var
    a : Integer;
begin
   a := (3 + 4);
end.
```

The solution here was to assign the result of the expression (3+4) to the variable 'a'. Another solution would have been to remove the offending expression from the source code - the choice depends on the situation.

14. Operator not applicable to this operand type

Complete list of compiler error messages

This error message is given whenever an operator cannot be applied to the operands it was given - for instance if a boolean operator is applied to a pointer.

```
program Produce;
var
  P: ^Integer;
begin
  if P and P^ > 0 then
    Writeln('P points to a number greater 0');
end.
```

Here a C++ programmer was unclear about operator precedence in Pascal - P is not a boolean expression, and the comparison needs to be parenthesized.

```
program Solve;
var
  P: ^Integer;
begin
  if (P <> nil) and (P^ > 0) then
    Writeln('P points to a number greater 0');
end.
```

If we explicitly compare P to nil and use parentheses, the compiler is happy.

15. Array type required

Complete list of compiler error messages

This error message is given if you either index into an operand that is not an array, or if you pass an argument that is not an array to an open array parameter.

```
program Produce;
var
  P: ^Integer;
  I: Integer;
begin
  Writeln(P[I]);
end.
```

We try to apply an index to a pointer to integer - that would be legal in C, but is not in Pascal.

```
program Solve;
type
  TIntArray = array [0..MaxInt DIV sizeof(Integer)-1] of Integer;
var
  P: ^TIntArray;
  I: Integer;
begin
  Writeln(P^[I]);   (*Actually, P[I] would also be legal in
Delphi32*)
end.
```

In Pascal, we must tell the compiler that we intend P to point to an array of integers.

16. Pointer type required

Complete list of compiler error messages

This error message is given when you apply the dereferencing operator '^' to an operand that is not a pointer, and, as a very special case, when the second operand in a 'Raise <exception> at <address>' statement is not a pointer.

```
program Produce;
var
  C: TObject;
begin
  C^.Destroy;
end.
```

Even though class types are implemented internally as pointers to the actual information, it is illegal to apply the dereferencing operator to class types at the source level. It is also not necessary - the compiler will dereference automatically whenever it is appropriate.

```
program Solve;
var
  C: TObject;
begin
  C.Destroy;
end.
```

Simply leave off the dereferencing operator—the compiler will do the right thing automatically.

17. Record, object or class type required

Complete list of compiler error messages

The compiler was expecting to find the type name which specified a record, object or class but did not find one.

```
program Produce;

  type
    RecordDesc = class
      ch : Char;
    end;

  var
    pCh : PChar;
    r : RecordDesc;

  procedure A;
  begin
    pCh.ch := 'A';    (* case 1 *)

    with pCh do begin (* case 2 *)
    end;
  end;
end.
```

There are two causes for the same error in this program. The first is the application of '.' to a object that is not a record. The second case is the use of a variable which is of the wrong type in a WITH statement.

```
program Solve;

  type
    RecordDesc = class
      ch : Char;
    end;

  var
    r : RecordDesc;

  procedure A;
  begin
    r.ch := 'A';    (* case 1 *)

    with r do begin (* case 2 *)
    end;
  end;
end.
```

The easy solution to this error is to always make sure that the '.' and WITH are both applied only to records, objects or class variables.

18. Object type required

Complete list of compiler error messages

This error is given whenever an object type is expected by the compiler. For instance, the ancestor type of an object must also be an object type.

```
type
  MyObject = object(TObject)
  end;
begin
end.
```

Confusingly enough, TObject in the unit System has a class type, so we cannot derive an object type from it.

```
program Solve;
type
  MyObject = class  (*Actually, this means: class(TObject)*)
  end;
begin
end.
```

Make sure the type identifier really stands for an object type - maybe it is misspelled, or maybe is hidden by an identifier from another unit.

19. Object or class type required

Complete list of compiler error messages

This error message is given when the syntax 'Typename.Methodname' is used, but the typename does not refer to an object or class type.

```
program Produce;
type
  TInteger = class
    Value: Integer;
  end;
var
  V: TInteger;
begin
  V := Integer.Create;
end.
```

Type Integer does not have a Create method, but TInteger does.

```
program Solve;
type
  TInteger = class
    Value: Integer;
  end;
var
  V: TInteger;
begin
  V := TInteger.Create;
end.
```

Make sure the identifier really refers to an object or class type - maybe it is misspelled or it is hidden by an identifier from another unit.

20. Class type required

Complete list of compiler error messages

In certain situations the compiler requires a class type:

- As the ancestor of a class type
- In the on-clause of a try-except statement
- As the first argument of a raise statement
- As the final type of a forward declared class type

```
program Produce;
begin
  raise 'This would work in C++, but does not in Delphi';
end.
```

```
program Solve;
uses SysUtils;
begin
  raise Exception.Create('There is a simple workaround,
however');
end.
```

21. Function needs result type

Complete list of compiler error messages

You have declared a function, but have not specified a return type.

```
program Produce;

function Sum(A: array of Integer);
var I: Integer;
begin
  Result := 0;
  for I := 0 to High(A) do
    Result := Result + A[I];
end;

begin
end.
```

Here Sum is meant to be function, we have not told the compiler about it.

```
program Solve;

function Sum(A: array of Integer): Integer;
var I: Integer;
begin
  Result := 0;
  for I := 0 to High(A) do
    Result := Result + A[I];
end;

begin
end.
```

Just make sure you specify the result type.

22. Invalid function result type

Complete list of compiler error messages

File types are not allowed as function result types.

```
program Produce;

function OpenFile(Name: string): File;
begin
end;

begin
end.
```

You cannot return a file from a function.

```
program Solve;

procedure OpenFile(Name: string; var F: File);
begin
end;

begin
end.
```

You can 'return' the file as a variable parameter. Alternatively, you can also allocate a file dynamically and return a pointer to it.

23. Procedure cannot have a result type

Complete list of compiler error messages

You have declared a procedure, but given it a result type. Either you really meant to declare a function, or you should delete the result type.

```
program Produce;

procedure DotProduct(const A,B: array of Double): Double;
var
  I: Integer;
begin
  Result := 0.0;
  for I := 0 to High(A) do
    Result := Result + A[I]*B[I];
end;

const
  C: array [1..3] of Double = (1,2,3);

begin
  Writeln( DotProduct(C,C) );
end.
```

Here DotProduct was really meant to be a function, we just happened to use the wrong keyword...

```
program Solve;

function DotProduct(const A,B: array of Double): Double;
var
  I: Integer;
begin
  Result := 0.0;
  for I := 0 to High(A) do
    Result := Result + A[I]*B[I];
end;

const
  C: array [1..3] of Double = (1,2,3);

begin
  Writeln( DotProduct(C,C) );
end.
```

Just make sure you specify a result type when you declare a function, and no result type when you declare a procedure.

24. Text after final 'END.' - ignored by compiler

Complete list of compiler error messages

This warning is given when there is still source text after the final end and the period that constitute the logical end of the program. Possibly the nesting of begin-end is inconsistent (there is one `end` too many somewhere). Check whether you intended the source text to be ignored by the compiler - maybe it is actually quite important.

```
program Produce;

begin
end.

Text here is ignored by Delphi 16-bit - Delphi 32-bit gives a
warning.


program Solve;

begin
end.
```

25. Constant expression expected

Complete list of compiler error messages

The compiler expected a constant expression here, but the expression it found turned out not to be constant.

```
program Produce;
const
  Message = 'Hello World!';
  WPosition = Pos('W', Message);
begin
end.
```

The call to Pos is not a constant expression to the compiler, even though its arguments are constants, and it could in principle be evaluated at compile time.

```
program Solve;
const
  Message = 'Hello World!';
  WPosition = 7;
begin
end.
```

So in this case, we just have to calculate the right value for WPosition ourselves.

26. Constant expression violates subrange bounds

Complete list of compiler error messages

This error message occurs when the compiler can determine that a constant is outside the legal range. This can occur for instance if you assign a constant to a variable of subrange type.

```
program Produce;
var
  Digit: 1..9;
begin
  Digit := 0;  (*Get message: Constant expression violates
subrange bounds*)
end.
```

```
program Solve;
var
  Digit: 0..9;
begin
  Digit := 0;
end.
```

27. Duplicate tag value

Complete list of compiler error messages

This error message is given when a constant appears more than once in the declaration of a variant record.

```
program Produce;
type
  VariantRecord = record
    case Integer of
    0: (IntField: Integer);
    0: (RealField: Real);     (*<-- Error message here*)
  end;

begin
end.
```

```
program Solve;
type
  VariantRecord = record
    case Integer of
    0: (IntField: Integer);
    1: (RealField: Real);
  end;

begin
end.
```

28. Sets may have at most 256 elements

Complete list of compiler error messages

This error message appears when you try to declare a set type of more than 256 elements. More precisely, the ordinal values of the upper and lower bounds of the base type must be within the range 0..255.

```
program Produce;
type
  BigSet = set of 1..256;  (*<-- error message given here*)
begin
end.
```

In the example, BigSet really only has 256 elements, but is still illegal.

```
program Solve;
type
  BigSet = set of 0..255;
begin
end.
```

We need to make sure the upper and lower bounds and in the range 0..255.

29. <Token1> expected but <token2> found

Complete list of compiler error messages

This error message appears for syntax errors. There is probably a typo in the source, or something was left out. When the error occurs at the beginning of a line, the actual error is often on the previous line.

```
program Produce;
var
  I: Integer
begin               (*<-- Error message here: ';' expected but
'BEGIN' found*)
end.
```

After the type Integer, the compiler expects to find a semicolon to terminate the variable declaration. It does not find the semicolon on the current line, so it reads on and finds the 'begin' keyword at the start of the next line. At this point it finally knows something is wrong...

```
program Solve;
var
  I: Integer;       (*Semicolon was missing*)
begin
end.
```

In this case, just the semicolon was missing - a frequent case in practice. In general, have a close look at the line where the error message appears, and the line above it to find out whether something is missing or misspelled.

30. Duplicate case label

Complete list of compiler error messages

This error message occurs when there is more than one case label with a given value in a case statement.

```
program Produce;

function DigitCount(I: Integer): Integer;
begin
  case Abs(I) of
  0:                    DigitCount := 1;
  0        ..9:         DigitCount := 1;   (*<-- Error message
here*)
  10       ..99:        DigitCount := 2;
  100      ..999:       DigitCount := 3;
  1000     ..9999:      DigitCount := 4;
  10000    ..99999:     DigitCount := 5;
  100000   ..999999:    DigitCount := 6;
  1000000  ..9999999:   DigitCount := 7;
  10000000 ..99999999:  DigitCount := 8;
  100000000..999999999: DigitCount := 9;
  else                  DigitCount := 10;
  end;
end;

begin
  Writeln( DigitCount(12345) );
end.
```

Here we did not pay attention and mentioned the case label 0 twice.

```
program Solve;

function DigitCount(I: Integer): Integer;
begin
  case Abs(I) of
  0        ..9:         DigitCount := 1;
  10       ..99:        DigitCount := 2;
  100      ..999:       DigitCount := 3;
  1000     ..9999:      DigitCount := 4;
  10000    ..99999:     DigitCount := 5;
  100000   ..999999:    DigitCount := 6;
  1000000  ..9999999:   DigitCount := 7;
  10000000 ..99999999:  DigitCount := 8;
  100000000..999999999: DigitCount := 9;
  else                  DigitCount := 10;
  end;
end;

begin
  Writeln( DigitCount(12345) );
end.
```

In general, the problem might not be so easy to spot when you have symbolic constants and ranges of case labels - you might have to write down the real values of the constants to find out what is wrong.

31. Label expected

Complete list of compiler error messages

This error message occurs if the identifier given in a goto statement or used as a label in inline assembly is not declared as a label.

```
program Produce;

begin
  if 2*2 <> 4 then
    goto Exit; (*<-- Error message here: Exit is also a standard
procedure*)
  (*...*)
Exit:               (*Additional error messages here*)
end.
```

```
program Solve;
label
  Exit;             (*Labels must be declared in Pascal*)
begin
  if 2*2 <> 4 then
    goto Exit;
  (*...*)
Exit:
end.
```

32. For loop control variable must be simple local variable

Complete list of compiler error messages

This error message is given when the control variable of a for statement is not a simple variable (but a component of a record, for instance), or if it is not local to the procedure containing the for statement.

For backward compatibility reasons, it is legal to use a global variable as the control variable - the compiler gives a warning in this case. Note that using a local variable will also generate more efficient code.

```
program Produce;

var
  I: Integer;
  A: array [0..9] of Integer;

procedure Init;
begin
  for I := Low(A) to High(a) do  (*<-- Warning given here*)
    A[I] := 0;
end;

begin
  Init;
end.
```

```
program Solve;
var
  A: array [0..9] of Integer;

procedure Init;
var
  I: Integer;
begin
  for I := Low(A) to High(a) do
    A[I] := 0;
end;

begin
  Init;
end.
```

33. For loop control variable must have ordinal type

Complete list of compiler error messages

The control variable of a for loop must have type Boolean, Char, WideChar, Integer, an enumerated type, or a subrange type.

```
program Produce;
var
  x: Real;
begin (*Plot sine wave*)
  for x := 0 to 2*pi/0.2 do                              (*<--
Error message here*)
    Writeln( '*': Round((Sin(x*0.2) + 1)*20) + 1 );
end.
```

The example uses a variable of type Real as the for loop control variable, which is illegal.

```
program Solve;
var
  x: Integer;
begin (*Plot sine wave*)
  for x := 0 to Round(2*pi/0.2) do
    Writeln( '*': Round((Sin(x*0.2) + 1)*20) + 1 );
end.
```

Instead, use the Integer ordinal type.

You may see this error if a FOR loop uses an Int64 or Variant control variable. This results from a limitation in the compiler which you can work around by replacing the FOR loop with a WHILE loop.

34. Types of actual and formal var parameters must be identical

Complete list of compiler error messages

For a variable parameter, the actual argument must be of the exact type of the formal parameter.

```
program Produce;

procedure SwapBytes(var B1, B2: Byte);
var
  Temp: Byte;
begin
  Temp := B1; B1 := B2; B2 := Temp;
end;

var
  C1, C2: 0..255;     (*Similar to a byte, but NOT identical*)
begin
  SwapBytes(C1,C2);   (*<-- Error message here*)
end.
```

Arguments C1 and C2 are not acceptable to SwapBytes, although they have the exact memory representation and range that a Byte has.

```
program Solve;

procedure SwapBytes(var B1, B2: Byte);
var
  Temp: Byte;
begin
  Temp := B1; B1 := B2; B2 := Temp;
end;

var
  C1, C2: Byte;
begin
  SwapBytes(C1,C2);   (*<-- Error message here*)
end.
```

So you actually have to declare C1 and C2 as Bytes to make this example compile.

35. Too many actual parameters

Complete list of compiler error messages

This error message occurs when a procedure or function call gives more parameters than the procedure or function declaration specifies.

Additionally, this error message occurs when an OLE automation call has too many (more than 255), or too many named parameters.

```
program Produce;

function Max(A,B: Integer): Integer;
begin
  if A > B then Max := A else Max := B
end;

begin
  Writeln( Max(1,2,3) );    (*<-- Error message here*)
end.
```

It would have been convenient for Max to accept three parameters...

```
program Solve;

function Max(const A: array of Integer): Integer;
var
  I: Integer;
begin
  Result := Low(Integer);
  for I := 0 to High(A) do
    if Result < A[I] then
      Result := A[I];
end;

begin
  Writeln( Max([1,2,3]) );
end.
```

Normally, you would change to call site to supply the right number of parameters. Here, we have chose to show you how to implement Max with an unlimited number of arguments. Note that now you have to call it in a slightly different way.

36. Not enough actual parameters

Complete list of compiler error messages

This error message occurs when a call to procedure or function gives less parameters than specified in the procedure or function declaration.

This can also occur for calls to standard procedures or functions.

```
program Produce;
var
  X: Real;
begin
  Val('3.141592', X);    (*<-- Error message here*)
end.
```

The standard procedure Val has one additional parameter to return an error code in. The example did not supply that parameter.

```
program Solve;
var
  X: Real;
  Code: Integer;
begin
  Val('3.141592', X, Code);
end.
```

Typically, you will check the call against the declaration of the procedure called or the help, and you will find you forgot about a parameter you need to supply.

37. Variable required

Complete list of compiler error messages

This error message occurs when you try to take the address of an expression or a constant.

```
program Produce;
var
  I: Integer;
  PI: ^Integer;
begin
  PI := Addr(1);
end.
```

A constant like 1 does not have a memory address, so you cannot apply the operator or the Addr standard function to it.

```
program Solve;
var
  I: Integer;
  PI: ^Integer;
begin
  PI := Addr(I);
end.
```

You need to make sure you take the address of variable.

38. Declaration of <Name> differs from previous declaration

Complete list of compiler error messages

This error message occurs when the declaration of a procedure, function, method, constructor or destructor differs from its previous (forward) declaration.

This error message also occurs when you try to override a virtual method, but the overriding method has a different parameter list, calling convention etc.

```
program Produce;

type
  MyClass = class
    procedure Proc(Inx: Integer);
    function Func: Integer;
    procedure Load(const Name: string);
    procedure Perform(Flag: Boolean);
    constructor Create;
    destructor Destroy(Msg: string); override;      (*<-- Error
message here*)
    class function NewInstance: MyClass; override;  (*<-- Error
message here*)
  end;

procedure MyClass.Proc(Index: Integer);             (*<-- Error
message here*)
begin
end;

function MyClass.Func: Longint;                     (*<-- Error
message here*)
begin
end;

procedure MyClass.Load(Name: string);               (*<-- Error
message here*)
begin
end;

procedure MyClass.Perform(Flag: Boolean); cdecl;    (*<-- Error
message here*)
begin
end;

procedure MyClass.Create;                           (*<-- Error
message here*)
begin
end;

function MyClass.NewInstance: MyClass;              (*<-- Error
message here*)
begin
end;

begin
end.
```

As you can see, there are a number of reasons for this error message to be issued.

```
program Solve;

type
  MyClass = class
    procedure Proc(Inx: Integer);
    function Func: Integer;
    procedure Load(const Name: string);
    procedure Perform(Flag: Boolean);
    constructor Create;
    destructor Destroy; override;                    (*No
parameters*)
    class function NewInstance: TObject; override;  (*Result type
*)
  end;

procedure MyClass.Proc(Inx: Integer);               (*Parameter
name  *)
begin
end;

function MyClass.Func: Integer;                     (*Result type
*)
begin
end;

procedure MyClass.Load(const Name: string);         (*Parameter
kind  *)
begin
end;

procedure MyClass.Perform(Flag: Boolean);           (*Calling
convention*)
begin
end;

constructor MyClass.Create;
(*constructor*)
begin
end;

class function MyClass.NewInstance: TObject;        (*class
function*)
begin
end;

begin
end.
```

You need to carefully compare the 'previous declaration' with the one that causes the error to determine what is different between the two.

39. Illegal character in input file: '<char>' ($<hex>)

Complete list of compiler error messages

The compiler found a character that is illegal in Pascal programs.

This error message is caused most often by errors with string constants or comments.

```
program Produce;

begin
  Writeln("Hello world!");   (*<-- Error messages here*)
end.
```

Here a programmer fell back to C++ habits and quoted a string with double quotes.

```
program Solve;

begin
  Writeln('Hello world!');   (*Need single quotes in Pascal*)
end.
```

The solution is to use single quotes. In general, you need to delete the illegal character.

40. File not found: <Filename>

Complete list of compiler error messages

This error message occurs when the compiler cannot find an input file. This can be a source file, a compiled unit file (.dcu file), an include, an object file or a resource file.

Check the spelling of the name and the relevant search path.

```
program Produce;
uses SysUtilss;       (*<-- Error message here*)
begin
end.
```

```
program Solve;
uses SysUtils;      (*Fixed typo*)
begin
end.
```

41. Could not create output file <Filename>

Complete list of compiler error messages

The compiler could not create an output file. This can be a compiled unit file (.dcu file), an executable file, a map file or an object file.

Most likely causes are a nonexistent directory or a write protected file or disk.

42. Seek error on <Filename>

Complete list of compiler error messages

The compiler encountered a seek error on an input or output file.

This should never happen - if it does, the most likely cause is corrupt data.

43. Read error on <Filename>

Complete list of compiler error messages

The compiler encountered a read error on an input file.

This should never happen - if it does, the most likely cause is corrupt data.

44. Write error on <Filename>

Complete list of compiler error messages

The compiler encountered a write error while writing to an output file.

Most likely, the output disk is full.

45. Close error on <Filename>

Complete list of compiler error messages

The compiler encountered an error while closing an input or output file.

This should rarely happen. If it does, the most likely cause is a full or bad disk.

46. Bad file format: <Filename>

Complete list of compiler error messages

This error occurs if an object file loaded with a $L or $LINK directive is not of the correct format. Several restrictions must be met:

- Check the naming restrictions on segment names in the help file
- Not more than 10 segments
- Not more than 255 external symbols
- Not more than 50 local names in LNAMES records
- LEDATA and LIDATA records must be in offset order
- No THREAD subrecords are supported in FIXU32 records
- Only 32-bit offsets can be fixed up
- Only segment and self relative fixups
- Target of a fixup must be a segment, a group or an EXTDEF
- Object must be 32-bit object file
- Various internal consistency condition that should only fail if the object file is corrupted.

47. Out of memory

Complete list of compiler error messages

The compiler ran out of memory.

This should rarely happen. If it does, make sure your swap file is large enough and that there is still room on the disk.

48. Circular unit reference to <Unitname>

Complete list of compiler error messages

One or more units use each other in their interface parts.

As the compiler has to translate the interface part of a unit before any other unit can use it, the compiler must be able to find a compilation order for the interface parts of the units.

Check whether all the units in the uses clauses are really necessary, and whether some can be moved to the implementation part of a unit instead.

```
unit A;
interface
uses B;           (*A uses B, and B uses A*)
implementation
end.

unit B;
interface
uses A;
implementation
end.
```

The problem is caused because A and B use each other in their interface sections.

```
unit A;
interface
uses B;          (*Compilation order: B.interface, A,
B.implementation*)
implementation
end.

unit B;
interface
implementation
uses A;          (*Moved to the implementation part*)
end.
```

You can break the cycle by moving one or more uses to the implementation part.

49. Bad unit format: <Filename>

Complete list of compiler error messages

This error occurs if a compiled unit file (.dcu file) has a bad format.

Most likely, the .dcu file has been corrupted. Recompile the file or reinstall Delphi32.

50. Label declaration not allowed in interface part

Complete list of compiler error messages

This error occurs when you declare a label in the interface part of a unit.

```
unit Produce;
interface
label 99;
implementation
begin
99:
end.
```

It is just illegal to declare a label in the interface section of a unit.

```
unit Solve;
interface
implementation
label 99;
begin
99:
end.
```

You have to move it to the implementation section.

51. Statements not allowed in interface part

Complete list of compiler error messages

The interface part of a unit can only contain declarations, not statements.

Move the bodies of procedures to the implementation part.

```
unit Produce;

interface

procedure MyProc;
begin                (*<-- Error message here*)
end;

implementation

begin
end.
```

We got carried away and gave MyProc a body right in the interface section.

```
unit Solve;

interface

procedure MyProc;

implementation

procedure MyProc;
begin
end;

begin
end.
```

We need move the body to the implementation section - then it's fine.

52. Unit <Unit1> was compiled with a different version of <Unit2>

Complete list of compiler error messages

This error occurs when the declaration of symbol declared in the interface part of a unit has changed, and the compiler cannot recompile a unit that relies on this declaration because the source is not available to it.

There are several possible solutions - recompile Unit1 (assuming you have the source code available), use an older version of Unit2 or change Unit2, or get a new version of Unit1 from whoever has the source code to it.

53. Unterminated string

Complete list of compiler error messages

The compiler did not find a closing apostrophe at the end of a character string.

Note that character strings cannot be continued onto the next line - however, you can use the '+' operator to concatenate two character strings on separate lines.

```
program Produce;

begin
  Writeln('Hello world!);   (*<-- Error message here -*)
end.
```

We just forgot the closing quote at the string - no big deal, happens all the time.

```
program Solve;

begin
  Writeln('Hello world!');
end.
```

So we supplied the closing quote, and the compiler is happy.

54. Syntax error in real number

Complete list of compiler error messages

This error message occurs if the compiler finds the beginning of a scale factor (an 'E' or 'e' character) in a number, but no digits follow it.

```
program Produce;
const
  SpeedOfLight = 3.0E 8;     (*<-- Error message here*)
begin
end.
```

In the example, we put a space after '3.0E' - now for the compiler the number ends here, and it is incomplete.

```
program Solve;
const
  SpeedOfLight = 3.0E+8;
begin
end.
```

We could have just deleted the blank, but we put in a '+' sign because it looks a little nicer.

55. Illegal type in Write/Writeln statement

Complete list of compiler error messages

This error occurs when you try to output a type in a Write or Writeln statement that is not legal.

```
program Produce;
type
  TColor = (red,green,blue);
var
  Color : TColor;
begin
  Writeln(Color);
end.
```

It would have been convenient to use a writeln statement to output Color, wouldn't it?

```
program Solve;
type
  TColor = (red,green,blue);
var
  Color : TColor;
const
  ColorString : array [TColor] of string = ('red', 'green',
'blue');
begin
  Writeln(ColorString[Color]);
end.
```

Unfortunately, that is not legal, and we have to do it with an auxiliary table.

56. Illegal type in Read/Readln statement

Complete list of compiler error messages

This error occurs when you try to read a variable in a Read or Readln that is not of a legal type.

Check the type of the variable and make sure you are not missing a dereferencing, indexing or field selection operator.

```
program Produce;
type
  TColor = (red,green,blue);
var
  Color : TColor;
begin
  Readln(Color);     (*<-- Error message here*)
end.
```

We cannot read variables of enumerated types directly.

```
program Solve;
type
  TColor = (red,green,blue);
var
  Color : TColor;
  InputString: string;
const
  ColorString : array [TColor] of string = ('red', 'green',
'blue');
begin
  Readln(InputString);
  Color := red;
  while (color < blue) and (ColorString[color] <> InputString) do
    Inc(color);
end.
```

The solution is to read a string, and look up that string in an auxiliary table. In the example above, we didn't bother to do error checking - any string will be treated as 'blue'. In practice, we would probably output an error message and ask the user to try again.

57. Strings may have at most 255 elements

Complete list of compiler error messages

This error message occurs when you declare a string type with more than 255 elements, if you assign a string literal of more than 255 characters to a variable of type ShortString, or when you have more than 255 characters in a single character string.

Note that you can construct long string literals spanning more than one line by using the '+' operator to concatenate several string literals.

```
program Produce;
var
  LongString : string[256];  (*<-- Error message here*)
begin
end.
```

In the example above, the length of the string is just one beyond the limit.

```
program Solve;
var
  LongString : AnsiString;
begin
end.
```

The most convenient solution is to use the new long strings - then you don't even have to spend any time thinking about what a reasonable maximum length would be.

58. Unexpected end of file in comment started on line <Number>

Complete list of compiler error messages

This error occurs when you open a comment, but do not close it.

Note that a comment started with '{' must be closed with '}', and a comment started with '(*' must be closed with '*)'.

```
program Produce;
(*Let's start a comment here but forget to close it
begin
end.
```

So the example just didn't close the comment.

```
program Solve;
(*Let's start a comment here and not forget to close it*)
begin
end.
```

Doing so fixes the problem.

59. Invalid compiler directive: '<Directive>'

Complete list of compiler error messages

This error message means there is an error in a compiler directive or in a command line option. Here are some possible error situations:

- An external declaration was syntactically incorrect.

- A command line option or an option in a DCC32.CFG file was not recognized by the compiler or was invalid. For example, '-$M100' is invalid because the minimum stack size must be at least 1024.

- The compiler found a $XXXXX directive, but could not recognize it. It was probably misspelled.

- The compiler found a $ELSE or $ENDIF directive, but no preceding $IFDEF, $IFNDEF or $IFOPT directive.

- (*$IFOPT*) was not followed by a switch option and a + or -.

- The long form of a switch directive was not followed by ON or OFF.

- A directive taking a numeric parameter was not followed by a valid number.

- The $DESCRIPTION directive was not followed by a string.

- The $APPTYPE directive was not followed by CONSOLE or GUI.

- The $ENUMSIZE directive (short form $Z) was not followed by 1,2 or 4.

```
(*$Description Copyright Borland International 1996*)     (*<--
Error here*)
program Produce;
(*$AppType Console*)                                       (*<--
Error here*)

begin
(*$If O+*)                                                 (*<--
Error here*)
  Writeln('Optimizations are ON');
(*$Else*)                                                  (*<--
Error here*)
  Writeln('Optimizations are OFF');
(*$Endif*)                                                 (*<--
Error here*)
  Writeln('Hello world!');
end.
```

The example shows three typical error situations, and the last two errors are caused by the compiler not having recognized $If.

```
(*$Description 'Copyright Borland International 1996'*)  (*Need
string*)
program Solve;
(*$AppType Console*)
(*AppType*)

begin
(*$IfOpt O+*)
(*IfOpt*)
  Writeln('Optimizations are ON');
(*$Else*)                                                 (*Now
fine*)
  Writeln('Optimizations are OFF');
(*$Endif*)                                                (*Now
fine*)
  Writeln('Hello world!');
end.
```

So $Description needs a quoted string, we need to spell $AppType right, and checking options is done with $IfOpt. With these changes, the example compiles fine.

60. Bad global symbol definition: '<Name>' in object file '<Filename>'

Complete list of compiler error messages

This warning is given when an object file linked in with a $L or $LINK directive contains a definition for a symbol that was not declared in Pascal as an external procedure, but as something else (e.g. a variable).

The definition in the object will be ignored in this case.

61. Class or object types only allowed in type section

Complete list of compiler error messages

Class or object types must always be declared with an explicit type declaration in a type section - unlike record types, they cannot be anonymous.

The main reason for this is that there would be no way you could declare the methods of that type - after all, there is no type name.

```
program Produce;

var
  MyClass : class
    Field: Integer;
  end;

begin
end.
```

The example tries to declare a class type within a variable declaration - that is not legal.

```
program Solve;

type
  TMyClass = class
    Field: Integer;
  end;

var
  MyClass : TMyClass;

begin
end.
```

The solution is to introduce a type declaration for the class type. Alternatively, you could have changed the class type to a record type.

62. Local class or object types not allowed

Complete list of compiler error messages

Class and object cannot be declared local to a procedure.

```
program Produce;

  procedure MyProc;
  type
    TMyClass = class
      Field: Integer;
    end;
  begin
  (*...*)
  end;

begin
end.
```

So MyProc tries to declare a class type locally, which is illegal.

```
program Solve;

  type
    TMyClass = class
      Field: Integer;
    end;

  procedure MyProc;
  begin
  (*...*)
  end;

begin
end.
```

The solution is to move out the declaration of the class or object type to the global scope.

63. Virtual constructors are not allowed

Complete list of compiler error messages

Unlike class types, object types can only have static constructors.

```
program Produce;

type
  TMyObject = object
    constructor Init; virtual;
  end;

constructor TMyObject.Init;
begin
end;

begin
end.
```

The example tries to declare a virtual constructor, which does not really make sense for object types and is therefore illegal.

```
program Solve;

type
  TMyObject = object
    constructor Init;
  end;

constructor TMyObject.Init;
begin
end;

begin
end.
```

The solution is to either make the constructor static, or to use a new-style class type which can have a virtual constructor.

64. Could not compile used unit '<Unitname>'

Complete list of compiler error messages

This fatal error is given when a unit used by another could not be compiled. In this case, the compiler gives up compilation of the dependent unit because it is likely very many errors will be encountered as a consequence.

65. Left side cannot be assigned to

Complete list of compiler error messages

This error message is given when you try to modify a read-only object like a constant, a constant parameter, or the return value of function.

```
program Produce;

const
  c = 1;

procedure p(const s: string);
begin
  s := 'changed';             (*<-- Error message here*)
end;

function f: PChar;
begin
  f := 'Hello';               (*This is fine - we are setting the
return value*)
end;

begin
  c := 2;                     (*<-- Error message here*)
  f := 'h';                   (*<-- Error message here*)
end.
```

The example assigns to constant parameter, to a constant, and to the result of a function call. All of these are illegal.

```
program Solve;

var
  c : Integer = 1;            (*Use an initialized variable*)

procedure p(var s: string);
begin
  s := 'changed';             (*Use variable parameter*)
end;

function f: PChar;
begin
  f := 'Hello';               (*This is fine - we are setting the
return value*)
end;

begin
  c := 2;
  f^ := 'h';                  (*This compiles, but will crash at
runtime*)
end.
```

There two ways you can solve this kind of problem: either you change the definition of whatever you are assigning to, so the assignment becomes legal, or you eliminate the assignment.

66. Unsatisfied forward or external declaration: '<Procedurename>'

Complete list of compiler error messages

This error message appears when you have a forward or external declaration of a procedure or function, or a declaration of a method in a class or object type, and you don't define the procedure, function or method anywhere.

Maybe the definition is really missing, or maybe its name is just misspelled.

Note that a declaration of a procedure or function in the interface section of a unit is equivalent to a forward declaration - you have to supply the implementation (the body of the procedure or function) in the implementation section.

Similarly, the declaration of a method in a class or object type is equivalent to a forward declaration.

```
program Produce;

type
  TMyClass = class
  constructor Create;
  end;

function Sum(const a: array of Double): Double; forward;

function Summ(const a: array of Double): Double;
var
  i: Integer;
begin
  Result := 0.0;
  for i:= 0 to High(a) do
  Result := Result + a[i];
end;

begin
end.
```

The definition of Sum in the above example has an easy-to-spot typo.

```
program Solve;

type
  TMyClass = class
  constructor Create;
  end;

constructor TMyClass.Create;
begin
end;

function Sum(const a: array of Double): Double; forward;

function Sum(const a: array of Double): Double;
var
  i: Integer;
begin
  Result := 0.0;
  for i:= 0 to High(a) do
  Result := Result + a[i];
end;

begin
end.
```

The solution: make sure the definitions of your procedures, functions and methods are all there, and spelled correctly.

67. Missing operator or semicolon

Complete list of compiler error messages

This error message appears if there is no operator between two subexpressions, or no semicolon between two statements.

Often, a semicolon is missing on the previous line.

```
program Produce;
var
  I: Integer;
begin
  I := 1 2                  (*<-- Error message here*)
  if I = 3 then             (*<-- Error message here*)
  Writeln('Fine')
end.
```

The first statement in the example has two errors - a '+' operator and a semicolon are missing. The first error is reported on this statement, the second on the following line.

```
program Solve;
var
  I: Integer;
begin
  I := 1 + 2;               (*We were missing a '+' operator and a
semicolon*)
  if I = 3 then
  Writeln('Fine')
end.
```

The solution is to make sure the necessary operators and semicolons are there.

68. Missing parameter type

Complete list of compiler error messages

This error message is issued when a parameter list gives no type for a value parameter.

Leaving off the type is legal for constant and variable parameters.

```
program Produce;

procedure P(I;J: Integer);                                (*<-- Error
message here*)
begin
end;

function ComputeHash(Buffer; Size: Integer): Integer; (*<-- Error
message here*)
begin
end;


begin
end.
```

We intended procedure P to have two integer parameters, but we put a semicolon instead of a comma after the first parameters. The function ComputeHash was supposed to have an untyped first parameter, but untyped parameters must be either variable or constant parameters - they cannot be value parameters.

```
program Solve;

procedure P(I,J: Integer);
begin
end;

function ComputeHash(const Buffer; Size: Integer): Integer;
begin
end;

begin
end.
```

The solution in this case was to fix the type in P's parameter list, and to declare the Buffer parameter to ComputeHash as a constant parameter, because we don't intend to modify it.

69. Illegal reference to symbol '<Name>' in object file '<Filename>'

Complete list of compiler error messages

This error message is given if an object file loaded with a $L or $LINK directive contains a reference to a Pascal symbol that is not a procedure, function, variable, typed constant or thread local variable.

70. Line too long (more than 255 characters)

Complete list of compiler error messages

This error message is given when the length of a line in the source file exceeds 255 characters.

Usually, you can divide the long line into two shorter lines.

If you need a really long string constant, you can break it into several pieces on consecutive lines that you concatenate with the '+' operator.

71. Unknown directive: '<Directive>'

Complete list of compiler error messages

This error message appears when the compiler encounters an unknown directive in a procedure or function declaration.

The directive is probably misspelled, or a semicolon is missing.

```
program Produce;

procedure P; stcall;
begin
end;

procedure Q forward;

function GetLastError: Integer external 'kernel32.dll';

begin
end.
```

In the declaration of P, the calling convention "stdcall" is misspelled. In the declaration of Q and GetLastError, we're missing a semicolon.

```
program Solve;

procedure P; stdcall;
begin
end;

procedure Q; forward;

function GetLastError: Integer; external 'kernel32.dll';

begin
end.
```

The solution is to make sure the directives are spelled correctly, and that the necessary semicolons are there.

72. This type cannot be initialized

Complete list of compiler error messages

File types (including type Text), and the type Variant cannot be initialized, that is, you cannot declare typed constants or initialized variables of these types.

```
program Produce;

var
  V: Variant = 0;

begin
end.
```

The example tries to declare an initialized variable of type Variant, which illegal.

```
program Solve;

var
  V: Variant;

begin
  V := 0;
end.
```

The solution is to initialize a normal variable with an assignment statement.

73. Number of elements differs from declaration

Complete list of compiler error messages

This error message appears when you declare a typed constant or initialized variable of array type, but do not supply the appropriate number of elements.

```
program Produce;

var
  A : array [1..10] of Integer = (1,2,3,4,5,6,7,8,9);

begin
end.
```

The example declares an array of 10 elements, but the initialization only supplies 9 elements.

```
program Solve;

var
  A : array [1..10] of Integer = (1,2,3,4,5,6,7,8,9,10);

begin
end.
```

We just had to supply the missing element to make the compiler happy. When initializing bigger arrays, it can be sometimes hard to see whether you have supplied the right number of elements. To help with that, you layout the source file in a way that makes counting easy (e.g. ten elements to a line), or you can put the index of an element in comments next to the element itself.

74. Label already defined: '<Labelname>'

Complete list of compiler error messages

This error message is given when a label is set on more than one statement.

```
program Produce;
label 1;
begin
1:
  goto 1;
1:        (*<-- Error message here*)
end.
```

The example just tries to set label 1 twice.

```
program Solve;
label 1;
begin
1:
  goto 1;
end.
```

Make sure every label is set exactly once.

75. Label declared and referenced, but not set: '<label>'

Complete list of compiler error messages

You declared and used a label in your program, but the label definition was not encountered in the source code.

```
program Produce;

  procedure Labeled;
  label 10;
  begin
    goto 10;
  end;

begin
end.
```

Label 10 is declared and used in the procedure 'Labeled', but the compiler never finds a definition of the label.

```
program Produce;

  procedure Labeled;
  label 10;
  begin
    goto 10;
    10:
  end;

begin
end.
```

The simple solution is to ensure that a declared and used label has a definition, in the same scope, in your program.

76. This form of method call only allowed in methods of derived types

Complete list of compiler error messages

This error message is issued if you try to make a call to a method of an ancestor type, but you are in fact not in a method.

```
program Produce;

type
  TMyClass = class
    constructor Create;
  end;

procedure Create;
begin
  inherited Create;      (*<-- Error message here*)
end;

begin
end.
```

The example tries to call an inherited constructor in procedure Create, which is not a method.

```
program Solve;

type
  TMyClass = class
    constructor Create;
  end;

constructor TMyclass.Create;
begin
  inherited Create;
end;

begin
end.
```

The solution is to make sure you are in fact in a method when using this form of call.

77. This form of method call only allowed for class methods

Complete list of compiler error messages

You were trying to call a normal method by just supplying the class type, not an actual instance.

This is only allowed for class methods and constructors, not normal methods and destructors.

```
program Produce;

type
  TMyClass = class
  (*...*)
  end;
var
  MyClass: TMyClass;

begin
  MyClass := TMyClass.Create;  (*Fine, constructor*)
  Writeln(TMyClass.ClassName); (*Fine, class method*)
  TMyClass.Destroy;            (*<-- Error message here*)
end.
```

The example tries to destroy the type TMyClass - this doesn't make sense and is therefore illegal.

```
program Solve;
type
  TMyClass = class
  (*...*)
  end;
var
  MyClass: TMyClass;

begin
  MyClass := TMyClass.Create;  (*Fine, constructor*)
  Writeln(TMyClass.ClassName); (*Fine, class method*)
  MyClass.Destroy;             (*Fine, called on instance*)
end.
```

As you can see, we really meant to destroy the instance of the type, not the type itself.

78. Variable '<Name>' might not have been initialized

Complete list of compiler error messages

This warning is given if a variable has not been assigned a value on every code path leading to a point where it is used.

```
program Produce;
(*$WARNINGS ON*)
var
  B: Boolean;
  C: (Red,Green,Blue);

procedure Simple;
var
  I : Integer;
begin
  Writeln(I);        (*<-- Warning here*)
end;

procedure IfStatement;
var
  I : Integer;
begin
  if B then
    I := 42;
  Writeln(I);        (*<-- Warning here*)
end;

procedure CaseStatement;
var
  I: Integer;
begin
  case C of
  Red..Blue: I := 42;
  end;
  Writeln(I);        (*<-- Warning here*)
end;

procedure TryStatement;
var
  I: Integer;
begin
  try
    I := 42;
  except
    Writeln('Should not get here!');
  end;
  Writeln(I);        (*<-- Warning here*)
end;

begin
  B := False;
end.
```

In an if statement, you have to make sure the variable is assigned in both branches. In a case

statement, you need to add an else part to make sure the variable is assigned a value in every conceivable case. In a try-except construct, the compiler assumes that assignments in the try part may in fact not happen, even if they are at the very beginning of the try part and so simple that they cannot conceivably cause an exception.

```
program Solve;
(*$WARNINGS ON*)
var
  B: Boolean;
  C: (Red,Green,Blue);

procedure Simple;
var
  I : Integer;
begin
  I := 42;
  Writeln(I);
end;

procedure IfStatement;
var
  I : Integer;
begin
  if B then
    I := 42
  else
    I := 0;
  Writeln(I);        (*Need to assign I in the else part
end;

procedure CaseStatement;
var
  I: Integer;
begin
  case C of
  Red..Blue: I := 42;
  else       I := 0;
  end;
  Writeln(I);        (*Need to assign I in the else part*)
end;

procedure TryStatement;
var
  I: Integer;
begin
  I := 0;
  try
    I := 42;
  except
    Writeln('Should not get here!');
  end;
  Writeln(I);        (*Need to assign I before the try*)
end;

begin
  B := False;
end.
```

The solution is to either add assignments to the code paths where they were missing, or to add an assignment before a conditional statement or a try-except construct.

79. Value assigned to '<Name>' never used

Complete list of compiler error messages

The compiler gives this hint message if the value assigned to a variable is not used. If optimization is enabled, the assignment is eliminated.

This can happen because either the variable is not used anymore, or because it is reassigned before it is used.

```
program Produce;
(*$HINTS ON*)

procedure Simple;
var
  I: Integer;
begin
  I := 42;                (*<-- Hint message here*)
end;

procedure Propagate;
var
  I: Integer;
  K: Integer;
begin
  I := 0;                 (*<-- Hint message here*)
  Inc(I);                 (*<-- Hint message here*)
  K := 42;
  while K > 0 do begin
    if Odd(K) then
      Inc(I);             (*<-- Hint message here*)
    Dec(K);
  end;
end;

procedure TryFinally;
var
  I: Integer;
begin
  I := 0;                 (*<-- Hint message here*)
  try
    I := 42;
  finally
    Writeln('Reached finally');
  end;
  Writeln(I);             (*Will always write 42 - if an
exception happened,
          we wouldn't get here*)
end;

begin
end.
```

In procedure Propagate, the compiler is smart enough to realize that as variable I is not used after the while loop, it does not need to be incremented inside the while, and therefore the increment and the assignment before the while loop are also superfluous.

In procedure TryFinally, the assignment to I before the try-finally construct is not necessary. If an

exception happens, we don't execute the Writeln statement at the end, so the value of I does not matter. If no exception happens, the value of I seen by the Writeln statement is always 42. So the first assignment will not change the behavior of the procedure, and can therefore be eliminated.

This hint message does not indicate your program is wrong - it just means the compiler has determined there is an assignment that is not necessary.

You can usually just delete this assignment - it will be dropped in the compiled code anyway if you compile with optimizations on.

Sometimes, however, the real problem is that you assigned to the wrong variable, e.g. to meant to assign J but instead assigned I. So it is worthwhile to check the assignment in question carefully.

80. Return value of function '<Functionname>' might be undefined

Complete list of compiler error messages

This warning is given if the return value of a function has not been assigned a value on every code path.

To put it a little differently, the function could execute in a way that never assigns anything to the return value.

```
program Produce;
(*$WARNINGS ON*)
var
  B: Boolean;
  C: (Red,Green,Blue);

function Simple: Integer;
begin
end;                          (*<-- Warning here*)

function IfStatement: Integer;
begin
  if B then
    Result := 42;
end;                            (*<-- Warning here*)

function CaseStatement: Integer;
begin
  case C of
  Red..Blue: Result := 42;
  end;
end;                            (*<-- Warning here*)

function TryStatement: Integer;
begin
  try
    Result := 42;
  except
    Writeln('Should not get here!');
  end;
end;                            (*<-- Warning here*)

begin
  B := False;
end.
```

The problem with procedure IfStatement and CaseStatement is that the result is not assigned in every code path. In TryStatement, the compiler assumes that an exception could happen before Result is assigned.

```
program Solve;
(*$WARNINGS ON*)
var
  B: Boolean;
  C: (Red,Green,Blue);

function Simple: Integer;
begin
  Result := 42;
end;

function IfStatement: Integer;
begin
  if B then
    Result := 42
  else
    Result := 0;
end;

function CaseStatement: Integer;
begin
  case C of
  Red..Blue: Result := 42;
  else       Result := 0;
  end;
end;

function TryStatement: Integer;
begin
  Result := 0;
  try
    Result := 42;
  except
    Writeln('Should not get here!');
  end;
end;

begin
  B := False;
end.
```

The solution is to make sure there is an assignment to the result variable in every possible code path.

81. Procedure FAIL only allowed in constructor

Complete list of compiler error messages

The standard procedure Fail can only be called from within a constructor - it is illegal otherwise.

82. Procedure NEW needs constructor

Complete list of compiler error messages

This error message is issued when an identifier given in the parameter list to New is not a constructor.

```
program Produce;

type
  PMyObject = ^TMyObject;
  TMyObject = object
  F: Integer;
  constructor Init;
  destructor Done;
  end;

constructor TMyObject.Init;
begin
  F := 42;
end;

destructor TMyObject.Done;
begin
end;

var
  P: PMyObject;

begin
  New(P, Done);               (*<-- Error message here*)
end.
```

By mistake, we called New with the destructor, not the constructor.

```
program Solve;

type
  PMyObject = ^TMyObject;
  TMyObject = object
  F: Integer;
  constructor Init;
  destructor Done;
  end;

constructor TMyObject.Init;
begin
  F := 42;
end;

destructor TMyObject.Done;
begin
end;

var
  P: PMyObject;

begin
  New(P, Init);
end.
```

Make sure you give the New standard function a constructor, or no additional argument at all.

83. Procedure DISPOSE needs destructor

Complete list of compiler error messages

This error message is issued when an identifier given in the parameter list to Dispose is not a destructor.

```
program Produce;

type
  PMyObject = ^TMyObject;
  TMyObject = object
  F: Integer;
  constructor Init;
  destructor Done;
  end;

constructor TMyObject.Init;
begin
  F := 42;
end;

destructor TMyObject.Done;
begin
end;

var
  P: PMyObject;

begin
  New(P, Init);
  (*...*)
  Dispose(P, Init);         (*<-- Error message here*)
end.
```

In this example, we passed the constructor to Dispose by mistake.

```
program Solve;

type
  PMyObject = ^TMyObject;
  TMyObject = object
  F: Integer;
  constructor Init;
  destructor Done;
  end;

constructor TMyObject.Init;
begin
  F := 42;
end;

destructor TMyObject.Done;
begin
end;

var
  P: PMyObject;

begin
  New(P, Init);
  Dispose(P, Done);
end.
```

The solution is to either pass a destructor to Dispose, or to eliminate the second argument.

84. Assignment to FOR-Loop variable '<Name>'

Complete list of compiler error messages

It is illegal to assign a value to the for loop control variable inside the for loop.

If the purpose is to leave the loop prematurely, use a break or goto statement.

```
program Produce;

var
  I: Integer;
  A: array [0..99] of Integer;
begin
  for I := 0 to 99 do begin
    if A[I] = 42 then
      I := 99;
  end;
end.
```

In this case, the programmer thought that assigning 99 to I would cause the program to exit the loop.

```
program Solve;

var
  I: Integer;
  A: array [0..99] of Integer;
begin
  for I := 0 to 99 do begin
    if A[I] = 42 then
      Break;
  end;
end.
```

Using a break statement is a cleaner way to exit out of a for loop.

85. FOR-Loop variable '<Name>' may be undefined after loop

Complete list of compiler error messages

This warning is issued if the value of a for loop control variable is used after the loop.

You can only rely on the final value of a for loop control variable if the loop is left with a goto or exit statement.

The purpose of this restriction is to enable the compiler to generate efficient code for the for loop.

```
program Produce;
(*$WARNINGS ON*)

function Search(const A: array of Integer; Value: Integer):
Integer;
begin
  for Result := 0 to High(A) do
    if A[Result] = Value then
      break;
end;

const
  A : array [0..9] of Integer = (1,2,3,4,5,6,7,8,9,10);

begin
  Writeln( Search(A,11) );
end.
```

In the example, the Result variable is used implicitly after the loop, but it is undefined if we did not find the value - hence the warning.

```
program Solve;
(*$WARNINGS ON*)

function Search(const A: array of Integer; Value: Integer):
Integer;
begin
  for Result := 0 to High(A) do
    if A[Result] = Value then
      exit;
  Result := High(a)+1;
end;

const
  A : array [0..9] of Integer = (1,2,3,4,5,6,7,8,9,10);

begin
  Writeln( Search(A,11) );
end.
```

The solution is to assign the intended value to the control variable for the case where we don't exit the loop prematurely.

86. TYPEOF can only be applied to object types with a VMT

Complete list of compiler error messages

This error message is issued if you try to apply the standard function TypeOf to an object type that does not have a virtual method table.

A simple workaround is to declare a dummy virtual procedure to force the compiler to generate a VMT.

```
program Produce;

type
  TMyObject = object
    procedure MyProc;
  end;

procedure TMyObject.MyProc;
begin
  (*...*)
end;

var
  P: Pointer;
begin
  P := TypeOf(TMyObject);    (*<-- Error message here*)
end.
```

The example tries to apply the TypeOf standard function to type TMyObject which does not have virtual functions, and therefore no virtual function table (VMT).

```
program Solve;

type
  TMyObject = object
    procedure MyProc;
    procedure Dummy; virtual;
  end;

procedure TMyObject.MyProc;
begin
  (*...*)
end;

procedure TMyObject.Dummy;
begin
end;

var
  P: Pointer;
begin
  P := TypeOf(TMyObject);
end.
```

The solution is to introduce a dummy virtual function, or to eliminate the call to TypeOf.

87. Order of fields in record constant differs from declaration

Complete list of compiler error messages

This error message occurs if record fields in a typed constant or initialized variable are not initialized in declaration order.

```
program Produce;

type
  TPoint = record
    X, Y: Integer;
  end;

var
  Point : TPoint = (Y: 123; X: 456);

begin
end.
```

The example tries to initialize first Y, then X, in the opposite order from the declaration.

```
program Solve;

type
  TPoint = record
    X, Y: Integer;
  end;

var
  Point : TPoint = (X: 456; Y: 123);

begin
end.
```

The solution is to adjust the order of initialization to correspond to the declaration order.

88. Internal error: <ErrorCode>

Complete list of compiler error messages

You should never get this error message - it means there is a programming error in the compiler.

If you do, please call Inprise Developer Support and let us know the ErrorCode (e.g. "C1196") that appears in the error message. This will give us a rough indication what went wrong. It is even more helpful if you can give us an example program that produces this message.

89. Unit name mismatch: '<Unitname>'

Complete list of compiler error messages

This error message is issued if after loading a unit mentioned in a uses clause, the compiler finds that it does not have the requested name.

This can happen for instance because file names may be truncated to the MSDOS 8.3 filename format.

```
----- Contents of MY_UNIT_.PAS
---------------------------------------------
unit My_Unit_With_A_Long_Name;
interface
implementation
end.
----- End of MY_UNIT_.PAS
-----------------------------------------------


program Produce;
uses My_Unit_With_Another_Long_Name; (*Will find MY_UNIT_.PAS if
-P command line
               switch is active - but it's the wrong unit.*)
begin
end.
```

In this case, the problem is that the compiler found the wrong unit, because the filenames were truncated to 8 characters.

The solution is to use long filenames or to make sure the filenames differ in the first 8 characters. Also, you need to make sure the filename of a unit corresponds to the unit name.

90. Type '<Name>' is not yet completely defined

Complete list of compiler error messages

This error occurs if there is either a reference to a type that is just being defined, or if there is a forward declared class type in a type section and no final declaration of that type.

```
program Produce;

type
  TListEntry = record
    Next: ^TListEntry;                      (*<-- Error message
here*)
    Data: Integer;
  end;
  TMyClass = class;                         (*<-- Error message
here*)
  TMyClassRef = class of TMyClass;
  TMyClasss = class                         (*<-- Typo ...*)
    (*...*)
  end;

begin
end.
```

The example tries to refer to record type before it is completely defined. Also, because of a typo, the compiler never sees a complete declaration for TMyClass.

```
program Solve;

type
  PListEntry = ^TListEntry;
  TListEntry = record
    Next: PListEntry;
    Data: Integer;
  end;
  TMyClass = class;
  TMyClassRef = class of TMyClass;
  TMyClass = class
    (*...*)
  end;

begin
end.
```

The solution for the first problem is to introduce a type declaration for an auxiliary pointer type. The second problem is fixed by spelling TMyClass correctly.

91. This Demo Version has been patched

Complete list of compiler error messages

This error message is currently unused.

92. Integer constant or variable name expected

Complete list of compiler error messages

This error message is issued if you try to declare an absolute variable, but the absolute directive is not followed by an integer constant or a variable name.

```
program Produce;

var
  I : Integer;
  J : Integer absolute Addr(I);    (*<-- Error message here*)

begin
end.
```

```
program Solve;

const
  Addr = 0;

var
  I : Integer;
  J : Integer absolute I;

begin
end.
```

93. Invalid typecast

Complete list of compiler error messages

This error message is issued for type casts not allowed by the rules. The following kinds of casts are allowed:

- Ordinal or pointer type to another ordinal or pointer type

- A character, string, array of character or pchar to a string

- An ordinal, real, string or variant to a variant

- A variant to an ordinal, real, string or variant

- A variable reference to any type of the same size.

Note that casting real types to integer can be performed with the standard functions Trunc and Round.

There are other transfer functions like Ord and Chr that might make your intention clearer.

```
program Produce;

begin
  Writeln( Integer(Pi) );
end.
```

This programmer thought he could cast a floating point constant to Integer, like in C.

```
program Solve;

begin
  Writeln( Trunc(Pi) );
end.
```

In Pascal, we have separate Transfer functions to convert floating point values to integer.

94. User break - compilation aborted

Complete list of compiler error messages

This message is currently unused.

95. Assignment to typed constant '<Name>'

Complete list of compiler error messages

This warning message is currently unused.

96. "Segment/Offset pairs not supported in Borland 32-bit Pascal

Complete list of compiler error messages

32-bit code no longer uses the segment/offset addressing scheme that 16-bit code used.

In 16-bit versions of Borland Pascal, segment/offset pairs were used to declare absolute variables, and as arguments to the Ptr standard function.

Note that absolute addresses should not be used in 32-bit protected mode programs. Instead appropriate Win32 API functions should be called.

```
program Produce;

var
  VideoMode : Integer absolute $0040:$0049;

begin
  Writeln( Byte(Ptr($0040,$0049)^) );
end.
```

```
program Solve;
(*This version will compile, but will not run; absolute addresses
are to be carefully avoided*)
var
  VideoMode : Integer absolute $0040*16+$0049;

begin
  Writeln( Byte(Ptr($0040*16+$0049)^) );
end.
```

97. Program or unit '<name>' recursively uses itself

Complete list of compiler error messages

An attempt has been made for a unit to use itself.

```
unit Produce;
interface
  uses Produce;
implementation

begin
end.
```

In the above example the uses clause specifies the same unit, which causes the compiler to emit an error message.

```
unit Solve;
interface
implementation

begin
end.
```

The only solution to this problem is to remove the offending uses clause.

98. Label '<Name>' is not declared in current procedure

Complete list of compiler error messages

In contrast to Standard Pascal, Borland Pascal does not allow a goto to jump out of the current procedure.

However, his construct is mainly useful for error handling, and Borland Pascal provides a more general and structured mechanism to deal with errors: exception handling.

```
program Produce;

label 99;

procedure MyProc;
begin
  (*Something goes very wrong...*)
  goto 99;
end;

begin
  MyProc;
  99:
    Writeln('Fatal error');
end.
```

The example above tries to halt computation by doing a non-local goto.

```
program Solve;

uses SysUtils;

procedure MyProc;
begin
  (*Something goes very wrong...*)
  raise Exception.Create('Fatal error');
end;

begin
  try
    MyProc;
  except
    on E: Exception do Writeln(E.Message);
  end;
end.
```

In our solution, we used exception handling to stop the program. This has the advantage that we can also pass an error message. Another solution would be to use the standard procedures Halt or RunError.

99. "Local procedure/function '<Name>' assigned to procedure variable

Complete list of compiler error messages

This error message is issued if you try to assign a local procedure to a procedure variable, or pass it as a procedural parameter.

This is illegal, because the local procedure could then be called even if the enclosing procedure is not active. This situation would cause the program to crash if the local procedure tried to access any variables of the enclosing procedure.

```
program Produce;

var
  P: Procedure;

procedure Outer;

  procedure Local;
  begin
    Writeln('Local is executing');
  end;

begin
  P := Local;       (*<-- Error message here*)
end;

begin
  Outer;
  P;
end.
```

The example tries to assign a local procedure to a procedure variable. This is illegal because it is unsafe at run time.

```
program Solve;

var
  P: Procedure;

procedure NonLocal;
begin
  Writeln('NonLocal is executing');
end;

procedure Outer;

begin
  P := NonLocal;
end;

begin
  Outer;
  P;
end.
```

The solution is to move the local procedure out of the enclosing one.

100. Missing ENDIF directive

Complete list of compiler error messages

This error message is issued if the compiler does not find a corresponding $ENDIF directive after an $IFDEF, $IFNDEF or $IFOPT directive.

```
program Produce;
(*$APPTYPE CONSOLE*)
begin
(*$IfOpt O+*)
  Writeln('Compiled with optimizations');
(*$Else*)
  Writeln('Compiled without optimizations');
(*Endif*)
end.                                          (*<-- Error
message here*)
```

In this example, we left out the $ character in the (*$Endif*) directive, so the compiler mistook it for a comment.

```
program Solve;
(*$APPTYPE CONSOLE*)
begin
(*$IfOpt O+*)
  Writeln('Compiled with optimizations');
(*$Else*)
  Writeln('Compiled without optimizations');
(*$Endif*)
end.
```

The solution is to make sure all the conditional directives have a valid $ENDIF directive.

101. Method identifier expected

Complete list of compiler error messages

This error message will be issued in several different situations:

- Properties in an automated section must use methods for access, they cannot use fields in their read or write clauses.

- You tried to call a class method with the "ClassType.MethodName" syntax, but "MethodName" was not the name of a method.

- You tried calling an inherited with the "Inherited MethodName" syntax, but "MethodName" was not the name of a method.

```
program Produce;

type
  TMyBase = class
    Field: Integer;
  end;
  TMyDerived = class (TMyBase)
    Field: Integer;
    function Get: Integer;
  Automated
    property Prop: Integer read Field;    (*<-- Error message
here*)
  end;

function TMyDerived.Get: Integer;
begin
  Result := TMyBase.Field;                (*<-- Error message
here*)
end;

begin
end.
```

The example tried to declare an automated property that accesses a field directly. The second error was caused by trying to get at a field of the base class - this is also not legal.

```
program Solve;

type
  TMyBase = class
    Field: Integer;
  end;
  TMyDerived = class (TMyBase)
    Field: Integer;
    function Get: Integer;
  Automated
    property Prop: Integer read Get;
  end;

function TMyDerived.Get: Integer;
begin
  Result := TMyBase(Self).Field;
end;

begin
  Writeln( TMyDerived.Create.Prop );
end.
```

The first problem is fixed by accessing the field via a method. The second problem can be fixed by casting the Self pointer to the base class type, and accessing the field off of that.

102. FOR-Loop variable '<name>' cannot be passed as var parameter

Complete list of compiler error messages

An attempt has been made to pass the control variable of a FOR-loop to a procedure or function which takes a var parameter. This is an error because the procedure which receives the control variable is able to modify it, thereby changing the semantics of the FOR-loop which issued the call.

```
program Produce;

  procedure p1(var x : Integer);
  begin
  end;

  procedure p0;
    var
      i : Integer;
  begin
    for i := 0 to 1000 do
      p1(i);
  end;

begin
end.
```

In this example, the loop control variable, i, is passed to a procedure which receives a var parameter. This is the main cause of the error.

```
program Solve;
  procedure p1(x : Integer);
  begin
  end;

  procedure p0;
    var
      i : Integer;
  begin
    i := 0;
    while i <= 1000 do
      p1(i);
  end;


begin
end.
```

The easiest way to approach this problem is to change the parameter into a by-value parameter. However, there may be a good reason that it was a by-reference parameter in the begging, so you must be sure that this change of semantics in your program does not affect other code. Another way to approach this problem is change the for loop into an equivalent while loop, as is done in the above program.

103. Typed constant '<name>' passed as var parameter

Complete list of compiler error messages

This error message is reserved.

104. BREAK or CONTINUE outside of loop

Complete list of compiler error messages

The compiler has found a BREAK or CONTINUE statement which is not contained inside a WHILE or REPEAT loop. These two constructs are only legal in loops.

```
program Produce;

  procedure Error;
    var i : Integer;
  begin
    i := 0;
    while i < 100 do
      INC(i);
      if odd(i) then begin
        INC(i);
  continue;
      end;
  end;

begin
end.
```

The example above shows how a continue statement could seem to be included in the body of a looping construct but, due to the compound-statement nature of Pascal, it really is not.

```
program Solve;

  procedure Error;
    var i : Integer;
  begin
    i := 0;
    while i < 100 do begin
      INC(i);
      if odd(i) then begin
        INC(i);
  continue;
      end;
    end;
  end;

begin
end.
```

Often times it is a simple matter to create compound statement out of the looping construct to ensure that your CONTINUE or BREAK statements are included.

105. Division by zero

Complete list of compiler error messages

The compiler has detected a constant division by zero in your program.

Check your constant expressions and respecify them so that a division by zero error will not occur.

106. Overflow in conversion or arithmetic operation

Complete list of compiler error messages

The compiler has detected an overflow in an arithmetic expression: the result of the expression is too large to be represented in 32 bits.

Check your computations to ensure that the value can be represented by the computer hardware.

107. Data type too large: exceeds 2 GB

Complete list of compiler error messages

You have specified a data type which is too large for the compiler to represent. The compiler will generate this error for datatypes which are greater or equal to 2 GB in size. You must decrease the size of the description of the type.

```
program Produce;

  type
    EnormousArray = array [0..MaxLongint] OF Longint;
    BigRecord = record
      points : array [1..10000] of Extended;
    end;

  var
    data : array [0..500000] of BigRecord;

begin
end.
```

It is easily apparent to see why these declarations will elicit error messages.

```
program Solve;
  type
    EnormousArray = array [0..MaxLongint DIV 8] OF Longint;

    DataPoints = ^DataPointDesc;
    DataPointDesc = array [1..10000] of Extended;
    BigRecord = record
      points : DataPoints;
    end;

  var
    data : array [0..500000] OF BigRecord;


begin
end.
```

The easy solution to avoid this error message is to make sure that the size of your data types remain under 2Gb in size. A more complicated method would involve the restructuring of your data, as has been begun with the BigRecord declaration.

108. Integer constant too large

Complete list of compiler error messages

You have specified an integer constant that requires more than 64 bits to represent.

```
program Produce;

  const
    VeryBigHex = $80000000000000001;

begin
end.
```

The constant in the above example is too large to represent in 64 bits, thus the compiler will output an error.

```
program Solve;

  const
    BigHex = $8000000000000001;

begin
end.
```

Check the constants that you have specified and ensure that they are representable in 64 bits.

109. 16-Bit fixup encountered in object file '<name>'

Complete list of compiler error messages

A 16-bit fixup has been found in one of the object modules linked to your program with the $L compiler directive. The compiler only supports 32 bit fixups in linked object modules.

Make sure that the linked object module is a 32 bit object module.

110. Inline assembler syntax error

Complete list of compiler error messages

You have entered an expression which the inline assembler is unable to interpret as a valid assembly instruction.

```
program Produce;

  procedure Assembly;
  asm
    adx  eax, 151
  end;

begin
end.
```

```
program Solve;

  procedure Assembly;
  asm
    add  eax, 151
  end;

begin
end.
```

Examine the offending inline assembly statement and ensure that it conforms to the proper syntax.

111. Inline assembler stack overflow

Complete list of compiler error messages

Your inline assembler code has exceeded the capacity of the inline assembler.

Contact Inprise if you encounter this error.

112. Operand size mismatch

Complete list of compiler error messages

The size required by the instruction operand does not match the size given.

```
program Produce;

  var
    v : Integer;

  procedure Assembly;
  asm
    db offset v
  end;

begin
end.
```

In the sample above, the compiler will complain because the 'offset' operator produces a 'dword', but the operator is expecting a 'byte'.

```
program Solve;

  var
    v : Integer;

  procedure Assembly;
  asm
    dd offset v
  end;

begin
end.
```

The solution, for this example, is to change the operator to receive a 'dword'. In the general case you will need to closely examine your code and ensure that the operator and operand sizes match.

113. Memory reference expected

Complete list of compiler error messages

The inline assembler has expected to find a memory reference expression but did not find one.

Ensure that the offending statement is indeed a memory reference.

114. Constant expected

Complete list of compiler error messages

The inline assembler was expecting to find a constant but did not find one.

```
program Produce;

  procedure Assembly(x : Integer);
  asm
    mov   ax, x MOD 10
  end;

begin
end.
```

The inline assembler is not capable of performing a MOD operation on a Pascal variable, thus the above code will cause an error.

Many of the inline assembler expressions require constants to assemble correctly. Change the offending statement to have a assemble-time constant.

115. Type expected

Complete list of compiler error messages

Contact Inprise if you receive this error.

116. Cannot add or subtract relocatable symbols

Complete list of compiler error messages

The inline assembler is not able to add or subtract memory address which may be changed by the linker.

```
program Produce;

  var
    a : array [1..10] of Integer;
    endOfA : Integer;

  procedure Relocatable;
  begin
  end;

  procedure Assembly;
  asm
    mov eax, a + endOfA
  end;

begin
end.
```

Global variables fall into the class of items which produce relocatable addresses, and the inline assembler is unable to add or subtract these.

Make sure you don't try to add or subtract relocatable addresses from within your inline assembler statements.

117. Invalid register combination

Complete list of compiler error messages

You have specified an illegal combination of registers in a inline assembler Please refer to an assembly language guide for more information on addressing modes allowed on the Intel 80x86 family. statement.

```
program Produce;

  procedure AssemblerExample;
  asm
    mov eax, [ecx + esp * 4]
  end;

begin
end.
```

The right operand specified in this mov instruction is illegal.

```
program Solve;

  procedure AssemblerExample;
  asm
    mov eax, [ecx + ebx * 4]
  end;

begin
end.
```

The addressing mode specified by the right operand of this mov instruction is allowed.

118. Numeric overflow

Complete list of compiler error messages

The inline assembler has detected a numeric overflow in one of your expressions.

```
program Produce;

  procedure AssemblerExample;
  asm
    mov eax, $0fffffffffffffffffff
  end;

begin
end.
```

Specifying a number which requires more than 32bits to represent will elicit this error.

```
program Solve;

  procedure AssemblerExample;
  asm
    mov al, $0ff
  end;

begin
end.
```

Make sure that your numbers all fit in 32bits.

119. String constant too long

Complete list of compiler error messages

The inline assembler has not found the end of the string that you specified. The most likely cause is a misplaced closing quote.

```
program Produce;

  procedure AssemblerExample;
  asm
    db 'Hello world.  I am an inline assembler statement
  end;

begin
end.
```

The inline assembler is unable to find the end of the string, before the end of the line, so it reports that the string is too long.

```
program Solve;

  procedure AssemblerExample;
  asm
    db 'Hello world.  I am an inline assembler statement'
  end;

begin
end.
```

Adding the closing quote will vanquish this error.

120. Error in numeric constant

Complete list of compiler error messages

The inline assembler has found an error in the numeric constant you entered.

```
program Produce;

  procedure AssemblerExample;
  asm
    mov al, $z0f0
  end;

begin
end.
```

In the example above, the inline assembler was expecting to parse a hexadecimal constant, but it found an erroneous character.

```
program Solve;

  procedure AssemblerExample;
  asm
    mov al, $f0
  end;

begin
end.
```

Make sure that the numeric constants you enter conform to the type that the inline assembler is expecting to parse.

121. Invalid combination of opcode and operands

Complete list of compiler error messages

You have specified an inline assembler statement which is not correct.

```
program Produce;

  procedure AssemblerExample;
  asm
    mov al, $0f0 * 16
  end;

begin
end.
```

The inline assembler is not capable of storing the result of $f0 * 16 into the 'al' register—it simply won't fit.

```
program Solve;
  procedure AssemblerExample;
  asm
    mov al, $0f * 16
  end;

begin
end.
```

Make sure that the type of both operands are compatible.

122. 486/487 instructions not enabled

Complete list of compiler error messages

You should not receive this error as 486 instructions are always enabled.

123. Division by zero

Complete list of compiler error messages

The inline assembler has encountered an expression which results in a division by zero.

```
program Produce;

  procedure AssemblerExample;
  asm
    dw  1000 / 0
  end;

begin
end.
```

If you are using program constants instead of constant literals this error might not be quite so obvious.

```
program Solve;

  procedure AssemblerExample;
  asm
    dw  1000 / 10
  end;

begin
end.
```

The solution, as when programming in high level languages, is to make sure that you don't divide by zero.

124. Structure field identifier expected

Complete list of compiler error messages

The inline assembler recognized an identifier on the right side of a '.', but it was not a field of the record found on the left side of the '.'. One common, yet difficult to realize, error of this sort is to use a record with a field called 'ch' - the inline assembler will always interpret 'ch' to be a register name.

```
program Produce;

  type
    Data = record
      x : Integer;
    end;

  procedure AssemblerExample(d : Data; y : Char);
  asm
    mov  eax, d.y
  end;

begin
end.
```

In this example, the inline assembler has recognized that 'y' is a valid identifier, but it has not found 'y' to be a member of the type of 'd'.

```
program Solve;

  type
    Data = record
      x : Integer;
    end;

  procedure AssemblerExample(d : Data; y : Char);
  asm
    mov  eax, d.x
  end;

begin
end.
```

By specifying the proper variable name, the error will go away.

125. LOOP/JCXZ distance out of range

Complete list of compiler error messages

You have specified a LOOP or JCXZ destination which is out of range. You should not receive this error as the jump range is 2Gb for LOOP and JCXZ instructions.

126. Procedure or function name expected

Complete list of compiler error messages

You have specified an identifier which does not represent a procedure or function in an EXPORTS clause.

```
library Produce;

  var
   y : procedure;

exports y;
begin
end.
```

It is not possible to export variables from a Delphi library, even though the variable is of 'procedure' type.

```
program Solve;

  procedure ExportMe;
  begin
  end;

exports ExportMe;
begin
end.
```

Always be sure that all the identifiers listed in an EXPORTS clause truly represent procedures.

127. PROCEDURE or FUNCTION expected

Complete list of compiler error messages

This error message is produced by two different constructs, but in both cases the compiler is expecting to find the keyword 'procedure' or the keyword 'function'.

```
program Produce;

  type
    Base = class
      class AProcedure; (*case 1*)
    end;

  class Base.AProcedure; (*case 2*)
  begin
  end;

begin
end.
```

In both cases above, the word 'procedure' should follow the keyword 'class'.

```
program Solve;

  type
    Base = class
      class procedure AProcedure;
    end;

  class procedure Base.AProcedure;
  begin
  end;

begin
end.
```

As can be seen, adding the keyword 'procedure' removes the error from this program.

128. Instance variable '<name>' inaccessible here

Complete list of compiler error messages

You are attempting to reference a instance variable from within a class procedure.

```
program Produce;

  type
    Base = class
      Title : String;

      class procedure Init;
    end;

  class procedure Base.Init;
  begin
    Self.Title := 'Does not work';
    Title := 'Does not work';
  end;

begin
end.
```

Class procedures do not have an instance pointer, so they cannot access any methods or instance data of the class.

```
program Solve;

  type
    Base = class
      Title : String;

      class procedure Init;
    end;

  class procedure Base.Init;
  begin
  end;

begin
end.
```

The only solution to this error is to not access any member data or methods from within a class method.

129. EXCEPT or FINALLY expected

Complete list of compiler error messages

The compiler was expecting to find a FINALLY or EXCEPT keyword, during the processing of exception handling code, but did not find either.

```
program Produce;

begin
  try
  end;
end.
```

In the code above, the 'except' or 'finally' clause of the exception handling code is missing, so the compiler will issue an error.

```
program Solve;

begin
  try
  except
  end;
end.
```

By adding the missing clause, the compiler will be able to complete the compilation of the code. In this case, the 'except' clause will easily allow the program to finish.

130. Cannot BREAK, CONTINUE or EXIT out of a FINALLY clause

Complete list of compiler error messages

Because a FINALLY clause may be entered and exited through Delphi's exception handling mechanism or through normal program control, the explicit control flow of your program may not be followed. When the FINALLY is entered through the exception handling mechanism, it is not possible to exit the clause with BREAK, CONTINUE, or EXIT - when the finally clause is being executed by the exception handling system, control must return to the exception handling system.

```
program Produce;

  procedure A0;
  begin
    try
      (* try something that might fail *)
    finally
      break;
    end;
  end;

  begin
  end.
```

The program above attempts to exit the finally clause with a break statement. It is not legal to exit a FINALLY clause in this manner.

```
program Solve;

  procedure A0;
  begin
    try
      (* try something that might fail *)
    finally
    end;
  end;

  begin
  end.
```

The only solution to this error is to restructure your code so that the offending statement does not appear in the FINALLY clause.

131. 'GOTO <label>' leads into or out of TRY statement

Complete list of compiler error messages

The GOTO statement cannot jump into or out of an exception handling statement.

```
program Produce;

label 1, 2;

begin
  goto 1;
  try
1:
  except
    goto 2;
  end;
2:
end.
```

Both GOTO statements in the above code are incorrect. It is not possible to jump into, or out of, exception handling blocks.

The ideal solution to this problem is to avoid using GOTO statements altogether, however, if that is not possible you will have to perform more detailed analysis of the program to determine the correct course of action.

132. <clause1> clause expected, but <clause2> found

Complete list of compiler error messages

The compiler was, due to the Pascal syntax, expecting to find a clause1 in your program, but instead found clause2.

```
program Produce;

  type
    CharDesc = class
      vch : Char;

property Ch : Char;
    end;
end.
```

The first declaration of a property must specify a read and write clause, and since both are missing on the 'Ch' property, an error will result when compiling. In the case of properties, the original intention might have been to hoist a property defined in a base class to another visibility level - for example, from public to private. In this case, the most probable cause of the error is that the property name was not found in the base class. Make sure that you have spelled the property name correctly and that it is actually contained in one of the parent classes.

```
program Produce;

  type
    CharDesc = class
      vch : Char;

property Ch : Char read vch write vch;
    end;
end.
```

The solution is to ensure that all the proper clauses are specified, where required.

133. Cannot assign to a read-only property

Complete list of compiler error messages

The property to which you are attempting to assign a value did not specify a 'write' clause, thereby causing it to be a read-only property.

```
program Produce;

  type
    Base = class
      s : String;

      property Title : String read s;
    end;

  var
    c : Base;

  procedure DiddleTitle
  begin
    if c.Title = '' then
      c.Title := 'Super Galactic Invaders with Turbo Gungla
Sticks';

      (*perform other work on the c.Title*)
  end;

begin
end.
```

If a property does not specify a 'write' clause, it effectively becomes a read-only property; it is not possible to assign a value to a property which is read-only, thus the compiler outputs an error on the assignment to 'c.Title'.

```
program Solve;

  type
    Base = class
      s : String;

      property Title : String read s;
    end;

  var
    c : Base;

  procedure DiddleTitle
    var title : String;
  begin
    title := c.Title;
    if Title = '' then
      Title := 'Super Galactic Invaders with Turbo Gungla
Sticks';
      (*perform other work on title*)
  end;

begin
end.
```

One solution, if you have source code, is to provide a write clause for the read-only property - of course, this could dramatically alter the semantics of the base class and should not be taken lightly. Another alternative would be to introduce an intermediate variable which would contain the value of the read-only property - it is this second alternative which is shown in the code above.

134. Cannot read a write-only property

Complete list of compiler error messages

The property from which you are attempting to read a value did not specify a 'read' clause, thereby causing it to be a write-only property.

```
program Produce;

  type
    Base = class
      s : String;

      property Password : String write s;
    end;

  var
    c : Base;
    s : String;

begin
  s := c.Password;
end.
```

Since c.Password has not specified a read clause, it is not possible to read its value.

```
program Solve;

  type
    Base = class
      s : String;

      property Password : String read s write s;
    end;

  var
    c : Base;
    s : String;

begin
  s := c.Password;
end.
```

One easy solution to this problem, if you have source code, would be to add a read clause to the write-only property. But, adding a read clause is not always desirable and could lead to holes in a security system - consider, for example, a write-only property called 'Password', as in this example: you certainly wouldn't want to casually allow programs using this class to read the stored password. If a property was created as write-only, there is probably a good reason for it and you should reexamine why you need to read this property.

135. Class already has a default property

Complete list of compiler error messages

You have tried to assign a default property to a class which already has defined a default property.

```
program Produce;

  type
    Base = class
      function GetV(i : Integer) : Char;
      procedure SetV(i : Integer; const x : Char);

      property Data[i : Integer] : Char read GetV write SetV;
default;
      property Access[i : Integer] : Char read GetV write SetV;
default;
    end;

  function Base.GetV(i : Integer) : Char;
  begin GetV := 'A';
  end;

  procedure Base.SetV(i : Integer; const x : Char);
  begin
  end;

begin
end.
```

The Access property in the code above attempts to become the default property of the class, but Data has already been specified as the default. There can be only one default property in a class.

```
program Solve;

  type
    Base = class
      function GetV(i : Integer) : Char;
      procedure SetV(i : Integer; const x : Char);

      property Data[i : Integer] : Char read GetV write SetV;
default;
    end;

  function Base.GetV(i : Integer) : Char;
  begin GetV := 'A';
  end;

  procedure Base.SetV(i : Integer; const x : Char);
  begin
  end;

begin
end.
```

The solution is to remove the incorrect default property specifications from the program source.

136. Default property must be an array property

Complete list of compiler error messages

The default property which you have specified for the class is not an array property. Default properties are required to be array properties.

```
program Produce;

  type
    Base = class
      function GetV : Char;
      procedure SetV(x : Char);

      property Data : Char read GetV write SetV; default;
    end;

  function Base.GetV : Char;
  begin GetV := 'A';
  end;

  procedure Base.SetV(x : Char);
  begin
  end;

begin
end.
```

When specifying a default property, you must make sure that it conforms to the array property syntax. The 'Data' property in the above code specifies a 'Char' type rather than an array.

```
program Solve;

  type
    Base = class
      function GetV(i : Integer) : Char;
      procedure SetV(i : Integer; const x : Char);

      property Data[i : Integer] : Char read GetV write SetV;
default;
    end;

  function Base.GetV(i : Integer) : Char;
  begin GetV := 'A';
  end;

  procedure Base.SetV(i : Integer; const x : Char);
  begin
  end;

begin
end.
```

By changing the specification of the offending property to an array, or by removing the 'default' directive, you can remove this error.

137. TYPEINFO standard function expects a type identifier

Complete list of compiler error messages

You have attempted to obtain type information for an identifier which does not represent a type.

```
program Produce;

  var
    p : Pointer;

  procedure NotType;
  begin
  end;


begin
  p := TypeInfo(NotType);
end.
```

The TypeInfo standard procedure requires a type identifier as it's parameter. In the code above, 'NotType' does not represent a type identifier.

```
program Solve;

  type
    Base = class
    end;

  var
    p : Pointer;

begin
  p := TypeInfo(Base);
end.
```

By ensuring that the parameter used for TypeInfo is a type identifier, you will avoid this error.

138. Type '<name>' has no type info

Complete list of compiler error messages

You have applied the TypeInfo standard procedure to a type identifier which does not have any run-time type information associated with it.

```
program Produce;

  type
    Data = record
    end;

  var
    v : Pointer;

begin
  v := TypeInfo(Data);
end.
```

Record types do not generate type information, so this use of TypeInfo is illegal.

```
program Solve;

  type
    Base = class
    end;

  var
    v : Pointer;

begin
  v := TypeInfo(Base);
end.
```

A class does generate RTTI, so the use of TypeInfo here is perfectly legal.

139. FOR or WHILE loop executes zero times - deleted

Complete list of compiler error messages

The compiler has determined that the specified looping structure will not ever execute, so as an optimization it will remove it. Example:

```
program Produce;
(*$HINTS ON*)

  var
    i : Integer;

begin
  i := 0;
  WHILE FALSE AND (i < 100) DO
    INC(i);
end.
```

The compiler determines that 'FALSE AND (i < 100)' always evaluates to FALSE, and then easily determines that the loop will not be executed.

```
program Solve;
(*$HINTS ON*)

  var
    i : Integer;

begin
  i := 0;
  WHILE i < 100 DO
    INC(i);
end.
```

The solution to this hint is to check the boolean expression used to control while statements is not always FALSE. In the for loops you should make sure that (upper bound - lower bound) >= 1.

You may see this warning if a FOR loop increments its control variable from a value within the range of Longint to a value outside the range of Longint. For example:

```
var I: Cardinal;
begin
  For I := 0 to $FFFFFFFF do
...
```

This results from a limitation in the compiler which you can work around by replacing the FOR loop with a WHILE loop.

140. No definition for abstract method '<name>' allowed

Complete list of compiler error messages

You have declared <name> to be abstract, but the compiler has found a definition for the method in the source file. It is illegal to provide a definition for an abstract declaration.

```
program Produce;

  type
    Base = class
      procedure Foundation; virtual; abstract;
    end;

    procedure Base.Foundation;
    begin
    end;

begin
end.
```

Abstract methods cannot be defined. An error will appear at the point of Base.Foundation when you compile this program.

```
program Solve;

  type
    Base = class
      procedure Foundation; virtual; abstract;
    end;

    Derived = class (Base)
      procedure Foundation; override;
    end;

    procedure Derived.Foundation;
    begin
    end;

begin
end.
```

Two steps are required to solve this error. First, you must remove the definition of the abstract procedure which is declared in the base class. Second, you must extend the base class, declare the abstract procedure as an 'override' in the extension, and then provide a definition for the newly declared procedure.

141. Method '<name>' not found in base class

Complete list of compiler error messages

You have applied the 'override' directive to a method, but the compiler is unable to find a procedure of the same name in the base class.

```
program Produce;

  type
    Base = class
      procedure Title; virtual;
    end;

    Derived = class (Base)
      procedure Titl; override;
    end;

    procedure Base.Title;
    begin
    end;

    procedure Derived.Titl;
    begin
    end;

begin
end.
```

A common cause of this error is a simple typographical error in your source code. Make sure that the name used as the 'override' procedure is spelled the same as it is in the base class. In other situations, the base class will not provide the desired procedure: it is those situations which will require much deeper analysis to determine how to solve the problem.

```
program Solve;

  type
    Base = class
      procedure Title; virtual;
    end;

    Derived = class (Base)
      procedure Title; override;
    end;

    procedure Base.Title;
    begin
    end;

    procedure Derived.Title;
    begin
    end;

begin
end.
```

The solution (in this example) was to correct the spelling of the procedure name in Derived.

142. Invalid message parameter list

Complete list of compiler error messages

A message procedure can take only one, VAR, parameter; it's type is not checked.

```
program Produce;

  type
    Base = class
      procedure Msg1(x : Integer); message 151;
      procedure Msg2(VAR x, y : Integer); message 152;
    end;

  procedure Base.Msg1(x : Integer);
  begin
  end;

  procedure Base.Msg2(VAR x, y : Integer);
  begin
  end;

begin
end.
```

The obvious error in the first case is that the parameter is not VAR. The error in the second case is that more than one parameter is declared.

```
program Solve;

  type
    Base = class
      procedure Msg1(VAR x : Integer); message 151;
      procedure Msg2(VAR y : Integer); message 152;
    end;

  procedure Base.Msg1(VAR x : Integer);
  begin
  end;

  procedure Base.Msg2(VAR y : Integer);
  begin
  end;

begin
end.
```

The solution in both cases was to only specify one, VAR, parameter in the message method declaration.

143. Illegal message method index

Complete list of compiler error messages

You have specified value for your message index which <= 0.

```
program Produce;

  type
    Base = class
      procedure Dynamo(VAR x : Integer); message -151;
    end;

  procedure Base.Dynamo(VAR x : Integer);
  begin
  end;

begin
end.
```

The specification of -151 as the message index is illegal in the above example.

```
program Solve;

  type
    Base = class
      procedure Dynamo(VAR x : Integer); message 151;
    end;

  procedure Base.Dynamo(VAR x : Integer);
  begin
  end;

begin
end.
```

Always make sure that your message index values are >= 1.

144. Duplicate dynamic method index

Complete list of compiler error messages

You have specified an index for a dynamic method which is already used by another dynamic method.

```
program Produce;

  type
    Base = class
      procedure First(VAR x : Integer); message 151;
      procedure Second(VAR x : Integer); message 151;
    end;

  procedure Base.First(VAR x : Integer);
  begin
  end;

  procedure Base.Second(VAR x : Integer);
  begin
  end;

begin
end.
```

The declaration of 'Second' attempts to reuse the same message index which is used by 'First'; this is illegal.

```
program Solve;

  type
    Base = class
      procedure First(VAR x : Integer); message 151;
      procedure Second(VAR x : Integer); message 152; (*change to
unique index*)
    end;

    Derived = class (Base)
      procedure First(VAR x : Integer); override; (*override base
class behavior*)
    end;

  procedure Base.First(VAR x : Integer);
  begin
  end;

  procedure Base.Second(VAR x : Integer);
  begin
  end;

  procedure Derived.First(VAR x : Integer);
  begin
  end;

begin
end.
```

There are two straightforward solutions to this problem. First, if you really do not need to use the same message value, you can change the message number to be unique. Alternatively, you could derive a new class from the base and override the behavior of the message handler declared in the base class. Both options are shown in the above example.

145. Bad file format '<name>'

Complete list of compiler error messages

The compiler state file has become corrupted. It is not possible to reload the previous compiler state.

Delete the corrupt file.

146. Inaccessible value

Complete list of compiler error messages

You have tried to view a value that is not accessible from within the integrated debugger. Certain types of values, such as a 0 length Variant-type string, cannot be viewed within the debugger.

147. Destination cannot be assigned to

Complete list of compiler error messages

The integrated debugger has determined that your assignment is not valid in the current context.

148. Expression has no value

Complete list of compiler error messages

You have attempted to assign the result of an expression, which did not produce a value, to a variable.

149. Destination is inaccessible

Complete list of compiler error messages

The address to which you are attempting to put a value is inaccessible from within the IDE.

150. Re-raising an exception only allowed in exception handler

Complete list of compiler error messages

You have used the syntax of the raise statement which is used to reraise an exception, but the compiler has determined that this reraise has occurred outside of an exception handler block. A limitation of the current exception handling mechanism disallows reraising exceptions from nested exception handlers. for the exception.

```
program Produce;

  procedure RaiseException;
  begin
    raise;            (*case 1*)
    try
      raise;          (*case 2*)
    except
      try
        raise;        (*case 3*)
      except
      end;
      raise;
    end;
  end;


begin
end.
```

There are several reasons why this error might occur. First, you might have specified a raise with no exception constructor outside of an exception handler. Secondly, you might be attempting to reraise an exception in the try block of an exception handler. Thirdly, you might be attempting to reraise the exception in an exception handler nested in another exception handler.

```
program Solve;
  uses SysUtils;

  procedure RaiseException;
  begin
    raise Exception.Create('case 1');
    try
      raise Exception.Create('case 2');
    except
      try
        raise Exception.Create('case 3');
      except
      end;
      raise;
    end;
  end;

begin
end.
```

One solution to this error is to explicitly raise a new exception; this is probably the intention in situations like 'case 1' and 'case 2'. For the situation of 'case 3', you will have to examine your code to determine a suitable workaround which will provide the desired results.

151. Default values must be of ordinal, pointer or small set type

Complete list of compiler error messages

You have declared a property containing a default clause, but the type property type is incompatible with default values.

```
program Produce;

  type
    VisualGauge = class
      pos : Single;
property Position : Single read pos write pos default 0.0;
    end;

begin
end.
```

The program above creates a property and attempts to assign a default value to it, but since the type of the property does not allow default values, an error is output.

```
program Produce;

  type
    VisualGauge = class
      pos : Integer;
property Position : Integer read pos write pos default 0;
    end;

begin
end.
```

When this error is encountered, there are two easy solutions: the first is to remove the default value definition, and the second is to change the type of the property to one which allows a default value. Your program, however, may not be as simple to fix; consider when you have a set property which is too large - it is this case which will require you to carefully examine your program to determine the best solution to this problem.

152. Property '<name>' does not exist in base class

Complete list of compiler error messages

The compiler believes you are attempting to hoist a property to a different visibility level in a derived class, but the specified property does not exist in the base class.

```
program Produce;

  type
    Base = class
    private
      a : Integer;
      property BaseProp : integer read a write a;
    end;

    Derived = class (Base)
      ch : Char;
      property Alpha read ch write ch; (*case 1*)
      property BesaProp; (*case 2*)
    end;

begin
end.
```

There are two basic causes of this error. The first is the specification of a new property without specifying a type; this usually is not supposed to be a movement to a new visibility level. The second is the specification of a property which should exist in the base class, but is not found by the compiler; the most likely cause for this is a simple typo (as in "BesaProp"). In the second form, the compiler will also output errors that a read or write clause was expected. of a proper

```
program Solve;

  type
    Base = class
    private
      a : Integer;
      property BaseProp : integer read a write a;
    end;

    Derived = class (Base)
      ch : Char;
    public
      property Alpha : Char read ch write ch; (*case 1*)
      property BaseProp; (*case 2*)
    end;

begin
end.
```

The solution for the first case is to supply the type of the property. The solution for the second case is to check the spelling of the property name.

153. Dynamic method or message handler not allowed here

Complete list of compiler error messages

Dynamic and message methods cannot be used as accessor functions for properties.

```
program Produce;

  type
    Base = class
      v : Integer;
      procedure SetV(x : Integer); dynamic;
      function GetV : Integer; message;
      property Velocity : Integer read GetV write v;
      property Value : Integer read v write SetV;
    end;

  procedure Base.SetV(x : Integer);
  begin v := x;
  end;

  function Base.GetV : Integer;
  begin GetV := v;
  end;

begin
end.
```

Both 'Velocity' and 'Value' above are in error since they both have illegal accessor functions assigned to them.

```
program Solve;

  type
    Base = class
      v : Integer;
      procedure SetV(x : Integer);
      function GetV : Integer;
      property Velocity : Integer read GetV write v;
      property Value : Integer read v write SetV;
    end;

  procedure Base.SetV(x : Integer);
  begin v := x;
  end;

  function Base.GetV : Integer;
  begin GetV := v;
  end;

begin
end.
```

The solution taken in this is example was to remove the offending compiler directives from the procedure declarations; this may not be the right solution for you. You may have to closely examine the logic of your program to determine how best to provide accessor functions for your properties.

154. Class does not have a default property

Complete list of compiler error messages

You have used a class instance variable in an array expression, but the class type has not declared a default array property.

```
program Produce;

  type
    Base = class
    end;

  var
    b : Base;

  procedure P;
    var ch : Char;
  begin
    ch := b[1];
  end;

begin
end.
```

The example above elicits an error because 'Base' does not declare an array property, and 'b' is not an array itself.

```
program Solve;

  type
    Base = class
      function GetChar(i : Integer) : Char;
      property data[i : Integer] : Char read GetChar; default;
    end;

  var
    b : Base;

  function Base.GetChar(i : Integer) : Char;
  begin GetChar := 'A';
  end;

  procedure P;
    var ch : Char;
  begin
    ch := b[1];
    ch := b.data[1];
  end;

begin
end.
```

When you have declared a default property for a class, you can use the class instance variable in array expression, as if the class instance variable itself were actually an array. Alternatively, you can use the name of the property as the actual array accessor. Note: if you have hints turned

on, you will receive two warnings about the value assigned to 'ch' never being used.

155. Bad argument type in variable type array constructor

Complete list of compiler error messages

You are attempting to construct an array using a type which is not allowed in variable arrays.

```
program Produce;

  type
    Fruit = (apple, orange, pear);
    Data = record
      x : Integer;
      ch : Char;
    end;

  var
    f : Fruit;
    d : Data;

  procedure Examiner(v : array of TVarRec);
  begin
  end;

begin
  Examiner([d]);
  Examiner([f]);
end.
```

Both calls to Examiner will fail because enumerations and records are not supported in array constructors.

```
program Solve;

  var
    i : Integer;
    r : Real;
    v : Variant;

  procedure Examiner(v : array of TVarRec);
  begin
  end;

begin
  i := 0; r := 0; v := 0;
  Examiner([i, r, v]);
end.
```

Many data types, like those in the example above, are allowed in array constructors.

156. Could not load RLINK32.DLL

Complete list of compiler error messages

RLINK32.DLL could not be found. Please ensure that it is on the path.

Contact Inprise if you encounter this error.

157. Wrong or corrupted version of RLINK32.DLL

Complete list of compiler error messages

The internal consistency check performed on the RLINK32.DLL file has failed.

Contact Inprise if you encounter this error.

158. "';' not allowed before 'ELSE'

Complete list of compiler error messages

You have placed a ';' directly before an ELSE in an IF-ELSE statement. The reason for this is that the ';' is treated as a statement separator, not a statement terminator - IF-ELSE is one statement, a ';' cannot appear in the middle (unless you use compound statements).

```
program Produce;

  var
    b : Integer;

begin
  if b = 10 then
    b := 0;
  else
    b := 10;
end.
```

Pascal does not allow a ';' to be placed directly before an ELSE statement. In the code above, an error will be flagged because of this fact.

```
program Solve;

  var
    b : Integer;

begin
  if b = 10 then
    b := 0
  else
    b := 10;

  if b = 10 then begin
    b := 0;
  end
  else begin
    b := 10;
  end;

end.
```

There are two easy solutions to this problem. The first is to remove the offending ';'. The second is to create compound statements for each part of the IF-ELSE. If $HINTS are turned on, you will receive a hint about the value assigned to 'b' is never used. statement.

159. Type '<name>' needs finalization - not allowed in variant record

Complete list of compiler error messages

Certain types are treated specially by the compiler on an internal basis in that they must be correctly finalized to release any resources that they might currently own. Because the compiler cannot determine what type is actually stored in a record's variant section at runtime, it is not possible to guarantee that these special data types are correctly finalized.

```
program Produce;

  type
    Data = record
      case kind:Char of
      'A': (str : String);
    end;

begin
end.
```

String is one of those types which requires special treatment by the compiler to correctly release the resources. As such, it is illegal to have a String in a variant section.

```
program Solve;

  type
    Data = record
      str : String;
    end;

begin
end.
```

One solution to this error is to move all offending declarations out of the variant section. Another solution would be to use pointer types (^String, for example) and manage the memory by yourself.

160. Type '<name>' needs finalization - not allowed in file type

Complete list of compiler error messages

Certain types are treated specially by the compiler on an internal basis in that they must be correctly finalized to release any resources that they might currently own. Because the compiler cannot determine what type is actually stored in a record's variant section at runtime, it is not possible to guarantee that these special data types are correctly finalized.

```
program Produce;

  type
    Data = record
      name : string;
    end;

  var
    inFile : file of Data;

begin
end.
```

String is one of those data types which need finalization, and as such they cannot be stored in a File type.

```
program Solve;

  type
    Data = record
      name : array [1..25] of Char;
    end;

  var
    inFile : file of Data;

begin
end.
```

One simple solution, for the case of String, is to redeclare the type as an array of characters. For other cases which require finalization, it becomes increasingly difficult to maintain a binary file structure with standard Pascal features, such as 'file of'. In these situations, it is probably easier to write specialized file I/O routines.

161. Expression too complicated

Complete list of compiler error messages

The compiler has encounter an expression in your source code that is too complicated for it to handle.

Reduce the complexity of your expression by introducing some temporary variables.

162. Element 0 inaccessible - use 'Length' or 'SetLength'

Complete list of compiler error messages

The Delphi32 String type does not store the length of the string in element 0. The old method of changing, or getting, the length of a string by accessing element 0 does not work with long strings.

```
program Produce;

  var
    str : String;
    len : Integer;

begin
  str := 'Kojo no tsuki';
  len := str[0];
end.
```

Here the program is attempting to get the length of the string by directly accessing the first element. This is not legal.

```
program Solve;

  var
    str : String;
    len : Integer;

begin
  str := 'Kojo no tsuki';
  len := Length(str);
end.
```

You can use the SetLength and Length standard procedures to provide the same functionality as directly accessing the first element of the string. If hints are turned on, you will receive a warning about the value of 'len' not being used.

163. System unit out of date or corrupted: missing '<name>'

Complete list of compiler error messages

The compiler is looking for a special function which resides in System.dcu but could not find it. Your System unit is either corrupted or obsolete.

Make sure there are no conflicts in your library search path which can point to another System.dcu. Try reinstalling System.dcu. If neither of these solutions work, contact Inprise Developer Support.

164. Type not allowed in OLE Automation call

Complete list of compiler error messages

If a data type cannot be converted by the compiler into a Variant, then it is not allowed in an OLE automation call.

```
program Produce;

  type
    Base = class
      x : Integer;
    end;

  var
    B : Base;
    V : Variant;

begin
  V.Dispatch(B);
end.
```

A class cannot be converted into a Variant type, so it is not allowed in an OLE call.

```
program Solve;


  type
    Base = class
      x : Integer;
    end;

  var
    B : Base;
    V : Variant;

begin
  V.Dispatch(B.i);
end.
```

The only solution to this problem is to manually convert these data types to Variants or to only use data types that can automatically be converted into a Variant.

165. RLINK32 error

Complete list of compiler error messages

RLINK32 has encountered an error. Contact Inprise Developer Support if you encounter this error.

166. RLINK32 error

Complete list of compiler error messages

RLINK32 has encountered an error. Contact Inprise Developer Support if you encounter this error.

167. Too many conditional symbols

Complete list of compiler error messages

You have exceeded the memory allocated to conditional symbols defined on the command line (including configuration files). There are 256 bytes allocated for all the conditional symbols. Each conditional symbol requires 1 extra byte when stored in conditional symbol area.

The only solution is to reduce the number of conditional compilation symbols contained on the command line (or in configuration files).

168. Method '<name>' hides virtual method of base type '<name>'

Complete list of compiler error messages

You have declared a method which has the same name as a virtual method in the base class. Your new method is not a virtual method; it will hide access to the base's method of the same name.

```
program Produce;

  type
    Base = class
      procedure VirtuMethod; virtual;
      procedure VirtuMethod2; virtual;
    end;

    Derived = class (Base)
      procedure VirtuMethod;
      procedure VirtuMethod2;
    end;

  procedure Base.VirtuMethod;
  begin
  end;

  procedure Base.VirtuMethod2;
  begin
  end;

  procedure Derived.VirtuMethod;
  begin
  end;

  procedure Derived.VirtuMethod2;
  begin
  end;

begin
end.
```

Both methods declared in the definition of Derived will hide the virtual functions of the same name declared in the base class.

```
program Solve;

  type
    Base = class
      procedure VirtuMethod; virtual;
      procedure VirtuMethod2; virtual;
    end;

    Derived = class (Base)
      procedure VirtuMethod; override;
      procedure Virtu2Method;
    end;

  procedure Base.VirtuMethod;
  begin
  end;

  procedure Base.VirtuMethod2;
  begin
  end;

  procedure Derived.VirtuMethod;
  begin
  end;

  procedure Derived.Virtu2Method;
  begin
  end;

begin
end.
```

There are three alternatives to take when solving this warning.

First, you could specify override to make the derived class' procedure also virtual, and thus allowing inherited calls to still reference the original procedure.

Secondly, you could change the name of the procedure as it is declared in the derived class. Both methods are exhibited in this example.

Finally, you could add the `reintroduce` directive to the procedure declaration to cause the warning to be silenced for that particular method.

Virtual Methods Static Methods Overriding Methods

169. Variable '<name>' is declared but never used in '<name>'

Complete list of compiler error messages

You have declared a variable in a procedure, but you never actually use it. -H

```
program Produce;
(*$HINTS ON*)

  procedure Local;
    var i : Integer;
  begin
  end;

begin
end.
```

```
program Solve;

(*$HINTS ON*)

  procedure Local;
  begin
  end;

begin
end.
```

One simple solution is to remove any unused variable from your procedures. However, unused variables can also indicate an error in the implementation of your algorithm.

170. Compile terminated by user

Complete list of compiler error messages

You pressed Ctrl-Break during a compile.

171. Unnamed arguments must precede named arguments in OLE Automation call

Complete list of compiler error messages

You have attempted to follow named OLE Automation arguments with unnamed arguments.

```
program Produce;

  var
    ole : variant;

begin ole.dispatch(filename:='FrogEggs', 'Tapioca');
end.
```

The named argument, 'filename' must follow the unnamed argument in this OLE dispatch.

```
program Solve;

  var
    ole : variant;

begin ole.dispatch('Tapioca', filename:='FrogEggs');
end.
```

This solution, reversing the parameters, is the most straightforward but it may not be appropriate for your situation. Another alternative would be to provide the unnamed parameter with a name.

172. Abstract methods must be virtual or dynamic

Complete list of compiler error messages

When declaring an abstract method in a base class, it must either be of regular virtual or dynamic virtual type.

```
program Produce;

  type
    Base = class
      procedure DaliVision; abstract;
      procedure TellyVision; abstract;
    end;

begin
end.
```

The declaration above is in error because abstract methods must either be virtual or dynamic.

```
program Solve;

  type
    Base = class
      procedure DaliVision; virtual; abstract;
      procedure TellyVision; dynamic; abstract;
    end;

begin
end.
```

It is possible to remove this error by either specifying 'virtual' or 'dynamic', whichever is most appropriate for your application.

173. Case label outside of range of case expression

Complete list of compiler error messages

You have provided a label inside a case statement which cannot be produced by the case statement control variable. -W

```
program Produce;
(*$WARNINGS ON*)

  type
    CompassPoints = (n, e, s, w, ne, se, sw, nw);
    FourPoints = n..w;

  var
    TatesCompass : FourPoints;

begin

   TatesCompass := e;
   case TatesCompass OF
   n:    Writeln('North');
   e:    Writeln('East');
   s:    Writeln('West');
   w:    Writeln('South');
   ne:   Writeln('Northeast');
   se:   Writeln('Southeast');
   sw:   Writeln('Southwest');
   nw:   Writeln('Northwest');
   end;
end.
```

It is not possible for a TatesCompass to hold all the values of the CompassPoints, and so several of the case labels will elicit errors.

```
program Solve;
(*$WARNINGS ON*)

  type
    CompassPoints = (n, e, s, w, ne, se, sw, nw);
    FourPoints = n..w;

  var
    TatesCompass : CompassPoints;

begin

   TatesCompass := e;
   case TatesCompass OF
   n:    Writeln('North');
   e:    Writeln('East');
   s:    Writeln('West');
   w:    Writeln('South');
   ne:   Writeln('Northeast');
   se:   Writeln('Southeast');
   sw:   Writeln('Southwest');
   nw:   Writeln('Northwest');
   end;
end.
```

After examining your code to determine what the intention was, there are two alternatives. The first is to change the type of the case statement's control variable so that it can produce all the case labels. The second alternative would be to remove any case labels that cannot be produced by the control variable. The first alternative is shown in this example.

174. Field or method identifier expected

Complete list of compiler error messages

You have specified an identifier for a read or write clause to a property which is not a field or method.

```
program Produce;

  var
    r : string;

  type
    Base = class
      t : string;
      property Title : string read Title write Title;
      property Caption : string read r write r;

    end;

begin
end.
```

The two properties in this code both cause errors. The first causes an error because it is not possible to specify the property itself as the read & write methods. The second causes an error because 'r' is not a member of the Base class.

```
program Solve;

  type
    Base = class
      t : string;
      property Title : string read t write t;
    end;

begin
end.
```

To solve this error, make sure that all read & write clauses for properties specify a valid field or method identifier that is a member of the class which owns the property.

175. Constructing instance of '<name>' containing abstract methods

Complete list of compiler error messages

The code you are compiling is constructing instances of classes which contain abstract methods.

```
program Produce;
(*$WARNINGS ON*)
(*$HINTS ON*)

  type
    Base = class
      procedure Abstraction; virtual; abstract;
    end;

  var
    b : Base;

begin
  b := Base.Create;
end.
```

An abstract procedure does not exist, so it becomes dangerous to create instances of a class which contains abstract procedures. In this case, the creation of 'b' is the cause of the warning. Any invocation of 'Abstraction' through the instance of 'b' created here would cause a runtime error. A hint will be issued that the value assigned to 'b' is never used.

```
program Solve;
(*$WARNINGS ON*)
(*$HINTS ON*)

  type
    Base = class
      procedure Abstraction; virtual;
    end;

  var
    b : Base;

  procedure Base.Abstraction;
  begin
  end;

begin
  b := Base.Create;
end.
```

One solution to this problem is to remove the abstract directive from the procedure declaration, as is shown here. Another method of approaching the problem would be to derive a class from Base and then provide a concrete version of Abstraction. A hint will be issued that the value assigned to 'b' is never used.

176. Field definition not allowed after methods or properties

Complete list of compiler error messages

You have attempted to add more fields to a class after the first method or property declaration has been encountered. You must place all field definitions before methods and properties.

```
program Produce;

  type
    Base = class
      procedure FirstMethod;
      a : Integer;
    end;


  procedure Base.FirstMethod;
  begin
  end;

begin
end.
```

The declaration of 'a' after 'FirstMethod' will cause an error.

```
program Solve;

  type
    Base = class
      a : Integer;
      procedure FirstMethod;
    end;


  procedure Base.FirstMethod;
  begin
  end;

begin
end.
```

To solve this error, it is normally sufficient to move all field definitions before the first field or property declaration.

177. Cannot override a static method

Complete list of compiler error messages

You have tried, in a derived class, to override a base method which was not declared as one of the virtual types.

```
program Produce;

  type
    Base = class
      procedure StaticMethod;
    end;

    Derived = class (Base)
      procedure StaticMethod; override;
    end;

    procedure Base.StaticMethod;
    begin
    end;

    procedure Derived.StaticMethod;
    begin
    end;

begin
end.
```

The example above elicits an error because Base.StaticMethod is not declared to be a virtual method, and as such it is not possible to override its declaration.

```
program Solve;

  type
    Base = class
      procedure StaticMethod;
    end;

    Derived = class (Base)
      procedure StaticMethod;
    end;

    procedure Base.StaticMethod;
    begin
    end;

    procedure Derived.StaticMethod;
    begin
    end;

begin
end.
```

The only way to remove this error from your program, when you don't have the source for the base classes, is to remove the 'override' specification from the declaration of the derived method.

If you have source to the base classes, you could, with careful consideration, change the base's method to be declared as one of the virtual types - but be aware that this change can have a drastic affect on your programs.

178. Variable '<name>' inaccessible here due to optimization

Complete list of compiler error messages

The evaluator or watch statement is attempting to retrieve the value of <name>, but the compiler was able to determine that the variables actual lifetime ended prior to this inspection point. This error will often occur if the compiler determines a local variable is assigned a value that is not used beyond a specific point in the program's control flow.

```
Create a new application.
Place a button on the form.
Double click the button to be taken to the 'click' method.
Add a global variable, 'c', of type Integer to the implementation
section.

The click method should read as:

  procedure TForm1.Button1Click(Sender: TObject);
    var a, b : integer;
  begin
    a := 10;
    b := 20;
    c := b;
    a := c;
  end;

Set a breakpoint on the assignment to 'c'.
Compile and run the application.
Press the button.
After the breakpoint is reached, open the evaluator (Run|
Evaluate/Watch).
Evaluate 'a'.
```

The compiler realizes that the first assignment to 'a' is dead, since the value is never used. As such, it defers even using 'a' until the second assignment occurs - up until the point where 'c' is assigned to 'a', the variable 'a' is considered to be dead and cannot be used by the evaluator.

The only solution is to only attempt to view variables which are known to have live values.

179. Necessary library helper function was eliminated by linker

Complete list of compiler error messages

The integrated debugger is attempting to use some of the compiler helper functions to perform the requested evaluate. The linker, on the other hand, determined that the helper function was not actually used by the program and it did not link it into the program.

```
Create a new application.
Place a button on the form.
Double click the button to be taken to the 'click' method.
Add a global variable, 'v', of type String to the interface
section.
Add a global variable, 'p', of type PChar to the interface
section.

The click method should read as:

  procedure TForm1.Button1Click(Sender: TObject);
  begin
       v := 'Initialized';
       p := NIL;
       v := 'Abid';
  end;

Set a breakpoint on the second assignment to 'v'.
Compile and run the application.
Press the button.
After the breakpoint is reached, open the evaluator (Run|
Evaluate/Watch).
Evaluate 'v'.
Move the cursor to the 'New Value' box.
Type in 'p'.
Choose Modify.
```

The compiler uses a special function to copy a PChar to a String. In order to reduce the size of the produced executable, if that special function is not used by the program, it is not linked in. In this case, there is no assignment of a PChar to a String, so it is eliminated by the linker.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     v := 'Initialized';
     p := NIL;
     v := 'Abid';
     v := p;
end;
```

Adding the extra assignment of a PChar to a String will ensure that the linker includes the desired procedure in the program. Encountering this error during a debugging session is an indicator that you are using some language/environment functionality that was not needed in the original program.

180. Missing or invalid conditional symbol in '$<symbol>' directive

Complete list of compiler error messages

The $IFDEF, $IFNDEF, $DEFINE and $UNDEF directives require that a symbol follow them.

```
program Produce;

(*$IFDEF*)
(*$ENDIF*)

begin
end.
```

The $IFDEF conditional directive is incorrectly specified here and will result in an error.

```
program Solve;

(*$IFDEF WIN32*)
(*$ENDIF*)

begin
end.
```

The solution to the problem is to ensure that a symbol to test follows the appropriate directives.

181. '<name>' not previously declared as a PROPERTY

Complete list of compiler error messages

You have attempted to hoist a property to a different visibility level by redeclaration, but <name> in the base class was not declared as a property. -W

```
program Produce;
(*$WARNINGS ON*)

  type
    Base = class
    protected
      Caption : String;
      Title : String;
      property TitleProp : string read Title write Title;
    end;

    Derived = class (Base)
    public
      property Title read Caption write Caption;
    end;

begin
end.
```

The intent of the redeclaration of 'Derived.Title' is to change the field which is used to read and write the property 'Title' as well as hoist it to 'public' visibility. Unfortunately, the programmer really meant to use 'TitleProp', not 'Title'.

```
program Solve;
(*$WARNINGS ON*)

  type
    Base = class
    protected
      Caption : String;
      Title : String;
      property TitleProp : string read Title write Title;
    end;

    Derived = class (Base)
    public
      property TitleProp read Caption write Caption;
      property Title : string read Caption write Caption;
    end;

begin
end.
```

There are a couple ways of approaching this error. The first, and probably the most commonly taken, is to specify the real property which is to be redeclared. The second, which can be seen in the redeclaration of 'Title' addresses the problem by explicitly creating a new property, with the same name as a field in the base class. This new property will hide the base field, which will no longer be accessible without a typecast. (Note: If you have warnings turned on, the redeclaration of 'Title' will issue a warning notifying you that the redeclaration will hide the base class' member.)

182. Field definition not allowed in OLE automation section

Complete list of compiler error messages

You have tried to place a field definition in an OLE automation section of a class declaration. Only properties and methods may be declared in an 'automated' section.

```
program Produce;

  type
    Base = class
    automated
      i : Integer;
    end;

begin
end.
```

The declaration of 'i' in this class will cause the compile error.

```
program Solve;

  type
    Base = class
      i : Integer;
    automated
    end;

begin
end.
```

Moving the declaration of 'i' out of the automated section will vanquish the error.

183. Illegal type in OLE automation section: '<typename>'

Complete list of compiler error messages

<typename> is not an allowed type in an OLE automation section. Only a small subset of all the valid Pascal types are allowed in automation sections.

```
program Produce;

  type
    Base = class
      function GetC : Char;
      procedure SetC(c : Char);
    automated
      property Ch : Char read GetC write SetC dispid 151;
    end;

  procedure Base.SetC(c : Char);
  begin
  end;

  function Base.GetC : Char;
  begin GetC := '!';
  end;

begin
end.
```

Since the character type is not one allowed in the 'automated' section, the declaration of 'Ch' will produce an error when compiled.

```
program Solve;

  type
    Base = class
      function GetC : String;
      procedure SetC(c : String);
    automated
      property Ch : String read GetC write SetC dispid 151;
    end;

  procedure Base.SetC(c : String);
  begin
  end;

  function Base.GetC : String;
  begin GetC := '!';
  end;

begin
end.
```

There are two solutions to this problem. The first is to move the offending declaration out of the 'automated' section. The second is to change the offending type to one that is allowed in 'automated' sections.

184. String constant truncated to fit STRING[<number>]

Complete list of compiler error messages

A string constant is being assigned to a variable which is not large enough to contain the entire string. The compiler is alerting you to the fact that it is truncating the literal to fit into the variable. -W

```
program Produce;
(*$WARNINGS ON*)

  const
    Title = 'Super Galactic Invaders with Turbo Gungla Sticks';
    Subtitle = 'Copyright (c) 1968 by Frank Borland';

  type
    TitleString = String[25];
    SubtitleString = String[18];


  var
    ProgramTitle : TitleString;
    ProgramSubtitle : SubtitleString;

begin
  ProgramTitle := Title;
  ProgramSubtitle := Subtitle;
end.
```

The two string constants are assigned to variables which are too short to contain the entire string. The compiler will truncate the strings and perform the assignment.

```
program Solve;
(*$WARNINGS ON*)

  const
    Title = 'Super Galactic Invaders with Turbo Gungla Sticks';
    Subtitle = 'Copyright (c) 1968';

  type
    TitleString = String[55];
    SubtitleString = String[18];


  var
    ProgramTitle : TitleString;
    ProgramSubtitle : SubtitleString;

begin
  ProgramTitle := Title;
  ProgramSubtitle := Subtitle;
end.
```

There are two solutions to this problem, both of which are demonstrated in this example. The first solution is to increase the size of the variable to hold the string. The second is to reduce the size of the string to fit in the declared size of the variable.

185. Constructors and destructors not allowed in OLE automation section

Complete list of compiler error messages

You have incorrectly tried to put a constructor or destructor into the 'automated' section of a class declaration.

```
program Produce;

  type
    Base = class
    automated
      constructor HardHatBob;
      destructor  DemolitionBob;
    end;

  constructor Base.HardHatBob;
  begin
  end;

  destructor Base.DemolitionBob;
  begin
  end;

begin
end.
```

It is not possible to declare a class constructor or destruction in an OLE automation section. The constructor and destructor declarations in the above code will both elicit this error.

```
program Solve;

  type
    Base = class
      constructor HardHatBob;
      destructor  DemolitionBob;
    end;

  constructor Base.HardHatBob;
  begin
  end;

  destructor Base.DemolitionBob;
  begin
  end;

begin
end.
```

The only solution to this error is to move your declarations out of the automated section, as has been done in this example.

186. Dynamic methods and message handlers not allowed in OLE automation section

Complete list of compiler error messages

You have incorrectly put a dynamic or message method into an 'automated' section of a class declaration.

```
program Produce;

  type
    Base = class
    automated
      procedure DynaMethod; dynamic;
      procedure MessageMethod(VAR msg : Integer); message 151;
    end;

    procedure Base.DynaMethod;
    begin
    end;

    procedure Base.MessageMethod;
    begin
    end;

begin
end.
```

It is not possible to have a dynamic or message method declaration in an OLE automation section of a class. As such, the two method declarations in the above program both produce errors.

```
program Solve;

  type
    Base = class
      procedure DynaMethod; dynamic;
      procedure MessageMethod(VAR msg : Integer); message 151;
    end;

    procedure Base.DynaMethod;
    begin
    end;

    procedure Base.MessageMethod;
    begin
    end;

begin
end.
```

There are several ways to remove this error from your program. First, you could move any declaration which produces this error out of the automated section, as has been done in this example. Alternatively, you could remove the dynamic or message attributes of the method; of course, removing these attributes will not provide you with the desired behavior, but it will remove the error.

187. Only register calling convention allowed in OLE automation section

Complete list of compiler error messages

You have specified an illegal calling convention on a method appearing in an 'automated' section of a class declaration.

```
program Produce;

  type
    Base = class
    automated
      procedure Method; cdecl;
    end;

  procedure Base.Method; cdecl;
  begin
  end;

begin
end.
```

The language specification disallows all calling conventions except 'register' in an OLE automation section. The offending statement is 'cdecl' in the above code.

```
program Solve;

  type
    Base = class
    automated
      procedure Method; register;
      procedure Method2;
    end;

  procedure Base.Method; register;
  begin
  end;

  procedure Base.Method2;
  begin
  end;

begin
end.
```

There are three solutions to this error. The first is to specify no calling convention on methods declared in an auto section. The second is to specify only the register calling convention. The third is to move the offending declaration out of the automation section.

188. Dispid '<number>' already used by '<name>'

Complete list of compiler error messages

An attempt to use a dispid which is already assigned to another member of this class.

```
program Produce;

  type
    Base = class
      v : Integer;
      procedure setV(x : Integer);
      function getV : Integer;
    automated
      property Value : Integer read getV write setV dispid 151;
      property SecondValue : Integer read getV write setV dispid
151;
    end;

  procedure Base.setV(x : Integer);
  begin v := x;
  end;

  function Base.getV : Integer;
  begin getV := v;
  end;

begin
end.
```

Each automated property's dispid must be unique, thus SecondValue is in error.

```
program Solve;

  type
    Base = class
      v : Integer;
      procedure setV(x : Integer);
      function getV : Integer;
    automated
      property Value : Integer read getV write setV dispid 151;
      property SecondValue : Integer read getV write setV dispid
152;
    end;

  procedure Base.setV(x : Integer);
  begin v := x;
  end;

  function Base.getV : Integer;
  begin getV := v;
  end;

begin
end.
```

Giving a unique dispid to SecondValue will remove the error.

189. Redeclaration of property not allowed in OLE automation section

Complete list of compiler error messages

It is not allowed to move the visibility of a property into an automated section.

```
program Produce;

  type
    Base = class
      v : Integer;
      s : String;
    protected
      property Name : String read s write s;
      property Value : Integer read v write v;
    end;

    Derived = class (Base)
    public
      property Name; (* Move Name to a public visibility by
redeclaration *)
    automated
      property Value;
    end;

begin
end.
```

In the above example, Name is moved from a private visibility in Base to public visibility in Derived by redeclaration. The same idea is attempted on Value, but an error results.

```
program Solve;

  type
    Base = class
      v : Integer;
      s : String;
    protected
      property Name : String read s write s;
      property Value : Integer read v write v;
    end;

    Derived = class (Base)
    public
      property Name; (* Move Name to a public visibility by
redeclaration *)
      property Value;
    automated
    end;

begin
end.
```

It is not possible to change the visibility of a property to an automated section, therefore the solution to this problem is to not redeclare properties of base classes in automated sections.

190. '<clause>' clause not allowed in OLE automation section

Complete list of compiler error messages

INDEX, STORED, DEFAULT and NODEFAULT are not allowed in OLE automation sections.

```
program Produce;

  type
     Base = class
       v : integer;
       procedure setV(x : integer);
       function getV : integer;
       automated
       property Value : integer read getV write setV nodefault;
     end;

  procedure Base.setV(x : integer);
  begin v := x;
  end;

  function Base.getV : integer;
  begin getV := v;
  end;

begin
end.
```

Including a NODEFAULT clause on an automated property is not allowed.

```
program Solve;

  type
     Base = class
       v : integer;
       procedure setV(x : integer);
       function getV : integer;
       automated
       property Value : integer read getV write setV;
     end;

  procedure Base.setV(x : integer);
  begin v := x;
  end;

  function Base.getV : integer;
  begin getV := v;
  end;

begin
end.
```

Removing the offending clause will cause the error to go away. Alternatively, moving the property out of the automated section will also make the error go away.

191. Dispid clause only allowed in OLE automation section

Complete list of compiler error messages

A dispid has been given to a property which is not in an automated section.

```
program Produce;

  type
     Base = class
       v : integer;
       procedure setV(x : integer);
       function getV : integer;
       property Value : integer read getV write setV dispid 151;
     end;

  procedure Base.setV(x : integer);
  begin v := x;
  end;

  function Base.getV : integer;
  begin getV := v;
  end;

begin
end.
```

This program attempts to set the dispid for an OLE automation object, but the property has not been declared in an automated section.

```
program Solve;

  type
     Base = class
       v : integer;
       procedure setV(x : integer);
       function getV : integer;
     automated
       property Value : integer read getV write setV dispid 151;
     end;

  procedure Base.setV(x : integer);
  begin v := x;
  end;

  function Base.getV : integer;
  begin getV := v;
  end;

begin
end.
```

To solve the error, you can either remove the dispid clause from the property declaration, or move the property declaration into an automated section.

192. Type '<name>' must be a class to have OLE automation

Complete list of compiler error messages

Old-style Objects cannot have an automated section.

```
program Produce;

  type
    OldObject = object
    automated
    end;

begin
end.
```

It is not possible to have an automated section in an old-style object, thus an error will result from this example.

```
program Solve;

  type
    NewClass = class
    automated
    end;

begin
end.
```

Changing the type from 'object' to 'class', or removing the automated section will remove the error.

193. Type '<name>' must be a class to have a PUBLISHED section

Complete list of compiler error messages

Old-style Objects cannot have a published section. -$M+

```
(*$TYPEINFO ON*)
program Produce;

  type
    OldObject = object
    published
    end;

begin
end.
```

It is not possible to have a published section in an old-style object, thus an error will result from this example.

```
(*$TYPEINFO ON*)
program Solve;

  type
    NewClass = class
    published
    end;

begin
end.
```

Changing the type from 'object' to 'class', or removing the published section will remove the error.

194. Redeclaration of '<name>' hides a member in the base class

Complete list of compiler error messages

A property has been created in a class with the same name of a variable contained in one of the base classes. One possible, and not altogether apparent, reason for getting this error is that a new version of the base class hierarchy has been installed and it contains new member variables which have names identical to your properties' names. -W

```
(*$WARNINGS ON*)
program Produce;

  type
    Base = class
      v : integer;
    end;

    Derived = class (Base)
      ch : char;
      property v : char read ch write ch;
    end;

begin
end.
```

Derived.v overrides, and thus hides, Base.v; it will not be possible to access Base.v in any variable of type Derived without a typecast.

```
(*$WARNINGS ON*)
program Solve;
  type
    Base = class
      v : integer;
    end;

    Derived = class (Base)
      ch : char;
      property chV : char read ch write ch;
    end;

begin
end.
```

By changing the name of the property in the derived class, the error is alleviated.

195. Overriding automated virtual method '<name>' cannot specify a dispid

Complete list of compiler error messages

The dispid declared for the original virtual automated procedure declaration must be used by all overriding procedures in derived classes.

```
program Produce;

  type
    Base = class
    automated
      procedure Automatic; virtual; dispid 151;
    end;


    Derived = class (Base)
    automated
      procedure Automatic; override; dispid 152;
    end;

  procedure Base.Automatic;
  begin
  end;

  procedure Derived.Automatic;
  begin
  end;

begin
end.
```

The overriding declaration of Base.Automatic, in Derived (Derived.Automatic) erroneously attempts to define another dispid for the procedure.

```
program Solve;

  type
    Base = class
    automated
      procedure Automatic; virtual; dispid 151;
    end;


    Derived = class (Base)
    automated
      procedure Automatic; override;
    end;

  procedure Base.Automatic;
  begin
  end;

  procedure Derived.Automatic;
  begin
  end;

begin
end.
```

By removing the offending dispid clause, the program will now compile.

196. Published Real48 property '<name>' must be Single, Real, Double or Extended

Complete list of compiler error messages

You have attempted to publish a property of type Real, which is not allowed. Published floating point properties must be Single, Double or Extended.

```
program Produce;
  type
    Base = class
      R : Real48;
    published
      property RVal : Real read R write R;
    end;
end.
```

The published Real48 property in the program above must be either removed, moved to an unpublished section or changed into an acceptable type.

```
program Produce;
  type
    Base = class
      R : Single;
    published
      property RVal : Single read R write R;
    end;
end.
```

This solution changed the property into a real type that will actually produce run-time type information.

197. Size of published set '<name>' is >32 bits

Complete list of compiler error messages

The compiler does not allow sets greater than 32 bits to be contained in a published section. The size, in bytes, of a set can be calculated by High(setname) div 8 - Low(setname) div 8 + 1. -$M+

```
(*$TYPEINFO ON*)
program Produce;
  type
    CharSet = set of Char;
    NamePlate = class
      Characters : CharSet;
    published
      property TooBig : CharSet read Characters write
Characters ;
    end;

begin
end.
```

```
(*$TYPEINFO ON*)
program Solve;
  type
    CharSet = set of 'A'..'Z';
    NamePlate = class
      Characters : CharSet;
    published
      property TooBig : CharSet read Characters write
Characters ;
    end;

begin
end.
```

198. Published property '<name>' cannot be of type <type>

Complete list of compiler error messages

Published properties must be an ordinal type, Single, Double, Extended, Comp, a string type, a set type which fits in 32 bits, or a method pointer type. When any other property type is encountered in a published section, the compiler will remove the published attribute -$M+

```
(*$TYPEINFO ON*)
program Produce;

  type
    TitleArr = array [0..24] of char;
    NamePlate = class
    private
      titleStr : TitleArr;
    published
      property Title : TitleArr read titleStr write titleStr;
    end;

begin
end.
```

An error is induced because an array is not one of the data types which can be published.

```
(*$TYPEINFO ON*)
program Solve;

  type
    TitleArr = integer;
    NamePlate = class
      titleStr : TitleArr;
    published
      property Title : TitleArr read titleStr write titleStr;
    end;

begin
end.
```

Moving the property declaration out of the published section will avoid this error. Another alternative, as in this example, is to change the type of the property to be something that can actually be published.

199. Thread local variables cannot be local to a function

Complete list of compiler error messages

Thread local variables must be declared at a global scope.

```
program Produce;

  procedure NoTLS;
    threadvar
      x : Integer;
  begin
  end;

begin
end.
```

A thread variable cannot be declared local to a procedure.

```
program Solve;

  threadvar
    x : Integer;

  procedure YesTLS;
    var
      localX : Integer;
  begin
  end;

begin
end.
```

There are two simple alternatives for avoiding this error. First, the threadvar section can be moved to a local scope. Secondly, the threadvar in the procedure could be changed into a normal var section. Note that if compiler hints are turned on, a hint about localX being declared but not used will be emitted.

200. Thread local variables cannot be ABSOLUTE

Complete list of compiler error messages

A thread local variable cannot refer to another variable, nor can it reference an absolute memory address.

```
program Produce;

  threadvar
     secretNum : integer absolute $151;

begin
end.
```

The absolute directive is not allowed in a threadvar declaration section.

```
program Solve;

  threadvar
    secretNum : integer;

  var
    sNum : integer absolute $151;

begin
end.
```

There are two easy ways to solve a problem of this nature. The first is to remove the absolute directive from the threadvar section. The second would be to move the absolute variable to a normal var declaration section.

201. EXPORTS allowed only at global scope

Complete list of compiler error messages

An EXPORTS clause has been encountered in the program source at a non-global scope.

```
program Produce;

  procedure ExportedProcedure;
  exports ExportedProcedure;
  begin
  end;

begin
end.
```

It is not allowed to have an EXPORTS clause anywhere but a global scope.

```
program Solve;

  procedure ExportedProcedure;
  begin
  end;

exports ExportedProcedure;
begin
end.
```

The solution is to ensure that your EXPORTS clause is at a global scope and textually follows all procedures named in the clause. As a general rule, EXPORTS clauses are best placed right before the source file's initialization code.

202. Constants cannot be used as open array arguments

Complete list of compiler error messages

Open array arguments must be supplied with an actual array variable, a constructed array or a single variable of the argument's element type.

```
program Produce;

  procedure TakesArray(s : array of String);
  begin
  end;


begin TakesArray('Hello Error');
end.
```

The error is caused in this example because a string literal is being supplied when an array is expected. It is not possible to implicitly construct an array from a constant.

```
program Solve;

  procedure TakesArray(s : array of String);
  begin
  end;


begin TakesArray(['Hello Error']);
end.
```

The solution avoids the error because the array is explicitly constructed.

203. Slice standard function only allowed as open array argument

Complete list of compiler error messages

An attempt has been made to pass an array slice to a fixed size array. Array slices can only be sent to open array parameters. none

```
program Produce;

  type
    IntegerArray = array [1..10] OF Integer;

  var
    SliceMe : array [1..200] OF Integer;

  procedure TakesArray(x : IntegerArray);
  begin
  end;

begin TakesArray(SLICE(SliceMe, 5));
end.
```

In the above example, the error is produced because TakesArray expects a fixed size array.

```
program Solve;

  type
    IntegerArray = array [1..10] OF Integer;

  var
    SliceMe : array [1..200] OF Integer;

  procedure TakesArray(x : array of Integer);
  begin
  end;

begin TakesArray(SLICE(SliceMe, 5));
end.
```

In the above example, the error is not produced because TakesArray takes an open array as the parameter.

204. Cannot initialize thread local variables

Complete list of compiler error messages

The compiler does not allow initialization of thread local variables.

```
program Produce;

  threadvar
    tls : Integer = 151;

begin
end.
```

The declaration and initialization of 'tls' above is not allowed.

```
program Solve;

  threadvar
    tls : Integer;

begin tls := 151;
end.
```

You can declare thread local storage as normal, and then initialize it in the initialization section of your source file.

205. Cannot initialize local variables

Complete list of compiler error messages

The compiler disallows the use of initialized local variables.

```
program Produce;

  var
    j : Integer;

  procedure Show;
    var i : Integer = 151;
  begin
  end;

begin
end.
```

The declaration and initialization of 'i' in procedure 'Show' is illegal.

```
program Solve;

  var
    j : Integer;

  procedure Show;
    var i : Integer;
  begin
    i := 151;
  end;

begin
  j := 0;
end.
```

You can use a programmatic style to set all variables to known values.

206. Cannot initialize multiple variables

Complete list of compiler error messages

Variable initialization can only occur when variables are declared individually.

```
program Produce;

  var
    i, j : Integer = 151, 152;

begin
end.
```

The compiler will disallow the declaration and initialization of more than one variable at a time.

```
program Solve;

  var
    i : Integer = 151;
    j : Integer = 152;

begin
end.
```

Simple declare each variable by itself to allow initialization.

207. Constant object cannot be passed as var parameter

Complete list of compiler error messages

As variable parameters are intended to be modified by the called procedure or function, you can not pass a constant object to a variable parameter.

If your intention is to pass a large data structure efficiently, and the called function should not modify it, you can use a const parameter instead.

```
program Produce;
(*$APPTYPE CONSOLE*)

function Max(var A: array of Integer): Integer;
var I: Integer;
begin
  Result := Low(Integer);
  for I := 0 to High(A) do
    if Result < A[I] then
       Result := A[I];
end;

begin
  Writeln( Max([1,2,3]) );    (*<-- Error message here*)
end.
```

In the example, function has a variable parameter, but we are passing a constant to it.

```
program Solve;
(*$APPTYPE CONSOLE*)

function Max(const A: array of Integer): Integer;
var I: Integer;
begin
  Result := Low(Integer);
  for I := 0 to High(A) do
    if Result < A[I] then
      Result := A[I];
end;

begin
  Writeln( Max([1,2,3]) );
end.
```

The solution is to declare the parameter as a constant parameter (we do not intend to modify it, after all). Alternatively, you can also modify the call so it does not pass constants.

208. HIGH cannot be applied to a long string

Complete list of compiler error messages

It is not possible to use the standard function HIGH with long strings. The standard function HIGH can, however, be applied to old-style short strings.

Since long strings will dynamically size themselves, there is no analog to the HIGH function which can be used.

This error can be caused if you are porting a 16-bit application to, in which case the only string type available was a short string. If this is the case, then you can turn off the long strings with the $H command line switch or the long-form directive $LONGSTRINGS.

If the HIGH was applied to a string parameter, but you still wish to use long strings, you could change the parameter type to 'openstring'.

```
program Produce;
  var
    i : Integer;
    s : String;

begin
  s := 'Hello, Delphi';
  i := HIGH(s);
end.
```

In the example above, the programmer has attempted to apply the standard function HIGH to a long string variable. This cannot be done.

```
(*$LONGSTRINGS OFF*)
program Solve;
  var
    i : Integer;
    s : String;

begin
  s := 'Hello, Delphi';
  i := HIGH(s);
end.
```

By disabling long string parameters, the application of HIGH to a string variable is now allowed.

209. Unit '<Name>' implicitly imported into package '<Name>'

Complete list of compiler error messages

The unit specified was not named in the contains clause of the package, but a unit which has already been included in the package imports it.

This message will help the programmer avoid violating the rule that a unit may not reside in more than one related package.

Ignoring the warning, will cause the unit to be put into the package. You could also explicitly list the named unit in the contains clause of the package to accomplish the same result and avoid the warning altogether. Or, you could alter the package list to load the named unit from another package.

```
package Produce;
  contains Classes;
end.
```

In the above program, Classes uses (either directly or indirectly) 'consts', 'TypInfo', and 'SysUtils'. We will get a warning message for each of these units.

```
package Solve;
  contains consts, TypInfo, SysUtils, Classes;
end.
```

The best solution for this problem is to explicitly name all the units which will be imported into the package in the contains clause, as has been done here.

210. Packages '<name>' and '<name>' both contain unit '<name>'

Complete list of compiler error messages

The project you are trying to compile is using two packages which both contain the same unit. It is illegal to have two packages which are used in the same project containing the same unit since this would cause an ambiguity for the compiler.

A main cause of this problem is a package set which has been poorly defined.

The only solution to this problem is to redesign your package hierarchy to remove the ambiguity.

211. Package '<name>' already contains unit '<name>'

Complete list of compiler error messages

The package you are compiling requires (either through the requires clause or the package list) another package which already contains the unit specified in the message.

It is an error to have to related packages contain the same unit. The solution to this problem is to remove the unit from one of the packages or to remove the relation between the two packages.

212. File not found: '<name>.dcu'

Complete list of compiler error messages

The compiler needed to load the DCU file specified in the message but was unable to do so. Failure to set the unit/library path for the compiler is a likely cause of this message.

The only solution is to make sure the named unit can be found along the library path.

213. Need imported data reference ($G) to access '<name>' from unit '<name>'

Complete list of compiler error messages

The unit named in the message was not compiled with the $G switch turned on.

```
(*$IMPORTEDDATA OFF*)
unit u0;
interface
implementation
begin
  WriteLn(System.RandSeed);
end.

program u1;
  uses u0;
end.
```

In the above example, u0 should be compiled alone. Then, u1 should be compiled with the VCLxx (where xx represents the version). The problem occurs because u0 is compiled under the premise that it will never use data which resides in a package. in a package

```
(*$IMPORTEDDATA ON*)
unit u0;
interface
implementation
begin
  WriteLn(System.RandSeed);
end.

program u1;
  uses u0;
end.
```

To alleviate the problem, it is generally easiest just to turn on the $IMPORTEDDATA switch and recompile the unit which produces the error.

214. Required package '<name>' not found

Complete list of compiler error messages

The package which is referenced in the message appears on the package list, either explicitly or through a requires clause of another unit appearing on the package list, bit can not be found by the compiler.

The solution to this problem is to ensure that the DCP file for the named package is in one of the units named in the library path.

215. $WEAKPACKAGEUNIT '<name>' contains global data

Complete list of compiler error messages

A unit which was marked with $WEAKPACKAGEUNIT is being placed into a package, but it contains global data. It is not legal for such a unit to contain global data or initialization or finalization code.

The only solutions to this problem are to remove the $WEAKPACKAGEUNIT mark, or remove the global data from the unit before it is put into the package.

216. Improper GUID syntax

Complete list of compiler error messages

The GUID encountered in the program source is malformed. A GUID must be of the form: 00000000-0000-0000-0000-000000000000.

```
program Produce;

begin
end.
```

```
program Solve;

begin
end.
```

217. Interface type required

Complete list of compiler error messages

A type which is an interface was expected but was not found. A common cause of this error is the specification of a user-defined type which has not been declared as an interface type.

```
program Produce;
  type
    Name = string;

    MyObject = class
    end;

    MyInterface = interface(MyObject)
    end;

    Base = class(TObject, Name)
    end;

begin
end.
```

In this example, the type 'Base' is erroneously declared since 'Name' is not declared as an interface type. Likewise, 'MyInterface' is incorrectly declared because its ancestor interface was not declared as such.

```
program Solve;
  type
    BaseInterface = interface
    end;

    MyInterface = interface(BaseInterface)
    end;

    Base = class(TObject, MyInterface)
    end;

begin
end.
```

The best solution when encountering this error is to reexamine the source code to determine what was really intended. If a class is to implement an interface, it must first be explicitly derived from a base type such as TObject. When extended, interfaces can only have a single interface as its ancestor.

In the example above, the interface is properly derived from another interface and the object definition correctly specifies a base so that interfaces can be specified.

218. Property overrides not allowed in interface type

Complete list of compiler error messages

A property which was declared in a base interface has been overridden in an interface extension.

```
program Produce;
  type
    Base = interface
      function Reader : Integer;
      function Writer(a : Integer);
      property Value : Integer read Reader write Writer;
    end;

    Extension = interface (Base)
      function Reader2 : Integer;
      property Value Integer read Reader2;
    end;

begin
end.
```

The error in the example is that Extension attempts to override the Value property.

```
program Solve;
  type
    Base = interface
      function Reader : Integer;
      function Writer(a : Integer);
      property Value : Integer read Reader write Writer;
    end;

    Extension = interface (Base)
      function Reader2 : Integer;
      property Value2 Integer read Reader2;
    end;

begin
end.
```

A solution to this error is to rename the offending property. Another, more robust, approach is to determine the original intent and restructure the system design to solve the problem.

219. '<name>' clause not allowed in interface type

Complete list of compiler error messages

The clause noted in the message is not allowed in an interface type. Typically this error indicates that an illegal directive has been specified for a property field in the interface.

```
program Produce;
  type
    Base = interface
      function Reader : Integer;
      procedure Writer(a : Integer);
      property Value : Integer read Reader write Writer stored
false;
    end;
begin
end.
```

The problem in the above program is that the stored directive is not allowed in interface types.

```
program Solve;
  type
    Base = interface
      function Reader : Integer;
      procedure Writer(a : Integer);
      property Value : Integer read Reader write Writer;
    end;

begin
end.
```

The solution to problems of this nature are to remove the offending directive. Of course, it is best to understand the desired behavior and to implement it in some other fashion.

220. Interface '<name1>' already implemented by '<name2>'

Complete list of compiler error messages

The class specified by name2 has specified the interface name1 more than once in the inheritance section of the class definition.

```
program Produce;
  type
    IBaseIntf = interface
    end;

    TBaseClass = class (TInterfacedObject, IBaseIntf, IBaseIntf)
    end;

begin
end.
```

In this example, the IBaseIntf interface is specified multiple times in the inheritance section of the definition of TBaseClass. As a class can not implement the same interface more than once, this cause the compiler to emit the error message.

```
program Solve;

  type
    IBaseIntf = interface
    end;

    TBaseClass = class (TInterfacedObject, IBaseIntf)
    end;

begin
end.
```

The only solution to this error message is to ensure that a particular interface appears no more than once in the inheritance section of a class definition.

221. Field declarations not allowed in interface type

Complete list of compiler error messages

An interface has been encountered which contains definitions of fields; this is not permitted.

```
program Produce;
  type
    IBaseIntf = interface
      FVar : Integer;
      property Value : Integer read FVar write FVar;
    end;

begin
end.
```

The desire above is to have a property which has a value associated with it. However, as interfaces can have no fields, this idea will not work.

```
program Solve;
    IBaseIntf = interface
      function Reader : Integer;
      procedure Writer(a : Integer);
      property Value : Integer read Reader write Writer;
    end;

begin
end.
```

An elegant solution to the problem described above is to declare getter and setter procedures for the property. In this situation, any class implementing the interface must provide a method which will be used to access the data of the class.

222. '<name>' directive not allowed in interface type

Complete list of compiler error messages

A directive was encountered during the parsing of an interface which is not allowed.

```
program Produce;
  type
    IBaseIntf = interface
    private
      procedure fnord(x, y, z : Integer);
    end;

begin
end.
```

In this example, the compiler gives an error when it encounters the private directive, as it is not allowed in interface types.

```
program Solve;
  type
    IBaseIntf = interface
      procedure fnord(x, y, z : Integer);
    end;

    TBaseClass = class (TInterfacedObject, IBaseIntf)
    private
      procedure fnord(x, y, z : Integer);
    end;

  procedure TBaseClass.fnord(x, y, z : Integer);
  begin
  end;
begin
end.
```

The only solution to this problem is to remove the offending directive from the interface definition. While interfaces do not actually support these directives, you can place the implementing method into the desired visibility section. In this example, placing the TBaseClass.fnord procedure into a private section should have the desired results.

223. Declaration of '<name1>' differs from declaration in interface '<name2>'

Complete list of compiler error messages

A method declared in a class which implements an interface is different from the definition which appears in the interface. Probable causes are that a parameter type or return value is declared differently, the method appearing in the class is a message method, the identifier in the class is a field or the identifier in the class is a property, which does not match with the definition in the interface.

```
program Produce;

  type
    IBaseIntf = interface
      procedure p0(var x : Shortint);
      procedure p1(var x : Integer);
      procedure p2(var x : Integer);
    end;

    TBaseClass = class (TInterfacedObject)
      procedure p1(var x : Integer); message 151;
    end;

    TExtClass = class (TBaseClass, IBaseIntf)
      p2 : Integer;
      procedure p0(var x : Integer);
      procedure p1(var x : Integer); override;
    end;

  procedure TBaseClass.p1(var x : Integer);
  begin
  end;

  procedure TExtClass.p0(var x : Integer);
  begin
  end;

  procedure TExtClass.p1(var x : Integer);
  begin
  end;

begin
end.
```

Generally, as in this example, errors of this type are plain enough to be easily visible. However, as can be seen with p1, things can be more subtle. Since p1 is overriding a procedure from the inherited class, p1 also inherits the virtuality of the procedure defined in the base class.

```
program Solve;

  type
    IBaseIntf = interface
      procedure p0(var x : Shortint);
      procedure p1(var x : Integer);
      procedure p2(var x : Integer);
    end;

    TBaseClass = class (TInterfacedObject)
      procedure p1(var x : Integer); message 151;
    end;

    TExtClass = class (TBaseClass, IBaseIntf)
      p2 : Integer;

      procedure IBaseIntf.p1 = p3;
      procedure IBaseIntf.p2 = p4;

      procedure p0(var x : Shortint);
      procedure p1(var x : Integer); override;
      procedure p3(var x : Integer);
      procedure p4(var x : Integer);
    end;

  procedure TBaseClass.p1(var x : Integer);
  begin
  end;

  procedure TExtClass.p0(var x : Shortint);
  begin
  end;

  procedure TExtClass.p1(var x : Integer);
  begin
  end;

  procedure TExtClass.p3(var x : Integer);
  begin
  end;

  procedure TExtClass.p4(var x : Integer);
  begin
  end;

begin
end.
```

One approach to solving this problem is to use a message resolution clause for each problematic identifier, as is done in the example shown here. Another viable approach, which requires more thoughtful design, would be to ensure that the class identifiers are compatible to the interface identifiers before compilation.

224. Package unit '<name>' cannot appear in contains or uses clauses

Complete list of compiler error messages

The unit named in the error is a package unit and as such cannot be included in your project. A possible cause of this error is that somehow a Pascal unit and a package unit have been given the same name. The compiler is finding the package unit on its search path before it can locate a same-named Pascal file. Packages cannot be included in a project by inclusion of the package unit in the uses clause.

225. Bad packaged unit format: <name>.<name>

Complete list of compiler error messages

When the compiler attempted to load the specified unit from the package, it was found to be corrupt. This problem could be caused by an abnormal termination of the compiler when writing the package file (for example, a power loss). The first recommended action is to delete the offending DCP file and recompile the package. If this fails, contact Inprise Developer Support.

226. Package '<name>' is recursively required

Complete list of compiler error messages

When compiling a package, the compiler determined that the package requires itself. the

```
package Produce;
  requires Produce;

end.
```

The error is caused because it is not legal for a package to require itself.

The only solution to this problem is to remove the recursive use of the package.

227. 16-Bit segment encountered in object file '<name>'

Complete list of compiler error messages

A 16-bit segment has been found in an object file which was loaded using the $L directive.

```
end.
```

The only solution to this error is to obtain an object file which does not have a 16-bit segment definition. You should consult the documentation for the product which produced the object file for instructions on turning 16-bit segment definitions into 32-bit segment definitions.

228. Published field '<name>' not a class nor interface type

Complete list of compiler error messages

An attempt has been made to publish a field in a class which is not a class nor interface type.

```
program Produce;

  type
    TBaseClass = class
    published
      x : Integer;
    end;
begin
end.
```

The program above generates an error because x is included in a published section, despite the fact that it is not of a type which can be published.

```
program Solve;
  type
    TBaseClass = class
      Fx : Integer;
    published
      property X : Integer read Fx write Fx;
    end;

begin
end.
```

To solve this problem, all fields which are not class nor interface types must be removed from the published section of a class. If it is a requirement that the field actually be published, then it can be accomplished by changing the field into a property, as was done in this example.

229. Private symbol '<name>' declared but never used

Complete list of compiler error messages

The symbol referenced appears in a private section of a class, but is never used by the class. It would be more memory efficient if you removed the unused private field from your class definition.

```
program Produce;
  type
    Base = class
    private
      FVar : Integer;
      procedure Init;
    end;

procedure Base.Init;
begin
end;

begin
end.
```

Here we have declared a private variable which is never used. The message will be emitted for this case.

```
program Solve;
program Produce;
  type
    Base = class
    private
      FVar : Integer;
      procedure Init;
    end;

procedure Base.Init;
begin
  FVar := 0;
end;

begin
end.
```

There are various solutions to this problem, and since this message is not an error message, all are correct. If you have included the private field for some future use, it would be valid to ignore the message. Or, if the variable is truly superfluous, it can be safely removed. Finally, it might have been a programming oversight not to use the variable at all; in this case, simply add the code you forgot to implement.

230. Could not compile package '<name>'

Complete list of compiler error messages

An error occurred while trying to compile the package named in the message. The only solution to the problem is to correct the error and recompile the package.

231. Never-build package '<name>' requires always-build package '<name>'

Complete list of compiler error messages

You are attempting to create a no-build package which requires an always-build package. Since the interface of an always-build package can change at anytime, and since giving the no-build flag instructs the compiler to assume that a package is up-to-date, each no-build package can only require other packages that are also marked no-build.

```
package Base;
end.

(*$IMPLICITBUILD OFF*)
package NoBuild;
  requires Base;
end.
```

In this example, the NoBuild package requires a package which was compiled in the always-build compiler state.

```
(*$IMPLICITBUILD OFF*)
package Base;
end.

(*$IMPLICITBUILD OFF*)
package NoBuild;
  requires Base;
end.
```

The solution used in this example was to turn Base into a never-build package. Another viable option would have been to remove the (*$IMPLICITBUILD OFF*) from the NoBuild package, thereby turning it into an always-build package.

232. $WEAKPACKAGEUNIT '<name>' cannot have initialization or finalization code

Complete list of compiler error messages

A unit which has been flagged with the $weakpackageunit directive cannot contain initialization or finalization code, nor can it contain global data. The reason for this is that multiple copies of the same weakly packaged units can appear in an application, and then referring to the data for that unit becomes and ambiguous proposition. This ambiguity is furthered when dynamically loaded packages are used in your applications.

```
(*$WEAKPACKAGEUNIT*)
unit yamadama;
interface
implementation
  var
    Title : String;

initialization
  Title := 'Tiny Calc';
finalization
end.
```

In the above example, there are two problems: Title is a global variable, and Title is initialized in the initialization section of the unit.

There are only two alternatives: either remove the $weakpackageunit directive from the unit, or remove all global data, initialization and finalization code.

233. $WEAKPACKAGEUNIT & $DENYPACKAGEUNIT both specified

Complete list of compiler error messages

It is not legal to specify both $WEAKPACKAGEUNIT & $DENYPACKAGEUNIT. Correct the source code and recompile.

234. $DENYPACKAGEUNIT '<name>' cannot be put into a package

Complete list of compiler error messages

You are attempting to put a unit which was compiled with $DENYPACKAGEUNIT into a package. It is not possible to put a unit compiled with the $DENYPACKAGEUNIT direction into a package.

235. $DESIGNONLY and $RUNONLY only allowed in package unit

Complete list of compiler error messages

The compiler has encountered either $designonly or $runonly in a source file which is not a package. These directives affect the way that the IDE will treat a package DLL, and therefore can only be contained in package source files.

236. Never-build package '<name>' must be recompiled

Complete list of compiler error messages

The package referenced in the message was compiled as a never-build package, but it requires another package to which interface changes have been made. The named package cannot be used without recompiling because it was linked with a different interface of the required package.

The only solution to this error is to manually recompile the offending package. Be sure to specify the never-build switch, if it is still desired.

237. Compilation terminated; too many errors

Complete list of compiler error messages

The compiler has surpassed the maximum number of errors which can occur in a single compilation.

The only solution is to address some of the errors and recompile the project.

238. Imagebase is too high - program exceeds 2 GB limit

Complete list of compiler error messages

There are three ways to cause this error: 1. Specify a large enough imagebase that, when compiled, the application code passes the 2GB boundary. 2. Specify an imagebase via the command line which is above 2GB. 3. Specify an imagebase via $imagebase which is above 2GB.

The only solution to this problem is to lower the imagebase address sufficiently so that the entire application will fit below the 2GB limit.

239. A dispinterface type cannot have an ancestor interface

Complete list of compiler error messages

An interface type specified with dispinterface cannot specify an ancestor interface.

```
program Produce;

  type
    IBase = interface
    end;

    IExtend = dispinterface (IBase)
    ['{00000000-0000-0000-0000-000000000000}']

    end;

begin
end.
```

In the example above, the error is caused because IExtend attempts to specify an ancestor interface type.

```
program Solve;

  type
    IBase = interface
    end;

    IExtend = dispinterface
    ['{00000000-0000-0000-0000-000000000000}']

    end;

begin
end.
```

Generally there are two solutions when this error occurs: remove the ancestor interface declaration, or change the dispinterface into a regular interface type. In the example above, the former approach was taken.

240. A dispinterface type requires an interface identification

Complete list of compiler error messages

When using dispinterface types, you must always be sure to include a GUID specification for them.

```
program Produce;

  type
    IBase = dispinterface
    end;

begin
end.
```

In the example shown here, the dispinterface type does not include a GUID specification, and thus causes the compiler to emit an error.

```
program Solve;

  type
    IBase = dispinterface
    ['{00000000-0000-0000-0000-000000000000}']

    end;

begin
end.
```

Ensuring that each dispinterface has a GUID associated with it will cause this error to go away.

241. Methods of dispinterface types cannot specify directives

Complete list of compiler error messages

Methods declared in a dispinterface type cannot specify any calling convention directives.

```
program Produce;

  type
    IBase = dispinterface
    ['{00000000-0000-0000-0000-000000000000}']
      procedure yamadama; register;
    end;

begin
end.
```

The error in the example shown here is that the method 'yamadama' attempts to specify the register calling convention.

```
program Solve;

  type
    IBase = dispinterface
    ['{00000000-0000-0000-0000-000000000000}']
      procedure yamadama;
    end;

begin
end.
```

Since no dispinterface method can specify calling convention directives, the only solution to this problem is to remove the offending directive, as shown in this example.

242. '<text>' directive not allowed in dispinterface type

Complete list of compiler error messages

You have specified a clause in a dispinterface type which is not allowed.

```
program Produce;

  type
    IBase = dispinterface
    ['{00000000-0000-0000-0000-000000000000}']
      function Get : Integer;

      property BaseValue : Integer read Get;
    end;

    IExt = interface (IBase)
    end;


begin
end.
```

```
program Solve;

  type
    IBase = dispinterface
    ['{00000000-0000-0000-0000-000000000000}']
      function Get : Integer;

      property BaseValue : Integer;
    end;

begin
end.
```

243. Interface '<name>' has no interface identification

Complete list of compiler error messages

You have attempted to assign an interface to a GUID type, but the interface was not defined with a GUID.

```
program Produce;

  type
    IBase = interface
    end;

  var
    g : TGUID;

  procedure p(x : TGUID);
  begin
  end;

begin
  g := IBase;
  p(IBase);
end.
```

In this example, the IBase type is defined but it is not given an interface, and is thus cannot be assigned to a GUID type.

```
program Solve;

  type
    IBase = interface
    ['{00000000-0000-0000-0000-000000000000}']
    end;

  var
    g : TGUID;

  procedure p(x : TGUID);
  begin
  end;

begin
  g := IBase;
  p(IBase);
end.
```

To solve the problem, you must either not attempt to assign an interface type without a GUID to a GUID type, or you must assign a GUID to the interface when it is defined. In this solution, a GUID has been assigned to the interface type when it is defined.

244. Property '<name>' inaccessible here

Complete list of compiler error messages

An attempt has been made to access a property through a class reference type. It is not possible to access fields nor properties of a class through a class reference.

```
program Produce;

  type
    TBase = class
    public
      FX : Integer;
      property X : Integer read FX write FX;
    end;

    TBaseClass = class of TBase;

  var
    BaseRef : TBaseClass;
    x : Integer;

begin
  BaseRef := TBase;
  x := BaseRef.X;
end.
```

Attempting to access the property X in the example above causes the compiler to issue an error.

```
program Solve;

  type
    TBase = class
    public
      FX : Integer;
      property X : Integer read FX write FX;
    end;

    TBaseClass = class of TBase;

  var
    BaseRef : TBaseClass;
    x : Integer;

begin
  BaseRef := TBase;
end.
```

There is no other solution to this problem than to remove the offending property access from your source code. If you wish to access properties or fields of a class, then you need to create an instance variable of that class type and gain access through that variable.

245. Unsupported language feature: '<text>'

Complete list of compiler error messages

You are attempting to translate a Pascal unit to a C++ header file which contains unsupported language features.

You must remove the offending construct from the interface section before the unit can be translated.

246. Getter or setter for property '<name>' cannot be found

Complete list of compiler error messages

During translation of a unit to a C++ header file, the compiler is unable to locate a named symbol which is to be used as a getter or setter for a property. This is usually caused by having nested records in the class and the accessor is a field in the nested record.

247. Package '<name>' does not use or export '<unit>.<name>'

Complete list of compiler error messages

You have compiled a unit into a package which contains a symbol which does not appear in the interface section of the unit, nor is it referenced by any code in the unit. In effect, this code is dead code and could be removed from the unit without changing the semantics of your program.

248. Constructors and destructors must have register calling convention

Complete list of compiler error messages

An attempt has been made to change the calling convention of a constructor or destructor from the default register calling convention.

```
program Produce;

  type
    TBase = class
      constructor Create; pascal;
    end;

  constructor TBase.Create;
  begin
  end;

begin
end.
```

```
program Solve;

  type
    TBase = class
      constructor Create;
    end;

  constructor TBase.Create;
  begin
  end;

begin
end.
```

The only viable approach when this error has been issued by the compiler is to remove the offending calling convention directive from the constructor or destructor definition, as has been done in this example.

249. Parameter '<name>' not allowed here due to default value

Complete list of compiler error messages

When using default parameters a list of parameters followed by a type is not allowed; you must specify each variable and its default value individually.

```
program Produce;

  procedure p0(a, b : Integer = 151);
  begin
  end;

begin
end.
```

The procedure definitions shown above will cause this error since it declares two parameters with a default value.

```
program Solve;

  procedure p0(a : Integer; b : Integer = 151);
  begin
  end;

  procedure p1(a : Integer = 151; b : Integer = 151);
  begin
  end;

begin
end.
```

Depending on the desired result, there are different ways of approaching this problem. If only the last parameter is supposed to have the default value, then take the approach shown in the first example. If both parameters are supposed to have default values, then take the approach shown in the second example.

250. Default value required for '<name>'

Complete list of compiler error messages

```
program Produce;

  procedure p0(a : Integer = 151; b : Char);
  begin
  end;

  procedure p1(a : Integer = 151; b : Char);
  begin
  end;

begin
end.
```

The two procedures definitions shown above both will cause this error since they both declare a non-default parameter following a default value.

```
program Solve;

  procedure p0(a : Integer = 151; b : Char = 'A');
  begin
  end;

  procedure p1(b : Char; a : Integer = 151);
  begin
  end;

begin
end.
```

There are two ways of approaching this problem: add default values or rearrange the order of the parameters. As can be seen in the two example procedures above, both are simple to do.

251. Default parameter '<name>' must be by-value or const

Complete list of compiler error messages

Parameters which are given default values cannot be passed by reference.

```
program Produce;

  procedure p0(var x : Integer = 151);
  begin
  end;

begin
end.
```

Since the parameter x is passed by reference in this example, it cannot be given a default value.

```
program Solve;

  procedure p0(const x : Integer = 151);
  begin
  end;

begin
end.
```

In this solution, the by-reference parameter has been changed into a const parameter. Alternatively it could have been changed into a by-value parameter or the default value could have been removed.

252. Constant 0 converted to NIL

Complete list of compiler error messages

The Pascal compiler now allows the constant 0 to be used in pointer expressions in place of NIL. This change was made to allow older code to still compile with changes which were made in the low-level RTL.

```
program Produce;

  procedure p0(p : Pointer);
  begin
  end;

begin
  p0(0);
end.
```

In this example, the procedure p0 is declared to take a Pointer parameter yet the constant 0 is passed. The compiler will perform the necessary conversions internally, changing 0 into NIL, so that the code will function properly.

```
program Solve;

  procedure p0(p : Pointer);
  begin
  end;

begin
  p0(NIL);
end.
```

There are two approaches to solving this problem. In the case above the constant 0 has been replaced with NIL. Alternatively the procedure definition could be changed so that the parameter type is of Integer type.

253. $EXTERNALSYM and $NODEFINE not allowed for '<name>'; only global symbols

Complete list of compiler error messages

The $EXTERNALSYM and $NODEFINE directives can only be applied to global symbols.

254. $HPPEMIT '<text>' ignored

Complete list of compiler error messages

The $HPPEMIT directive can only appear after the unit header.

255. Integer and HRESULT interchanged

Complete list of compiler error messages

In Pascal Integer, Longint and HRESULT are compatible types, but in C++ the types are not compatible and will produce differently mangled C++ parameter names. To ensure that there will not be problems linking object files created with the Pascal compiler this message alerts you to possible problems. If you are compiling your source to an object file, this is an error otherwise it will be presented as a warning.

```
program Produce;
  uses Windows;

  type
    I0 = interface (IUnknown)
      procedure p0(var x : Integer);
    end;

    C0 = class (TInterfacedObject, I0)
      procedure p0(var x : HRESULT);
    end;

  procedure C0.p0(var x : HRESULT);
  begin
  end;

begin
end.
```

The example shown here declares the interface and class methods differently. While they are equivalent in Pascal they are not so in C++.

```
program Solve;

  uses Windows;

  type
    I0 = interface (IUnknown)
      procedure p0(var x : Integer);
    end;

    C0 = class (TInterfacedObject, I0)
      procedure p0(var x : Integer);
    end;

  procedure C0.p0(var x : Integer);
  begin
  end;

begin
end.
```

The easiest solution to this problem is to match the class-declared methods to be identical to the interface-declared methods.

256. C++ obj files must be generated (-jp)

Complete list of compiler error messages

Because of the language features used, standard C object files cannot be generated for this unit. You must generate C++ object files.

257. '<name>' is not the name of a unit

Complete list of compiler error messages

The $NOINCLUDE directive must be given a known Pascal unit name.

258. Expression needs no Initialize/Finalize

Complete list of compiler error messages

You have attempted to use the standard procedure Finalize on a Pascal type which requires no finalization.

```
program Produce;

  var
    ch : Char;

begin
  Finalize(ch);
end.
```

In this example, the Pascal type Char needs no finalization.

The usual solution to this problem is to remove the offending use of Finalize.

259. Pointer expression needs no Initialize/Finalize - need ^ operator?

Complete list of compiler error messages

You have attempted to finalize a Pointer type.

```
program Produce;

  var
    str : String;
    pstr : PString;

begin
  str := 'Sharene';
  pstr := @str;
  Finalize(pstr);  (*note: do not attempt to use 'str' after
this*)
end.
```

In this example the pointer, pstr, is passed to the Finalize procedure. This causes an hint since pointers do not require finalization.

```
program Solve;


  var
    str : String;
    pstr : PString;

begin
  str := 'Sharene';
  pstr := @str;
  Finalize(pstr^);  (*note: do not attempt to use 'str' after
this*)
end.
```

The solution to this problem is to apply the ^ operator to the pointer which is passed to the Finalization procedure.

260. Recursive include file <name>

Complete list of compiler error messages

The $I directive has been used to recursively include another file. You must check to make sure that all include files terminate without having cycles in them.

261. Need to specify at least one dimension for SetLength of dynamic array

Complete list of compiler error messages

The standard procedure `SetLength` has been called to alter the length of a dynamic array, but no array dimensions have been specified.

```
program Produce;

  var
    arr : array of integer;

begin
  SetLength(arr);
end.
```

The `SetLength` in the above example causes an error since no array dimensions have been specified.

```
program solve;

  var
    arr : array of integer;

begin
  SetLength(arr, 151);
end.
```

To remove this error from your program, specify the number of elements you wish the array to contain.

262. Cannot take the address when compiling to byte code

Complete list of compiler error messages

The *address-of* operator, `@`, cannot be used when compiling to byte code.

263. Cannot use old style object types when compiling to byte code

Complete list of compiler error messages

Old-style `Object` types are illegal when compiling to byte code.

264. Cannot use absolute variables when compiling to byte code

Complete list of compiler error messages

The use of absolute variables is prohibited when compiling to byte code.

265. There is no overloaded version of '<name>' that can be called with these arguments

Complete list of compiler error messages

An attempt has been made to call an overloaded function which cannot be resolved with the current set of overloads.

```
program Produce;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : char); overload;
begin
end;

begin
  f0(1.2);
end.
```

The overloaded procedure `f0` has two versions: one which takes a `char` and one which takes an `integer`. However, the call to `f0` uses a floating point type, which the compiler cannot resolve into neither a `char` nor an `integer`.

```
program Solve;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : char); overload;
begin
end;

begin
  f0(1);
end.
```

There are two basic ways to solve this problem: either supply a parameter type which can be resolved into a match of an overloaded procedure, or create a new version of the overloaded procedure which matches the parameter type.

In the example above, the parameter type has been modified to match one of the existing overloaded versions of `f0`.

266. Ambiguous overloaded call to '<name>'

Complete list of compiler error messages

Based on the current overload list for the specified function, and the programmed invocation, the compiler is unable to determine which version of the procedure should be invoked.

```
program Produce;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : integer; b : char = 'A'); overload;
begin
end;

begin
  f0(1);
end.
```

In this example, the default parameter that exists in one of the versions of `f0` makes it impossible for the compiler to determine which procedure should actually be called.

```
program Solve;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : integer; b : char); overload;
begin
end;

begin
  f0(1);
end.
```

The approach taken in this example was to remove the default parameter value. The result here is that the procedure taking only one `integer` parameter will be called. It should be noted that this approach is the only way that the single-parameter function can be called.

267. Method '<name>' with identical parameters exists already

Complete list of compiler error messages

A method with an identical signature already exists in the data type.

```
program Produce;

  type
    t0 = class
      procedure f0(a : integer); overload;
      procedure f0(a : integer); overload;
    end;

procedure T0.f0(a : integer);
begin
end;

begin
end.
```

The error is produced here because there are two overloaded declarations for the same procedure.

```
program Solve;

  type
    t0 = class
      procedure f0(a : integer); overload;
      procedure f0(a : char); overload;
    end;

procedure T0.f0(a : integer);
begin
end;

procedure T0.f0(a : char);
begin
end;

begin
end.
```

There are different approaches to curing this error. One approach is to remove the redundant declaration of the procedure. Another approach, taken here, is to change the parameter type of the duplicate declarations so that it creates a unique version of the overloaded procedure.

268. Ancestor type '<name>' does not have default constructor

Complete list of compiler error messages

The ancestor of the class being compiled does not have a default constructor. This error only occurs with the byte code version of the compiler.

269. Overloaded procedure '<name>' must be marked with the 'overload' directive

Complete list of compiler error messages

The compiler has encountered a procedure, which is not marked `overload`, with the same name as a procedure already marked `overload`. All overloaded procedures must be marked as such.

```
program Produce;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : integer; ch : char);
begin
end;

begin
end.
```

The procedure `f0(a : integer; ch : char)` causes the error since it is not marked with the `overload` keyword.

```
program solve;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : integer; ch : char); overload;
begin
end;

begin
end.
```

If the procedure is intended to be an overloaded version, then mark it as `overload`. If it is not intended to be an overloaded version, then change its name.

270. Class methods not allowed as property getters or setters

Complete list of compiler error messages

The compiler has encountered a property declaration which specified a "class methods" as its getter or setter method. These special method types have different semantics, such as not being able to access instance data, and cannot be used for property accessors.

```
program Produce;

  type
    T0 = class
      ch : char;
      class procedure access(a : char);
      property CharValue : Char read ch write access;
    end;

class procedure T0.access(a : char);
begin
end;

begin
end.
```

The solution to this problem is to change your program so that it does not use class methods as property accessors.

271. New not supported for dynamic arrays - use SetLength

Complete list of compiler error messages

The program has attempted to use the standard procedure `NEW` on a dynamic array. The proper method for allocating dynamic arrays is to use the standard procedure `SetLength`.

```
program Produce;
  var
    arr : array of integer;

begin
  new(arr, 10);
end.
```

The standard procedure `NEW` cannot be used on dynamic arrays.

```
program Solve;
  var
    arr : array of integer;

begin
  SetLength(arr, 10);
end.
```

Use the standard procedure `SetLength` to allocate dynamic arrays.

272. Dispose not supported (nor necessary) for dynamic arrays

Complete list of compiler error messages

The compiler has encountered a use of the standard procedure `DISPOSE` on a dynamic array. Dynamic arrays are reference counted and will automatically free themselves when there are no longer any references to them.

```
program Produce;
  var
    arr : array of integer;

begin
  SetLength(arr, 10);
  Dispose(arr);
end.
```

The use of `DISPOSE` on the dynamic array `arr` causes the error in this example.

```
program Produce;
  var
    arr : array of integer;

begin
  SetLength(arr, 10);
end.
```

The only solution here is to remove the offending use of `DISPOSE`

273. Duplicate implements clause for interface <name>

Complete list of compiler error messages

The compiler has encountered two different `property` declarations which claim to `implement` the same `interface`. An `interface` may be implemented by only one `property`.

```
program Produce;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IMyInterface)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface read FMyInterface
implements IMyInterface;
    property OtherInterface: IMyInterface read FMyInterface
implements IMyInterface;
  end;
end.
```

Both `MyInterface` and `OtherInterface` attempt to implement `IMyInterface`. Only one property may implement the chosen interface.

The only solution in this case is to remove one of the offending `implements` clauses.

274. Implements clause only allowed within class types

Complete list of compiler error messages

```
program Produce;
type
  IMyInterface = interface
    function getter : IMyInterface;
    property MyInterface: IMyInterface read getter implements
IMyInterface;
  end;
end.
```

The `interface` definition in this example attempt to use an implements clause which causes the error.

```
program Solve;
type
  IMyInterface = interface
    function getter : IMyInterface;
    property MyInterface: IMyInterface read getter;
  end;
end.
```

The only viable solution to this problem is to remove the offending `implements` clause.

275. Implements clause only allowed for properties of class or interface type

Complete list of compiler error messages

An attempt has been made to use the `implements` clause with an improper type. Only `class` or `interface` types may be used.

```
program Produce;
type
  TMyClass = class(TInterfacedObject)
    FInteger : Integer;
    property MyInterface: Integer read FInteger implements
Integer;
  end;
end.
```

In this example the error is caused because an `Integer` type is used with an `implements` clause.

The only solution for this error is to correct the `implements` clause so that it refers to a `class` or `interface` type, or to remove the offending clause altogether.

276. Implements clause not allowed together with index clause

Complete list of compiler error messages

277. Implements clause only allowed for readable property

Complete list of compiler error messages

The compiler has encountered a "write only" property that claims to `implement` an `interface`. A `property` must be read/write to use the `implements` clause.

```
program Produce;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IMyInterface)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface implements IMyInterface;
  end;
end.
```

The property in this example is *write only* and cannot be used to implement an interface.

```
program Solve;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IMyInterface)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface read FMyInterface
implements IMyInterface;
  end;
end.
```

by adding a `read` clause, the property can use the `implements` clause.

278. Implements getter must be register calling convention

Complete list of compiler error messages

The compiler has encountered a getter or setter which does not have `register` calling convention.

```
program Produce;
type
  I0 = interface
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0; cdecl;
    property p0 : I0 read getter implements I0;
  end;

function T0.getter : I0;
begin
end;
end.
```

As can be seen in this example, the `cdecl` on the function `getter` causes this error to be produced.

```
program Solve;
type
  I0 = interface
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0;
    property p0 : I0 read getter implements I0;
  end;

function T0.getter : I0;
begin
end;
end.
```

The only solution to this problem is to remove the offending *calling convention* from the property getter declaration.

279. Implements getter cannot be dynamic or message method

Complete list of compiler error messages

An attempt has been made to use a `dynamic` or `message` method as a property accessor of a property which has an `implements` clause.

```
program Produce;
type
  I0 = interface
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0; dynamic;
    property p0 : I0 read getter implements I0;
  end;

function T0.getter : I0;
begin
end;

end.
```

As shown in the example here, it is an error to use the `dynamic` modifier on a getter for a property which has an `implements` clause.

```
program Produce;
type
  I0 = interface
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0;
    property p0 : I0 read getter implements I0;
  end;

function T0.getter : I0;
begin
end;

end.
```

To remove this error from your programs, remove the offending `dynamic` or `method` declaration.

280. Cannot have method resolutions for interface '<name>'

Complete list of compiler error messages

An attempt has been made to use a method resolution clause for an interface named in an `implements` clause.

```
program Produce;
type
  I0 = interface
    procedure i0p0(a : char);
  end;

  T0 = class(TInterfacedObject, I0)
    procedure I0.i0p0 = proc0;
    function getter : I0;
    procedure proc0(a : char);
    property p0 : I0 read getter implements I0;
  end;

procedure T0.proc0(a : char);
begin
end;

function T0.getter : I0;
begin
end;
end.
```

In this example the method `proc0` is mapped onto the interface procedure `i0p0`, but because the interface is mentioned in a `implements` clause, this renaming is not allowed.

```
program Solve;
type
  I0 = interface
    procedure i0p0(a : char);
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0;
    procedure i0p0(a : char);
    property p0 : I0 read getter implements I0;
  end;

procedure T0.i0p0(a : char);
begin
end;

function T0.getter : I0;
begin
end;
end.
```

The solution for this error is to remove the offending "name resolution clause". One easy way to accomplish this is to name the procedure in the class to the same name as the interface method.

281. Interface '<name>' not mentioned in interface list

Complete list of compiler error messages

An `implements` clause references an interface which is not mentioned in the interface list of the class.

```
program Produce;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IUnknown)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface read FMyInterface
implements IMyInterface;
  end;
end.
```

The example shown here uses `implements` with the `IMyInterface` interface, but it is not mentioned in the interface list.

```
program Solve;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IUnknown, IMyInterface)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface read FMyInterface
implements IMyInterface;
  end;
end.
```

A quick solution, shown here, is to add the required interface to the interface list of the class definition. Of course, adding it to the interface list might require the implementation of the methods of the interface.

282. Exported package threadvar '<name>.<name>' cannot be used outside of this package

Complete list of compiler error messages

Windows does not support the exporting of `threadvar` variables from a DLL, but since using Delphi packages is meant to be semantically equivalent to compiling a project without them, the Pascal compiler must somehow attempt to support this construct.

This warning is to notify you that you have included a unit which contains a `threadvar` in an interface into a package. While this is not illegal, you will not be able to access the variable from a unit outside the package.

Attempting to access this variable may appear to succeed, but it actually did not.

A solution to this warning is to move the `threadvar` to the `implementation` section and provide function which will retrieve the variables value.

283. Only one of a set of overloaded methods can be published

Complete list of compiler error messages

Only one member of a set of overloaded functions may be published because the RTTI generated for procedures only contains the name.

```
(*$M+*)
(*$APPTYPE CONSOLE*)
program Produce;
type
  Base = class
  published
    procedure p1(a : integer); overload;
    procedure p1(a : boolean); overload;
  end;

  Extended = class (Base)
    procedure e1(a : integer); overload;
    procedure e1(a : boolean); overload;
  end;

  procedure Base.p1(a : integer);
  begin
  end;

  procedure Base.p1(a : boolean);
  begin
  end;

  procedure Extended.e1(a : integer);
  begin
  end;

  procedure Extended.e1(a : boolean);
  begin
  end;

end.
```

In the example shown here, both overloaded `p1` functions are contained in a `published` section, which is not allowed.

Further, since the `$M+` state is used, the `Extended` class starts with `published` visibility, thus the error will also appear for this class also.

```
(*$M+*)
(*$APPTYPE CONSOLE*)
program Solve;
type
  Base = class
  public
    procedure p1(a : integer); overload;
  published
    procedure p1(a : boolean); overload;
  end;

  Extended = class (Base)
  public
    procedure e1(a : integer); overload;
    procedure e1(a : boolean); overload;
  end;

  procedure Base.p1(a : integer);
  begin
  end;

  procedure Base.p1(a : boolean);
  begin
  end;

  procedure Extended.e1(a : integer);
  begin
  end;

  procedure Extended.e1(a : boolean);
  begin
  end;

end.
```

The solution here is to ensure that no more than one member of a set of overloaded function appears in a `published` section. The easiest way to achieve this is to change the visibility to `public`, `protected` or `private`; whichever is most appropriate.

284. Previous declaration of '<name>' was not marked with the 'overload' directive

Complete list of compiler error messages

```
program Produce;
type
  Base = class
    procedure func(a : integer);
    procedure func(a : char); overload;
  end;

  procedure Base.func(a : integer);
  begin
  end;

  procedure Base.func(a : char);
  begin
  end;

end.
```

This example attempts to overload the `char` version of `func` without marking the first version of `func` as overloadable.

You must mark all functions to be overloaded with the `overload` directive. If `overload` were not required on all versions it would be possible to introduce a new method which overloads an existing method and then a simple recompilation of the source could produce different behavior.

```
program Solve;
type
  Base = class
    procedure func(a : integer); overload;
    procedure func(a : char); overload;
  end;

  procedure Base.func(a : integer);
  begin
  end;

  procedure Base.func(a : char);
  begin
  end;

end.
```

There are two solutions to this problem. You can either remove the attempt at overloading or you can mark the original declaration with the `overload` directive. The example shown above marks the original declaration.

285. Parameters of this type cannot have default values

Complete list of compiler error messages

The default parameter mechanism incorporated into the Delphi Pascal compiler allows only simple types to be initialized in this manner. You have attempted to use a type that is not supported.

```
program Produce;
type
  ArrayType = array [0..1] of integer;

  procedure p1(proc : ArrayType = [1, 2]);
  begin
  end;
end.
```

Default parameters of this type are not supported in Delphi Pascal.

```
program solve;
type
  ArrayType = array [0..1] of integer;

  procedure p1(proc : ArrayType);
  begin
  end;

end.
```

The only way to get rid of this error is to remove the offending parameter assignment or to change the type of the parameter to one that can be initialized with a default value.

286. Overriding virtual method '<class>.<method>' has a lower visibility than base class

Complete list of compiler error messages

The method named in the error message has been declared as an override of a virtual method in a base class, but the visibility in the current class is lower than that used in the base class for the same method.

While the visibility rules of Pascal would seem to indicate that the function cannot be seen, the rules of invoking virtual functions will cause the function to be properly invoked through a virtual call.

Generally this means that the method of the derived class was declared in a `private` or `protected` section while the method of the base class was declared in a `protected` or `pubic` (including `published`) section respectively.

```
unit Produce;
interface

  type
    Base = class(TObject)
    public
      procedure VirtualProcedure(X: Integer); virtual;
    end;

    Extended = class(Base)
    protected
      procedure VirtualProcedure(X: Integer); override;
    end;

implementation

  procedure Base.VirtualProcedure(X: Integer);
  begin
  end;

  procedure Extended.VirtualProcedure(X: Integer);
  begin
  end;
end.
```

The example above aptly shows how this error is produced by putting `Extended.VirtualProcedure` into the `protected` section.

In practice this is never harmful, but it can be confusing to someone reading documentation and observing the visibility attributes of the document.

This hint will only be produced for classes appearing in the interface section of a unit.

```
unit Solve;
interface

  type
    Base = class(TObject)
    public
      procedure VirtualProcedure(X: Integer); virtual;
    end;

    Extended = class(Base)
    public
      procedure VirtualProcedure(X: Integer); override;
    end;

implementation

  procedure Base.VirtualProcedure(X: Integer);
  begin
  end;

  procedure Extended.VirtualProcedure(X: Integer);
  begin
  end;
end.
```

There are three basic solutions to this problem.

1. Ignore the hint
2. Change the visibility to match the base class
3. Move class definition to the implementation section.

The example program above has taken the approach of changing the visibility to match the base class.

287. Published property getters and setters must have register calling convention

Complete list of compiler error messages

A property appearing in a `published` section has a *getter* or *setter* procedure that does not have the `register` calling convention.

```
unit Produce;
interface
  type
    Base = class
    public
      function getter : Integer; cdecl;
    published
      property Value : Integer read getter;
    end;

implementation
function Base.getter : Integer;
begin getter := 0;
end;

end.
```

This example declares the getter function `getter` for the `published` property `Value` to be of `cdecl` calling convention, which produces the error.

```
unit Solve;
interface
  type
    Base = class
    public
      function getter : Integer;
    published
      property Value : Integer read getter;
    end;

implementation
function Base.getter : Integer;
begin getter := 0;
end;

end.
```

The only solution to this problem is to declare the getter function to be of `register` calling convention, which is the default. As can be seen in this example, no calling convention is specified.

288. Property getters and setters cannot be overloaded

Complete list of compiler error messages

A property has specified an overloaded procedure as either its *getter* or *setter*.

```
unit Produce;
interface
  type
    Base = class
    public
      function getter : Integer; overload;
      function getter(a : char) : Integer; overload;
      property Value : Integer read getter;
    end;

implementation
function Base.getter : Integer;
begin getter := 0;
end;

function Base.getter(a : char) : Integer;
begin
end;

end.
```

The overloaded function `getter` in the above example will cause this error to be produced.

```
unit Solve;
interface
  type
    Base = class
    public
      function getter : Integer;
      property Value : Integer read getter;
    end;

implementation
function Base.getter : Integer;
begin getter := 0;
end;

end.
```

The only solution when this problem occurs is to remove the offending `overload` specifications, as is shown in the above example.

289. Comparing signed and unsigned types - widened both operands

Complete list of compiler error messages

To compare signed and unsigned types correctly the compiler must promote both operands to the next larger size data type.

To see why this is necessary, consider two operands, a `Shortint` with the value -128 and a `Byte` with the value 130. The `Byte` type has one more digit of precision than the `Shortint` type, and thus comparing the two values cannot accurately be performed in only 8 bits. The proper solution for the compiler is to promote both these types to a larger, common, size and then to perform the comparison.

```
program Produce;
  var
    s : shortint;
    b : byte;

begin
  s := -128;
  b := 130;

  assert(b < s);
end.
```

290. Combining signed and unsigned types - widened both operands

Complete list of compiler error messages

To mathematically combine signed and unsigned types correctly the compiler must promote both operands to the next larger size data type and then perform the combination.

To see why this is necessary, consider two operands, an `Integer` with the value -128 and a `Cardinal` with the value 130. The `Cardinal` type has one more digit of precision than the `Integer` type, and thus comparing the two values cannot accurately be performed in only 32 bits. The proper solution for the compiler is to promote both these types to a larger, common, size and then to perform the comparison.

The compiler will only produce this warning when the size is extended beyond what would normally be used for calculating the result.

```
{$APPTYPE CONSOLE}
program Produce;
  var
    i : Integer;
    c : Cardinal;

begin
  i := -128;
  c := 130;
  WriteLn(i + c);
end.
```

In the example above, the compiler warns that the expression will be calculated at 64 bits rather than the supposed 32 bits.

291. Duplicate <text> <name> with identical parameters will be inaccessible from C++

Complete list of compiler error messages

An object file is being generated and Two, differently named, constructors or destructors with identical parameter lists have been created; they will be inaccessible if the code is translated to an `HPP` file because constructor and destructor names are converted to the `class` name. In C++ these duplicate declarations will appear to be the same function.

```
unit Produce;
interface
  type
    Base = class
      constructor ctor0(a, b, c : integer);
      constructor ctor1(a, b, c : integer);
    end;

implementation
constructor Base.ctor0(a, b, c : integer);
begin
end;

constructor Base.ctor1(a, b, c : integer);
begin
end;

begin
end.
```

As can be seen in this example, the two constructors have the same signature and thus, when the file is compiled with one of the `-j` options, will produce this warning.

```
unit Solve;
interface
  type
    Base = class
      constructor ctor0(a, b, c : integer);
      constructor ctor1(a, b, c : integer; dummy : integer = 0);
    end;

implementation
constructor Base.ctor0(a, b, c : integer);
begin
end;

constructor Base.ctor1(a, b, c : integer; dummy : integer);
begin
end;

begin
end.
```

A simple method to solve this problem is to change the signature of one of constructors, for example, to add an extra parameter. In the example above, a default parameter has been added to `ctor1`. This method of approaching this error has the benefit that Pascal code using `ctor1` does not need to be changed. C++ code, on the other hand, will have to specify the extra parameter to allow the compiler to determine which constructor is desired.

292. Comparison always evaluates to False

Complete list of compiler error messages

The compiler has determined that the expression will always evaluate to `False`. This most often can be the result of a boundary test against a specific variable type, for example, a `Integer` against `$80000000`. In versions of the Pascal compiler prior to 12.0, the hexadecimal constant `$80000000` would have been a negative `Integer` value, but with the introduction of the `int64` type, this same constant now becomes a positive `int64` type. As a result, comparisons of this constant against `Integer` variables will no longer behave as they once did.

As this is a warning rather than an error, there is no standard method of addressing the problems: sometimes the warning can be ignored, sometimes the code must be rewritten.

```
program Produce;

  var
    i : Integer;
    c : Cardinal;

begin
  c := 0;
  i := 0;
  if c < 0 then
    WriteLn('false');

  if i >= $80000000 then
    WriteLn('false');
end.
```

Here the compiler determines that the two expressions will always be `False`. In the first case, a `Cardinal`, which is unsigned, can never be less than 0. In the second case, a 32-bit `Integer` value can never be larger than, or even equal to, an `int64` value of `$80000000`.

293. Comparison always evaluates to True

Complete list of compiler error messages

The compiler has determined that the expression will always evaluate to `True`. This most often can be the result of a boundary test against a specific variable type, for example, a `Integer` against `$80000000`.

In versions of the Pascal compiler prior to 12.0, the hexadecimal constant `$80000000` would have been a negative `Integer` value, but with the introduction of the `int64` type, this same constant now becomes a positive `int64` type. As a result, comparisons of this constant against `Integer` variables will no longer behave as they once did.

As this is a warning rather than an error, there is no standard method of addressing the problems: sometimes the warning can be ignored, sometimes the code must be rewritten.

```
program Produce;

  var
    i : Integer;
    c : Cardinal;

begin
  c := 0;
  i := 0;
  if c >= 0 then
    WriteLn('true');

  if i < $80000000 then
    WriteLn('true');
end.
```

Here the compiler determines that the two expressions will always be `True`. In the first case, a `Cardinal`, which is unsigned, will always be greater or equal to 0. In the second case, a 32-bit `Integer` value will always be smaller than an `int64` value of `$80000000`.

294. Cannot use reserved unit name <name>

Complete list of compiler error messages

An attempt has been made to use one of the reserved unit names, such as `System`, as the name of a user-created unit.

The names in the following list are currently reserved by the compiler.

- System
- SysInit

```
unit System;
interface
implementation
begin
end.
```

The name of the unit in this example is illegal because it is reserved for use by the compiler.

```
unit MySystem;
interface
implementation
begin
end.
```

The only solution to this problem is to use a different name for the unit.

295. No overloaded version of '<name>' with this parameter list exists

Complete list of compiler error messages

An attempt has been made to call an overloaded procedure but no suitable match could be found.

```
program overload;
  procedure f(x : Char); overload;
  begin
  end;

  procedure f(x : Integer); overload;
  begin
  end;

begin
  f(1.0);

end.
```

In the use of `f` presented here, the compiler is unable to find a suitable match (using the type compatibility & *overloading* rules) given the actual parameter `1.0`.

```
program overload;
  procedure f(x : char); overload;
  begin
  end;

  procedure f(x : integer); overload;
  begin
  end;

begin
  f(1);
end.
```

Here, the call to `f` has been changed to pass an integer as the actual parameter which will allow the compiler to find a suitable match. Another approach to solving this problem would be to introduce a new procedure which takes a floating point parameter.

296. property attribute 'label' cannot be used in dispinterface

Complete list of compiler error messages

You have added a label to a property defined in a dispinterface, but this is disallowed by the language definition.

```
program Problem;

  type
    T0 = dispinterface
      ['{15101510-1510-1510-1510-151015101510}']
      function R : Integer;
      property value : Integer label 'Key';
    end;

begin
end.
```

Here an attempt is made to use a `label` attribute on a `dispinterface` property.

```
program Solve;

  type
    T0 = dispinterface
      ['{15101510-1510-1510-1510-151015101510}']
      function R : Integer;
      property value : Integer;
    end;

begin
end.
```

the only solution to this problem is to remove `label` attribute from the property definition.

297. property attribute 'label' cannot be an empty string

Complete list of compiler error messages

```
unit Problem;
interface
  type
    T0 = class
      f : integer;
      property g : integer read f write f label '';
    end;

implementation
begin
end.
```

The error is output because the `label` attribute for `g` is an empty string.

```
unit Solve;
interface
  type
    T0 = class
      f : integer;
      property g : integer read f write f label 'LabelText';
    end;

implementation
begin
end.
```

In this solution, the `label` attribute has been replaced by a non-zero length string.