

```
{  
    FILE* sfile;  
    int count = 0;  
  
    sfile = fopen("file1",  
    "r");  
  
    if( sfile == NULL)  
    {  
        return -1;  
    }  
  
    while (1)  
    {  
        char c;  
        c = fgetc(  
        if(c == EOF)  
        {  
            break;  
        }  
        else  
        {  
            count++;  
        }  
    }  
  
    return count;  
}
```

# 68K/ColdFire v10.0

## C++ Compiler User's Manual

A publication of  
Altium BV  
Documentation Department  
Copyright © 2003 Altium BV

All rights reserved. Reproduction in whole or part is prohibited  
without the written consent of the copyright owner.

TASKING is a brand name of Altium Limited.

The following trademarks are acknowledged:

FLEXlm is a registered trademark of Globetrotter Software, Inc.  
IBM is a trademark of International Business Machines Corp.  
Motorola is a trademark of Motorola, Inc.  
MS-DOS and Windows are registered trademarks of Microsoft Corporation.  
SUN is a trademark of Sun Microsystems, Inc.  
UNIX is a registered trademark of X/Open Company, Ltd.

All other trademarks are property of their respective owners.

Data subject to alteration without notice.

<http://www.tasking.com>  
<http://www.altium.com>

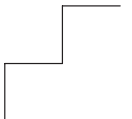
*The information in this document has been carefully reviewed and is believed to be accurate and reliable. However, Altium assumes no liabilities for inaccuracies in this document. Furthermore, the delivery of this information does not convey to the recipient any license to use or copy the software or documentation, except as provided in an executed license agreement covering the software and documentation.*

*Altium reserves the right to change specifications embodied in this document without prior notice.*

# CONTENTS

## TABLE OF CONTENTS

---



---

# CONTENTS

---

## **OVERVIEW** **1-1**

1.1	Introduction to C++ Compiler .....	1-3
1.2	Development Structure .....	1-3
1.2.1	The C++ Linker Driver (ldriver): Prelinker Phase .....	1-5
1.2.2	The C++ Linker Driver (ldriver): Muncher Phase .....	1-7

## **LANGUAGE IMPLEMENTATION** **2-1**

2.1	Introduction .....	2-3
2.2	C++ Language Extension Keywords .....	2-3
2.3	C++ Dialect Accepted .....	2-3
2.3.1	New Language Features Accepted .....	2-4
2.3.2	New Language Features Not Accepted .....	2-6
2.3.3	Anachronisms Accepted .....	2-7
2.3.4	Extensions Accepted in Normal C++ Mode .....	2-8
2.3.5	Extensions Accepted in Cfront 2.1 Compatibility Mode .....	2-9
2.3.6	Extensions Accepted in Cfront 2.1 and 3.0 Compatibility Mode .....	2-13
2.4	Namespace Support .....	2-20
2.5	Template Instantiation .....	2-22
2.5.1	Automatic Instantiation .....	2-23
2.5.2	Instantiation Modes .....	2-27
2.5.3	Instantiation #pragma Directives .....	2-28
2.5.4	Implicit Inclusion .....	2-30
2.6	Precompiled Headers .....	2-31
2.6.1	Automatic Precompiled Header Processing .....	2-31
2.6.2	Manual Precompiled Header Processing .....	2-35
2.6.3	Other Ways to Control Precompiled Headers .....	2-36
2.6.4	Performance Issues .....	2-36



<b>COMPILER USE</b>		<b>3-1</b>
3.1	Invocation .....	3-3
3.1.1	Detailed Description of the Compiler Options .....	3-12
3.2	Linker .....	3-80
3.3	Pragmas .....	3-81
<b>COMPILER DIAGNOSTICS</b>		<b>4-1</b>
4.1	Diagnostic Messages .....	4-3
4.2	Termination Messages .....	4-4
4.3	Response to Signals .....	4-5
4.4	Return Values .....	4-5
<b>ERROR MESSAGES</b>		<b>A-1</b>
1	Introduction .....	A-3
2	Messages .....	A-4
<b>INDEX</b>		

## **MANUAL PURPOSE AND STRUCTURE**

### **PURPOSE**

This manual is aimed at users of the TASKING 68K/ColdFire C++ Compiler. It assumes that you are conversant with the C and C++ language.

### **MANUAL STRUCTURE**

Related Publications

Conventions Used In This Manual

#### 1. Overview

Provides an overview of the TASKING 68K/ColdFire toolchain and gives you some familiarity with the different parts of it and their relationship. A sample session explains how to build an application from your C++ file.

#### 2. Language Implementation

Concentrates on the approach of the 68K/ColdFire architecture and describes the language implementation. The C++ language itself is not described in this document.

#### 3. Compiler Use

Deals with invocation, command line options and pragmas.

#### 4. Compiler Diagnostics

Describes the exit status and error/warning messages of the C++ compiler.

### **APPENDICES**

#### A. Error Messages

Contains an overview of the error messages.

### **INDEX**



## **RELATED PUBLICATIONS**

- The C++ Programming Language (second edition)  
by Bjarne Strastrup (1991, Addison Wesley)
- ISO/IEC 14882:1998 C++ standard [ANSI]  
More information on the standards can be found at  
<http://www.ansi.org>
- The Annotated C++ Reference Manual  
by Margaret A. Ellis and Bjarne Strastrup (1990, Addison Wesley)
- The C Programming Language (second edition)  
by B. Kernighan and D. Ritchie (1988, Prentice Hall)
- ANSI X3.159-1989 standard [ANSI]
- 68K/ColdFire C Compiler/Assembler User's Manual [TASKING,  
MA001-022-00-00]
- 68K/ColdFire C Compiler/Assembler Reference Manual [TASKING,  
MB001-022-00-00]
- 68K/ColdFire CrossView Pro Debugger User's Manual [TASKING,  
MA001-043-00-00]

## **CONVENTIONS USED IN THIS MANUAL**

The notation used to describe the format of call lines is given below:

{ }                      Items shown inside curly braces enclose a list from which you must choose an item.

[ ]                      Items shown inside square brackets enclose items that are optional.

|                        The vertical bar separates items in a list. It can be read as OR.

*italics*                Items shown in italic letters mean that you have to substitute the item. If italic items are inside square brackets, they are optional. For example:

*filename*

means: type the name of your file in place of the word *filename*.

...                      An ellipsis indicates that you can repeat the preceding item zero or more times.

screen font          Represents input examples and screen output examples.

**bold font**           Represents a command name, an option or a complete command line which you can enter.

### ***For example***

*command* [*option*]... *filename*

This line could be written in plain English as: execute the command *command* with the optional options *option* and with the file *filename*.

### ***Illustrations***

The following illustrations are used in this manual:



This is a note. It gives you extra information.



This is a warning. Read the information carefully.



This illustration indicates actions you can perform with the mouse.



This illustration indicates keyboard input.



This illustration can be read as “See also”. It contains a reference to another command, option or section.

# CHAPTER

# 1

## OVERVIEW

---



---

# 1

# CHAPTER

---

## **1.1 INTRODUCTION TO C++ COMPILER**

This manual provides a functional description of the TASKING C++ Compiler. This manual uses **cpxxx** (the name of the binary, where *xxx* is the name of the target) as a shorthand notation for "TASKING 68K/ColdFire C++ Compiler". See section *Derivatives Overview* in chapter *Tutorial* of the Getting Started Manual, for a list of all available targets. You should be familiar with the C++ language and with the ANSI/ISO C language.

The C++ compiler is part of a complete toolchain. For details about the C compiler see the "C Compiler/Assembler User's Manual".

The C++ compiler accepts the C++ language of the ISO/IEC 14882:1998 C++ standard, with some minor exceptions documented in the next chapter.

The C++ compiler provides complete error checking, produces clear error messages (including the position of the error within the source line), and avoids cascading errors. It also avoids seeming overly finicky to a knowledgeable C or C++ programmer.

## **1.2 DEVELOPMENT STRUCTURE**

The next figure explains the relationship between the different parts of the 68K/ColdFire toolchain:

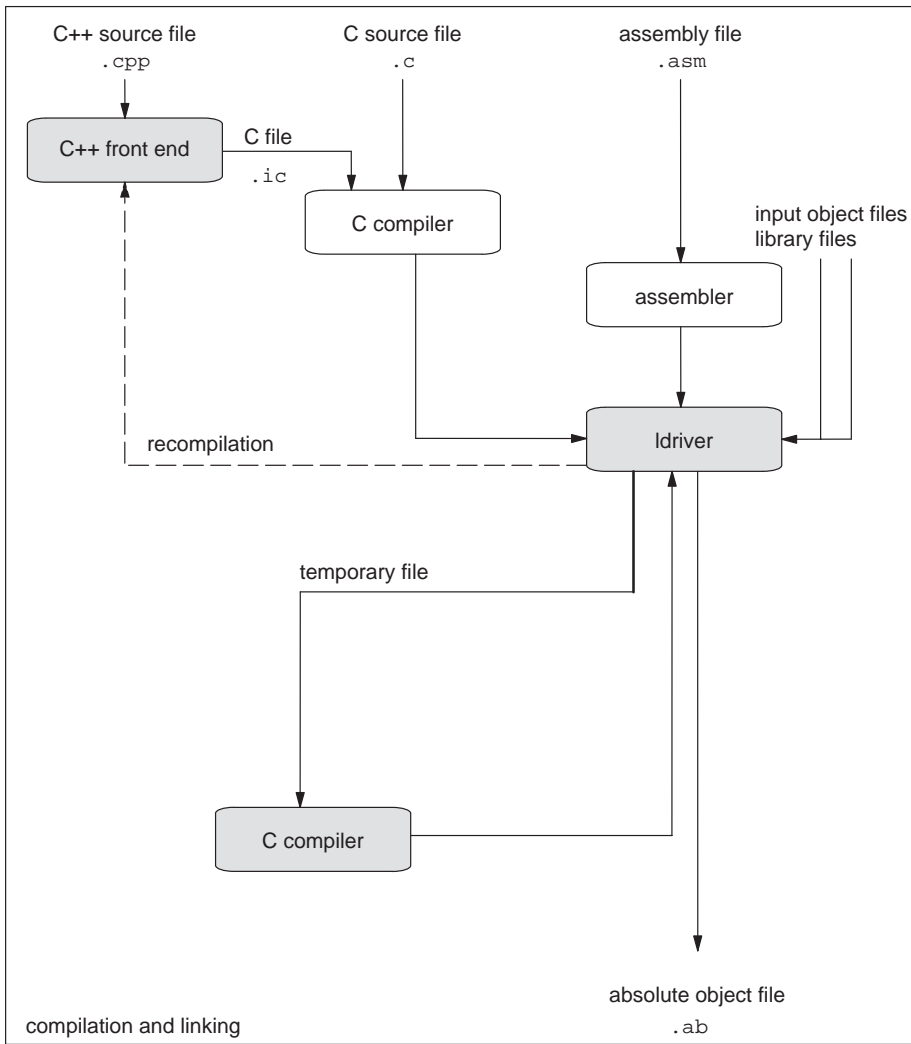


Figure 1-1: Development flow

### **1.2.1 THE C++ LINKER DRIVER (LDRIVER): PRELINKER PHASE**

The C++ compiler provides a complete implementation of an automatic instantiation mechanism. The automatic instantiation mechanism is a "linker feedback" mechanism. It works by providing additional information in the object file that is used by a "prelinker" to determine which template entities require instantiation so that the program can be linked successfully. Unlike most aspects of the C++ compiler the automatic instantiation mechanism is, by its nature, dependent on certain operating system and object file format properties. In particular, the prelinker is a separate program that accesses information about the symbols defined in object files.

At the end of each compilation, the C++ compiler determines whether any template entities were referenced in the translation unit. If so, an "instantiation information" file is created, referred to for convenience as a `.ii` file. If no template entities were referenced in the translation unit, the `.ii` file will not be created and any existing file will be removed. If an error occurs during compilation, the state of the `.ii` file is unchanged.

Once a complete set of object files has been generated, **ldriver** invokes the prelinker to determine whether any new instantiations are required or if any existing instantiations are no longer required. The command line arguments to the prelinker include a list of input files to be analyzed. The input files are the object files and libraries that constitute the application. The prelinker begins by looking for instantiation information files for each of the object files. If no instantiation information files are present, the prelinker concludes that no further action is required.

If there are instantiation information files, the prelinker reads the current instantiation list from each information file. The instantiation list contains the list of instantiations assigned to a given source file by a previous invocation of the prelinker. The prelinker produces a list of the global symbols that are referenced or defined by each of the input files. The prelinker then simulates a link operation to determine which symbols must be defined for the application to link successfully.



When the link simulation has been completed, the prelinker processes each input file to determine whether any new instantiations should be assigned to the input file or if any existing instantiations should be removed. The prelinker goes through the current instantiation list from the instantiation information file to determine whether any of the existing instantiations are no longer needed. An instantiation may be no longer needed because the template entity is no longer referenced by the program or because a user supplied specialization has been provided. If the instantiation is no longer needed, it is removed from the list (internally; the file will be updated later) and the file is flagged as requiring recompilation.

The prelinker then examines any symbols referenced by the input file. The responsibility for generating an instantiation of a given entity that has not already been defined is assigned to the first file that is capable of generating that instantiation.

Once all of the assignments have been updated, the prelinker once again goes through the list of object files. For each, if the corresponding instantiation information file must be updated, the new file is written. Only source files whose corresponding `.ii` file has been modified will be recompiled.

At this point each `.ii` file contains the information needed to recompile the source file and a list of instantiations assigned to the source file, in the form of mangled function and static data member names.

If an error occurs during a recompilation, the prelinker exits without updating the remaining information files and without attempting any additional compilations.

If all recompilations complete without error, the prelink process is repeated, since an instantiation can produce the demand for another instantiation. This prelink cycle (finding uninstantiated templates, updating the appropriate `.ii` files, and dispatching recompilations) continues until no further recompilations are required.

When the prelinker is finished, the linker is invoked. Note that simply because the prelinker completes successfully does not assure that the linker will not detect errors. Unresolvable template references and other linker errors will not be diagnosed by the prelinker.

### **1.2.2 THE C++ LINKER DRIVER (LDRIVER): MUNCHER PHASE**

The C++ muncher implements global initialization and termination code.

The muncher accepts the output of the linker as its input file. It generates a C program that defines a data structure containing a list of pointers to the initialization and termination routines. This generated program is then compiled and linked in with the executable. The data structure is consulted at run-time by startup code invoked from `_main`, and the routines on the list are invoked at the appropriate times.



# OVERVIEW

# CHAPTER

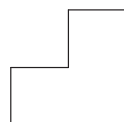
# 2

## LANGUAGE IMPLEMENTATION

---



TASKING



---

# 2

# CHAPTER

---

## 2.1 INTRODUCTION

The TASKING C++ compiler (**cpxxx**) offers a new approach to high-level language programming for the 68K/ColdFire family. The C++ compiler accepts the C++ language as defined by the ISO/IEC 14882:1998 standard, with the exceptions listed in section 2.3.

This chapter describes the C++ language extensions and some specific features.

## 2.2 C++ LANGUAGE EXTENSION KEYWORDS

The C++ compiler supports the same language extension keywords as the C compiler (as described in Appendix A of the *68K/ColdFire C Compiler/Assembler User's Manual*).

The following language extensions are supported:

```
#pragma separate
#pragma sep_on
#pragma sep_off
__ASMLINE
__ASM
__CASM
__IH
__SWI
SPL
GPL
TRAP
```

## 2.3 C++ DIALECT ACCEPTED

The C++ compiler accepts the C++ language as defined by the ISO/IEC 14882:1998 standard, with the exceptions listed below.

The C++ compiler also has a cfront compatibility mode, which duplicates a number of features and bugs of cfront 2.1 and 3.0.x. Complete compatibility is not guaranteed or intended; the mode is there to allow programmers who have unwittingly used cfront features to continue to compile their existing code. In particular, if a program gets an error when compiled by cfront, the C++ compiler may produce a different error or no error at all.

Command line options are also available to enable and disable anachronisms and strict standard-conformance checking.

### 2.3.1 NEW LANGUAGE FEATURES ACCEPTED

The following features not in traditional C++ (the C++ language of *"The Annotated C++ Reference Manual"* by Ellis and Stroustrup (ARM)) but in the standard are implemented:

- The dependent statement of an `if`, `while`, `do-while`, or `for` is considered to be a scope, and the restriction on having such a dependent statement be a declaration is removed.
- The expression tested in an `if`, `while`, `do-while`, or `for`, as the first operand of a `"?"` operator, or as an operand of the `"&&"`, `"::"`, or `"!"` operators may have a pointer-to-member type or a class type that can be converted to a pointer-to-member type in addition to the scalar cases permitted by the ARM.
- Qualified names are allowed in elaborated type specifiers.
- A global-scope qualifier is allowed in member references of the form `x.::A::B` and `p->::A::B`.
- The precedence of the third operand of the `"?"` operator is changed.
- If control reaches the end of the `main()` routine, and `main()` has an integral return type, it is treated as if a `return 0;` statement were executed.
- Pointers to arrays with unknown bounds as parameter types are diagnosed as errors.
- A functional-notation cast of the form `A()` can be used even if `A` is a class without a (nontrivial) constructor. The temporary created gets the same default initialization to zero as a static object of the class type.
- A cast can be used to select one out of a set of overloaded functions when taking the address of a function.
- Template friend declarations and definitions are permitted in class definitions and class template definitions.
- Type template parameters are permitted to have default arguments.
- Function templates may have nontype template parameters.
- A reference to `const volatile` cannot be bound to an rvalue.
- Qualification conversions, such as conversion from `T**` to `T const * const *` are allowed.

- Digraphs are recognized.
- Operator keywords (e.g., `not`, `and`, `bitand`, etc.) are recognized.
- Static data member declarations can be used to declare member constants.
- `wchar_t` is recognized as a keyword and a distinct type.
- `bool` is recognized.
- RTTI (run-time type identification), including `dynamic_cast` and the `typeid` operator, is implemented.
- Declarations in tested conditions (in `if`, `switch`, `for`, and `while` statements) are supported.
- Array `new` and `delete` are implemented.
- New-style casts (`static_cast`, `reinterpret_cast`, and `const_cast`) are implemented.
- Definition of a nested class outside its enclosing class is allowed.
- `mutable` is accepted on non-static data member declarations.
- Namespaces are implemented, including `using` declarations and directives. Access declarations are broadened to match the corresponding `using` declarations.
- Explicit instantiation of templates is implemented.
- The `typename` keyword is recognized.
- `explicit` is accepted to declare non-converting constructors.
- The scope of a variable declared in the `for-init-statement` of a `for` loop is the scope of the loop (not the surrounding scope).
- Member templates are implemented.
- The new specialization syntax (using “`template <>`”) is implemented.
- Cv-qualifiers are retained on rvalues (in particular, on function return values).
- The distinction between trivial and nontrivial constructors has been implemented, as has the distinction between PODs and non-PODs with trivial constructors.
- The linkage specification is treated as part of the function type (affecting function overloading and implicit conversions).
- `extern inline` functions are supported, and the default linkage for `inline` functions is external.
- A typedef name may be used in an explicit destructor call.
- Placement delete is implemented.



- An array allocated via a placement new can be deallocated via delete.
- Covariant return types on overriding virtual functions are supported.
- enum types are considered to be non-integral types.
- Partial specialization of class templates is implemented.
- Partial ordering of function templates is implemented.
- Function declarations that match a function template are regarded as independent functions, not as “guiding declarations” that are instances of the template.
- It is possible to overload operators using functions that take enum types and no class types.
- Explicit specification of function template arguments is supported.
- Unnamed template parameters are supported.
- The new lookup rules for member references of the form `x.A::B` and `p->A::B` are supported.
- The notation `:: template` (and `->template`, etc.) is supported.
- In a reference of the form `f()->g()`, with `g` a static member function, `f()` is evaluated. The ARM specifies that the left operand is not evaluated in such cases.

### **2.3.2 NEW LANGUAGE FEATURES NOT ACCEPTED**

The following features of the C++ standard are not implemented yet:

- enum types cannot contain values larger than can be contained in an int.
- `reinterpret_cast` does not allow casting a pointer to member of one class to a pointer to member of another class if the classes are unrelated.
- Two-phase name binding in templates, as described in [temp.res] and [temp.dep] of the standard, is not implemented.
- Class name injection is not implemented.
- Putting a `try/catch` around the initializers and body of a constructor is not implemented.
- Template template parameters are not implemented.
- Koenig lookup of function names on all calls is not implemented.
- Finding friend functions of the argument class types on name lookup on the function name in calls is not implemented.

- String literals do not have `const` type.
- Universal character set escapes (e.g., `\uabcd`) are not implemented.
- The `export` keyword for templates is not implemented.

### **2.3.3 ANACHRONISMS ACCEPTED**

The following anachronisms are accepted when anachronisms are enabled (with **`--anachronisms`**):

- `overload` is allowed in function declarations. It is accepted and ignored.
- Definitions are not required for static data members that can be initialized using default initialization. The anachronism does not apply to static data members of template classes; they must always be defined.
- The number of elements in an array may be specified in an array `delete` operation. The value is ignored.
- A single `operator++()` and `operator--()` function can be used to overload both prefix and postfix operations.
- The base class name may be omitted in a base class initializer if there is only one immediate base class.
- Assignment to `this` in constructors and destructors is allowed. This is allowed only if anachronisms are enabled and the "assignment to `this`" configuration parameter is enabled.
- A bound function pointer (a pointer to a member function for a given object) can be cast to a pointer to a function.
- A nested class name may be used as a non-nested class name provided no other class of that name has been declared. The anachronism is not applied to template classes.
- A reference to a non-`const` type may be initialized from a value of a different type. A temporary is created, it is initialized from the (converted) initial value, and the reference is set to the temporary.
- A reference to a non-`const` class type may be initialized from an rvalue of the class type or a derived class thereof. No (additional) temporary is used.
- A function with old-style parameter declarations is allowed and may participate in function overloading as though it were prototyped. Default argument promotion is not applied to parameter types of such functions when the check for compatibility is done, so that the following declares the overloading of two functions named `f`:

```
int f(int);
int f(x) char x; { return x; }
```

Note that in C this code is legal but has a different meaning: a tentative declaration of `f` is followed by its definition.

- When **--nonconst\_ref\_anachronism** is enabled, a reference to a non-const class can be bound to a class rvalue of the same type or a derived type thereof.

```
struct A {
    A(int);
    A operator=(A&);
    A operator+(const A&);
};
main () {
    A b(1);
    b = A(1) + A(2); // Allowed as anachronism
}
```

### 2.3.4 EXTENSIONS ACCEPTED IN NORMAL C++ MODE

The following extensions are accepted in all modes (except when strict ANSI violations are diagnosed as errors):

- A friend declaration for a class may omit the `class` keyword:

```
class A {
    friend B; // Should be "friend class B"
};
```

- Constants of scalar type may be defined within classes:

```
class A {
    const int size = 10;
    int a[size];
};
```

- In the declaration of a class member, a qualified name may be used:

```
struct A {
    int A::f(); // Should be int f();
};
```

- The preprocessing symbol `cplusplus` is defined in addition to the standard `__cplusplus`.
- A pointer to a constant type can be deleted.

- An assignment operator declared in a derived class with a parameter type matching one of its base classes is treated as a default assignment operator, that is, such a declaration blocks the implicit generation of a copy assignment operator. (This is cfront behavior that is known to be relied upon in at least one widely used library.) Here is an example:

```
struct A { };
struct B : public A {
    B& operator=(A&);
};
```

By default, as well as in cfront-compatibility mode, there will be no implicit declaration of `B::operator=(const B&)`, whereas in strict-ANSI mode `B::operator=(A&)` is not a copy assignment operator and `B::operator=(const B&)` is implicitly declared.

- Implicit type conversion between a pointer to an extern "C" function and a pointer to an extern "C++" function is permitted. Here's an example:

```
extern "C" void f(); // f's type has extern "C" linkage
void (*pf)()        // pf points to an extern "C++" function
    = &f;           // error unless implicit conversion is
                    // allowed
```

This extension is allowed in environments where C and C++ functions share the same calling conventions. It is enabled by default; it can also be enabled in cfront-compatibility mode or with option **--implicit\_extern\_c\_type\_conversion**. It is disabled in strict-ANSI mode.

### **2.3.5 EXTENSIONS ACCEPTED IN CFRONT 2.1 COMPATIBILITY MODE**

The following extensions are accepted in cfront 2.1 compatibility mode in addition to the extensions listed in the 2.1/3.0 section following (i.e., these are things that were corrected in the 3.0 release of cfront):

- The dependent statement of an `if`, `while`, `do-while`, or `for` is not considered to define a scope. The dependent statement may not be a declaration. Any objects constructed within the dependent statement are destroyed at exit from the dependent statement.
- Implicit conversion from integral types to enumeration types is allowed.

- A non-const member function may be called for a const object. A warning is issued.
- A const void \* value may be implicitly converted to a void \* value, e.g., when passed as an argument.
- When, in determining the level of argument match for overloading, a reference parameter is initialized from an argument that requires a non-class standard conversion, the conversion counts as a user-defined conversion.
- When a built-in operator is considered alongside overloaded operators in overload resolution, the match of an operand of a built-in type against the built-in type required by the built-in operator is considered a standard conversion in all cases (e.g., even when the type is exactly right without conversion).
- A reference to a non-const type may be initialized from a value that is a const-qualified version of the same type, but only if the value is the result of selecting a member from a const class object or a pointer to such an object.
- The cfront 2.1 "transitional model" for nested type support is simulated. In the transitional model a nested type is promoted to the file scope unless a type of the same name already exists at the file scope. It is an error to have two nested classes of the same name that need to be promoted to file scope or to define a type at file scope after the declaration of a nested class of the same name. This "feature" actually restricts the source language accepted by the compiler. This is necessary because of the effect this feature has on the name mangling of functions that use nested types in their signature. This feature does not apply to template classes.
- A cast to an array type is allowed; it is treated like a cast to a pointer to the array element type. A warning is issued.
- When an array is selected from a class, the type qualifiers on the class object (if any) are not preserved in the selected array. (In the normal mode, any type qualifiers on the object are preserved in the element type of the resultant array.)
- An identifier in a function is allowed to have the same name as a parameter of the function. A warning is issued.
- An expression of type void may be supplied on the return statement in a function with a void return type. A warning is issued.

- Cfront has a bug that causes a global identifier to be found when a member of a class or one of its base classes should actually be found. This bug is emulated in cfront compatibility mode. A warning is issued when, because of this feature, a nonstandard lookup is performed. The following conditions must be satisfied for the nonstandard lookup to be performed:
  - A member in a base class must have the same name as an identifier at the global scope. The member may be a function, static data member, or non-static data member. Member type names do not apply because a nested type will be promoted to the global scope by cfront which disallows a later declaration of a type with the same name at the global scope.
  - The declaration of the global scope name must occur between the declaration of the derived class and the declaration of an out-of-line constructor or destructor. The global scope name must be a type name.
  - No other member function definition, even one for an unrelated class, may appear between the destructor and the offending reference. This has the effect that the nonstandard lookup applies to only one class at any given point in time. For example:

```
struct B {
    void func(const char*);
};

struct D : public B {
public:
    D();
    void Init(const char* );
};

struct func {
    func( const char* msg);
};

D::D()

void D::Init(const char* t)
{
    //Should call B::func -- calls func::func instead.
    new func(t);
}
```

The global scope name must be present in a base class (`B::func` in this example) for the nonstandard lookup to occur. Even if the derived class were to have a member named `func`, it is still the presence of `B::func` that determines how the lookup will be performed.

- A parameter of type `"const void *"` is allowed on operator `delete`; it is treated as equivalent to `"void *"`.
- A period (`"."`) may be used for qualification where `"::"` should be used. Only `"::"` may be used as a global qualifier. Except for the global qualifier, the two kinds of qualifier operators may not be mixed in a given name (i.e., you may say `A::B::C` or `A.B.C` but not `A::B.C` or `A.B::C`). A period may not be used in a vacuous destructor reference nor in a qualifier that follows a template reference such as `A<T>::B`.
- Cfront 2.1 does not correctly look up names in friend functions that are inside class definitions. In this example function `f` should refer to the functions and variables (e.g., `f1` and `a1`) from the class declaration. Instead, the global definitions are used.

```
int a1;
int e1;
void f1();
class A {
    int a1;
    void f1();
    friend void f()
    {
        int i1 = a1; // cfront uses global a1
        f1(); // cfront uses global f1
    }
};
```

Only the innermost class scope is (incorrectly) skipped by cfront as illustrated in the following example.

```

int a1;
int b1;
struct A {
    static int a1;
    class B {
        static int b1;
        friend void f()
        {
            int i1 = a1; // cfront uses A::a1
            int j1 = b1; // cfront uses global b1
        }
    };
};

```

- `operator=` may be declared as a nonmember function. (This is flagged as an anachronism by cfront 2.1)
- A type qualifier is allowed (but ignored) on the declaration of a constructor or destructor. For example:

```

class A {
    A() const; // No error in cfront 2.1 mode
};

```

### **2.3.6 EXTENSIONS ACCEPTED IN CFONT 2.1 AND 3.0 COMPATIBILITY MODE**

The following extensions are accepted in both cfront 2.1 and cfront 3.0 compatibility mode (i.e., these are features or problems that exist in both cfront 2.1 and 3.0):

- Type qualifiers on the `this` parameter may to be dropped in contexts such as this example:

```

struct A {
    void f() const;
};
void (A::*fp)() = &A::f;

```

This is actually a safe operation. A pointer to a `const` function may be put into a pointer to `non-const`, because a call using the pointer is permitted to modify the object and the function pointed to will actually not modify the object. The opposite assignment would not be safe.

- Conversion operators specifying conversion to `void` are allowed.



- A nonstandard friend declaration may introduce a new type. A friend declaration that omits the elaborated type specifier is allowed in default mode, but in cfront mode the declaration is also allowed to introduce a new type name.

```
struct A {
    friend B;
};
```

- The third operand of the `?` operator is a conditional expression instead of an assignment expression as it is in the modern language.
- A reference to a pointer type may be initialized from a pointer value without use of a temporary even when the reference pointer type has additional type qualifiers above those present in the pointer value. For example,

```
int *p;
const int *&r = p; // No temporary used
```

- A reference may be initialized with a null.
- Because cfront does not check the accessibility of types, access errors for types are issued as warnings instead of errors.
- When matching arguments of an overloaded function, a `const` variable with value zero is not considered to be a null pointer constant. In general, in overload resolution a null pointer constant must be spelled `"0"` to be considered a null pointer constant (e.g., `'\0'` is not considered a null pointer constant).
- An alternate form of declaring pointer-to-member-function variables is supported, for example:

```
struct A {
    void f(int);
    static void sf(int);
    typedef void A::T3(int); // nonstd typedef decl
    typedef void T2(int);    // std typedef
};
typedef void A::T(int); // nonstd typedef decl
T* pmf = &A::f;        // nonstd ptr-to-member decl
A::T2* pf = A::sf;      // std ptr to static mem decl
A::T3* pmf2 = &A::f;    // nonstd ptr-to-member decl
```

where `T` is construed to name a routine type for a non-static member function of class `A` that takes an `int` argument and returns `void`; the use of such types is restricted to nonstandard pointer-to-member declarations. The declarations of `T` and `pmf` in combination are equivalent to a single standard pointer-to-member declaration:

```
void (A::* pmf)(int) = &A::f;
```

A nonstandard pointer-to-member declaration that appears outside of a class declaration, such as the declaration of `T`, is normally invalid and would cause an error to be issued. However, for declarations that appear within a class declaration, such as `A::T3`, this feature changes the meaning of a valid declaration. cfront version 2.1 accepts declarations, such as `T`, even when `A` is an incomplete type; so this case is also excepted.

- Protected member access checking is not done when the address of a protected member is taken. For example:

```
class B { protected: int i; };
class D : public B { void mf(); };
void D::mf() {
    int B::* pm1 = &B::i; // error, OK in cfront mode
    int D::* pm2 = &D::i; // OK
}
```



Protected member access checking for other operations (i.e., everything except taking a pointer-to-member address) is done in the normal manner.

- The destructor of a derived class may implicitly call the private destructor of a base class. In default mode this is an error but in cfront mode it is reduced to a warning. For example:

```
class A {
    ~A();
};
class B : public A {
    ~B();
};
B::~B(){} // Error except in cfront mode
```

- When disambiguation requires deciding whether something is a parameter declaration or an argument expression, the pattern *type-name-or-keyword(identifier...)* is treated as an argument. For example:

```
class A { A(); };
double d;
A x(int(d));
A(x2);
```

By default `int(d)` is interpreted as a parameter declaration (with redundant parentheses), and so `x` is a function; but in cfront-compatibility mode `int(d)` is an argument and `x` is a variable.

The declaration `A(x2);` is also misinterpreted by cfront. It should be interpreted as the declaration of an object named `x2`, but in cfront mode is interpreted as a function style cast of `x2` to the type `A`.

Similarly, the declaration

```
int xyz(int());
```

declares a function named `xyz`, that takes a parameter of type "function taking no arguments and returning an `int`". In cfront mode this is interpreted as a declaration of an object that is initialized with the value `int()` (which evaluates to zero).

- A named bit-field may have a size of zero. The declaration is treated as though no name had been declared.
- Plain bit fields (i.e., bit fields declared with a type of `int`) are always unsigned.
- The name given in an elaborated type specifier is permitted to be a typedef name that is the synonym for a class name, e.g.,

```
typedef class A T;
class T *pa;           // No error in cfront
mode
```

- No warning is issued on duplicate size and sign specifiers.
- ```
short short int i;    // No warning in cfront mode
```
- Virtual function table pointer update code is not generated in destructors for base classes of classes without virtual functions, even if the base class virtual functions might be overridden in a further-derived class. For example:

```

struct A {
    virtual void f() {}
    A() {}
    ~A() {}
};
struct B : public A {
    B() {}
    ~B() {f();}           // Should call A::f according to
                          // ARM 12.7
};
struct C : public B {
    void f() {}
} c;

```

In cfront compatibility mode, `B::~~B` calls `C::f`.

- An extra comma is allowed after the last argument in an argument list, as for example in

```
f(1, 2, );
```

- A constant pointer-to-member-function may be cast to a pointer-to-function. A warning is issued.

```

struct A {int f();};
main () {
    int (*p)();
    p = (int (*)( ))A::f;  // Okay, with warning
}

```

- Arguments of class types that allow bitwise copy construction but also have destructors are passed by value (i.e., like C structures), and the destructor is not called on the "copy". In normal mode, the class object is copied into a temporary, the address of the temporary is passed as the argument, and the destructor is called on the temporary after the call returns. Note that because the argument is passed differently (by value instead of by address), code like this compiled in cfront mode is not calling-sequence compatible with the same code compiled in normal mode. In practice, this is not much of a problem, since classes that allow bitwise copying usually do not have destructors.
- A union member may be declared to have the type of a class for which you have defined an assignment operator (as long as the class has no constructor or destructor). A warning is issued.

- When an unnamed class appears in a typedef declaration, the typedef name may appear as the class name in an elaborated type specifier.

```
typedef struct { int i, j; } S;
struct S x; // No error in cfront mode
```

- Two member functions may be declared with the same parameter types when one is static and the other is non-static with a function qualifier.

```
class A {
    void f(int) const;
    static void f(int); // No error in cfront mode
};
```

- The scope of a variable declared in the for-init-statement is the scope to which the for statement belongs.

```
int f(int i) {
    for (int j = 0; j < i; ++j) { /* ... */ }
    return j; // No error in cfront mode
}
```

- Function types differing only in that one is declared extern "C" and the other extern "C++" can be treated as identical:

```
typedef void (*PF)();
extern "C" typedef void (*PCF)();
void f(PF);
void f(PCF);
```

PF and PCF are considered identical and void f(PCF) is treated as a compatible redeclaration of f. (By contrast, in standard C++ PF and PCF are different and incompatible types — PF is a pointer to an extern "C++" function whereas PCF is a pointer to an extern "C" function — and the two declarations of f create an overload set.)

- Functions declared inline have internal linkage.
- enum types are regarded as integral types.
- An uninitialized const object of non-POD class type is allowed even if its default constructor is implicitly declared:

```
struct A { virtual void f(); int i; };
const A a;
```

- A function parameter type is allowed to involve a pointer or reference to array of unknown bounds.

- If the user declares an `operator=` function in a class, but not one that can serve as the default `operator=`, and bitwise assignment could be done on the class, a default `operator=` is not generated; only the user-written `operator=` functions are considered for assignments (and therefore bitwise assignment is not done).

## 2.4 NAMESPACE SUPPORT

Namespaces are enabled by default except in the cfront modes. You can use the command-line options **--namespaces** and **--no\_namespaces** to enable or disable the features.

Name lookup during template instantiations now does something that approximates the two-phase lookup rule of the standard. When a name is looked up as part of a template instantiation but is not found in the local context of the instantiation, it is looked up in a synthesized instantiation context. The C++ compiler follows the new instantiation lookup rules for namespaces as closely as possible in the absence of a complete implementation of the new template name binding rules. Here is an example:

```
namespace N {
    int g(int);
    int x = 0;
    template <class T> struct A {
        T f(T t) { return g(t); }
        T f() { return x; }
    };
}

namespace M {
    int x = 99;
    double g(double);
    N::A<int> ai;
    int i = ai.f(0);    // N::A<int>::f(int) calls
                       // N::g(int)
    int i2 = ai.f();    // N::A<int>::f() returns
                       // 0 (= N::x)
    N::A<double> ad;
    double d = ad.f(0);    // N::A<double>::f(double)
                           // calls M::g(double)
    double d2 = ad.f();    // N::A<double>::f() also
                           // returns 0 (= N::x)
}
```

The lookup of names in template instantiations does not conform to the rules in the standard in the following respects:

- Although only names from the template definition context are considered for names that are not functions, the lookup is not limited to those names visible at the point at which the template was defined.

- Functions from the context in which the template was referenced are considered for all function calls in the template. Functions from the referencing context should only be visible for dependent function calls.

The lookup rules for overloaded operators are implemented as specified by the standard, which means that the operator functions in the global scope overload with the operator functions declared extern inside a function, instead of being hidden by them. The old operator function lookup rules are used when namespaces are turned off. This means a program can have different behavior, depending on whether it is compiled with namespace support enabled or disabled:

```
struct A { };
A operator+(A, double);
void f() {
    A a1;
    A operator+(A, int);
    a1 + 1.0;    // calls operator+(A, double)
                // with namespaces enabled but
}              // otherwise calls operator+(A, int);
```



## 2.5 TEMPLATE INSTANTIATION

The C++ language includes the concept of *templates*. A template is a description of a class or function that is a model for a family of related classes or functions.<sup>1</sup> For example, one can write a template for a `Stack` class, and then use a stack of integers, a stack of floats, and a stack of some user-defined type. In the source, these might be written `Stack<int>`, `Stack<float>`, and `Stack<X>`. From a single source description of the template for a stack, the compiler can create *instantiations* of the template for each of the types required.

The instantiation of a class template is always done as soon as it is needed in a compilation. However, the instantiations of template functions, member functions of template classes, and static data members of template classes (hereafter referred to as template entities) are not necessarily done immediately, for several reasons:

- One would like to end up with only one copy of each instantiated entity across all the object files that make up a program. (This of course applies to entities with external linkage.)
- The language allows one to write a *specialization* of a template entity, i.e., a specific version to be used in place of a version generated from the template for a specific data type. (One could, for example, write a version of `Stack<int>`, or of just `Stack<int>::push`, that replaces the template-generated version; often, such a specialization provides a more efficient representation for a particular data type.) Since the compiler cannot know, when compiling a reference to a template entity, if a specialization for that entity will be provided in another compilation, it cannot do the instantiation automatically in any source file that references it.
- The language also dictates that template functions that are not referenced should not be compiled, that, in fact, such functions might contain semantic errors that would prevent them from being compiled. Therefore, a reference to a template class should not automatically instantiate all the member functions of that class.

(It should be noted that certain template entities are always instantiated when used, e.g., inline functions.)

<sup>1</sup> Since templates are descriptions of entities (typically, classes) that are parameterizable according to the types they operate upon, they are sometimes called **parameterized types**.

From these requirements, one can see that if the compiler is responsible for doing all the instantiations automatically, it can only do so on a program-wide basis. That is, the compiler cannot make decisions about instantiation of template entities until it has seen all the source files that make up a complete program.

This C++ compiler provides an instantiation mechanism that does automatic instantiation at link time. For cases where you want more explicit control over instantiation, the C++ compiler also provides instantiation modes and instantiation pragmas, which can be used to exert fine-grained control over the instantiation process.

### **2.5.1 AUTOMATIC INSTANTIATION**

The goal of an automatic instantiation mode is to provide painless instantiation. You should be able to compile source files to object code, then link them and run the resulting program, and never have to worry about how the necessary instantiations get done.

In practice, this is hard for a compiler to do, and different compilers use different automatic instantiation schemes with different strengths and weaknesses:

- AT&T/USL/Novell's *cfront* product saves information about each file it compiles in a special directory called `ptrepository`. It instantiates nothing during normal compilations. At link time, it looks for entities that are referenced but not defined, and whose mangled names indicate that they are template entities. For each such entity, it consults the `ptrepository` information to find the file containing the source for the entity, and it does a compilation of the source to generate an object file containing object code for that entity. This object code for instantiated objects is then combined with the "normal" object code in the link step.

If you are using *cfront* you must follow a particular coding convention: all templates must be declared in `.h` files, and for each such file there must be a corresponding `.cc` file containing the associated definitions. The compiler is never told about the `.cc` files explicitly; one does not, for example, compile them in the normal way. The link step looks for them when and if it needs them, and does so by taking the `.h` filename and replacing its suffix.<sup>2</sup>

This scheme has the disadvantage that it does a separate compilation for each instantiated function (or, at best, one compilation for all the member functions of one class). Even though the function itself is often quite small, it must be compiled along with the declarations for the types on which the instantiation is based, and those declarations can easily run into many thousands of lines. For large systems, these compilations can take a very long time. The link step tries to be smart about recompiling instantiations only when necessary, but because it keeps no fine-grained dependency information, it is often forced to "recompile the world" for a minor change in a `.h` file. In addition, *cfront* has no way of ensuring that preprocessing symbols are set correctly when it does these instantiation compilations, if preprocessing symbols are set other than on the command line.

- Borland's C++ compiler instantiates everything referenced in a compilation, then uses a special linker to remove duplicate definitions of instantiated functions.

If you are using Borland's compiler you must make sure that every compilation sees all the source code it needs to instantiate all the template entities referenced in that compilation. That is, one cannot refer to a template entity in a source file if a definition for that entity is not included by that source file. In practice, this means that either all the definition code is put directly in the `.h` files, or that each `.h` file includes an associated `.cc` (actually, `.cpp`) file.

This scheme is straightforward, and works well for small programs. For large systems, however, it tends to produce very large object files, because each object file must contain object code (and symbolic debugging information) for each template entity it references.

<sup>2</sup> The actual implementation allows for several different suffixes and provides a command-line option to change the suffixes sought.

Our approach is a little different. It requires that, for each instantiation required, there is some (normal, top-level, explicitly-compiled) source file that contains the definition of the template entity, a reference that causes the instantiation, and the declarations of any types required for the instantiation.<sup>3</sup> This requirement can be met in various ways:

- The Borland convention: each `.h` file that declares a template entity also contains either the definition of the entity or includes another file containing the definition.
- Implicit inclusion: when the compiler sees a template declaration in a `.h` file and discovers a need to instantiate that entity, it is given permission to go off looking for an associated definition file having the same base name and a different suffix, and it implicitly includes that file at the end of the compilation. This method allows most programs written using the *cfront* convention to be compiled with our approach. See the section on implicit inclusion.
- The ad hoc approach: you make sure that the files that define template entities also have the definitions of all the available types, and add code or pragmas in those files to request instantiation of the entities there.

Our compiler's automatic instantiation method works as follows:

1. The first time the source files of a program are compiled, no template entities are instantiated. However, the generated object files contain information about things that *could* have been instantiated in each compilation. For any source file that makes use of a template instantiation an associated `.ii` file is created if one does not already exist (e.g., the compilation of `abc.cc` would result in the creation of `abc.ii`).
2. When the object files are linked together, the linker examines the object files, looking for references and definitions of template entities, and for the added information about entities that could be instantiated.

<sup>3</sup> Isn't this always the case? No. Suppose that file A contains a definition of class X and a reference to `Stack<X>::push`, and that file B contains the definition for the member function `push`. There would be no file containing both the definition of `push` and the definition of X.

3. If the linker finds a reference to a template entity for which there is no definition anywhere in the set of object files, it looks for a file that indicates that it could instantiate that template entity. When it finds such a file, it assigns the instantiation to it. The set of instantiations assigned to a given file is recorded in the associated instantiation request file (with, by default, a `.ii` suffix).
4. The linker then executes the compiler again to recompile each file for which the `.ii` file was changed.
5. When the compiler compiles a file, it reads the `.ii` file for that file and obeys the instantiation requests therein. It produces a new object file containing the requested template entities (and all the other things that were already in the object file).
6. The linker repeats steps 3–5 until there are no more instantiations to be adjusted.
7. The object files are linked together.

Once the program has been linked correctly, the `.ii` files contain a complete set of instantiation assignments. From then on, whenever source files are recompiled, the compiler will consult the `.ii` files and do the indicated instantiations as it does the normal compilations. That means that, except in cases where the set of required instantiations changes, the link step from then on will find that all the necessary instantiations are present in the object files and no instantiation assignment adjustments need be done. That's true even if the entire program is recompiled.

If you provide a specialization of a template entity somewhere in the program, the specialization will be seen as a definition by the linker. Since that definition satisfies whatever references there might be to that entity, the linker will see no need to request an instantiation of the entity. If you add a specialization to a program that has previously been compiled, the linker will notice that too and remove the assignment of the instantiation from the proper `.ii` file.

The `.ii` files should not, in general, require any manual intervention. One exception: if a definition is changed in such a way that some instantiation no longer compiles (it gets errors), and at the same time a specialization is added in another file, and the first file is being recompiled before the specialization file and is getting errors, the `.ii` file for the file getting the errors must be deleted manually to allow the prelinker to regenerate it.

The linker will issue messages like:

```
C++ prelinker: T1 Mark<T1>::func(T1) [with T1=int]
      assigned to file m.o1
C++ prelinker: executing: cp68332 m.cpp
```

The automatic instantiation scheme can coexist with partial explicit control of instantiation by you through the use of pragmas or command-line specification of the instantiation mode. See the following sections.

The automatic instantiation mode is enabled by default. It can be turned off by the command-line option **--no\_auto\_instantiation**. If automatic instantiation is turned off, the extra information about template entities that could be instantiated in a file is not put into the object file.

## 2.5.2 INSTANTIATION MODES

Normally, when a file is compiled, no template entities are instantiated (except those assigned to the file by automatic instantiation). The overall instantiation mode can, however, be changed by a command line option:

### **--instantiate none**

Do not automatically create instantiations of any template entities. This is the default. It is also the usually appropriate mode when automatic instantiation is done.

### **--instantiate used**

Instantiate those template entities that were used in the compilation. This will include all static data members for which there are template definitions.

### **--instantiate all**

Instantiate all template entities declared or referenced in the compilation unit. For each fully instantiated template class, all of its member functions and static data members will be instantiated whether or not they were used. Non-member template functions will be instantiated even if the only reference was a declaration.

**--instantiate local**

Similar to **--instantiate used** except that the functions are given internal linkage. This is intended to provide a very simple mechanism for those getting started with templates. The compiler will instantiate the functions that are used in each compilation unit as local functions, and the program will link and run correctly (barring problems due to multiple copies of local static variables.) However, one may end up with many copies of the instantiated functions, so this is not suitable for production use. **--instantiate local** can not be used in conjunction with automatic template instantiation. If automatic instantiation **--instantiate local** option. If automatic instantiation is not enabled by default, use of **--instantiate local** and **--auto\_instantiation** is an error.

In the case where the **cpxxx** command is given a single file to compile and link, e.g.,

```
cpxxx test.cc
```

the compiler knows that all instantiations will have to be done in the single source file. Therefore, it uses the **--instantiate used** mode and suppresses automatic instantiation.

### **2.5.3 INSTANTIATION #PRAGMA DIRECTIVES**

Instantiation pragmas can be used to control the instantiation of specific template entities or sets of template entities. There are three instantiation pragmas:

- The **instantiate** pragma causes a specified entity to be instantiated.
- The **do\_not\_instantiate** pragma suppresses the instantiation of a specified entity. It is typically used to suppress the instantiation of an entity for which a specific definition will be supplied.
- The **can\_instantiate** pragma indicates that a specified entity can be instantiated in the current compilation, but need not be; it is used in conjunction with automatic instantiation, to indicate potential sites for instantiation if the template entity turns out to be required.

The argument to the instantiation pragma may be:

|                              |                           |
|------------------------------|---------------------------|
| a template class name        | <b>A&lt;int&gt;</b>       |
| a template class declaration | <b>class A&lt;int&gt;</b> |

|                                 |                                              |
|---------------------------------|----------------------------------------------|
| a member function name          | <code>A&lt;int&gt;::f</code>                 |
| a static data member name       | <code>A&lt;int&gt;::i</code>                 |
| a static data declaration       | <code>int A&lt;int&gt;::i</code>             |
| a member function declaration   | <code>void A&lt;int&gt;::f(int, char)</code> |
| a template function declaration | <code>char* f(int, float)</code>             |

A pragma in which the argument is a template class name (e.g., `A<int>` or `class A<int>`) is equivalent to repeating the pragma for each member function and static data member declared in the class. When instantiating an entire class a given member function or static data member may be excluded using the **do\_not\_instantiate** pragma. For example,

```
#pragma instantiate A<int>
#pragma do_not_instantiate A<int>::f
```

The template definition of a template entity must be present in the compilation for an instantiation to occur. If an instantiation is explicitly requested by use of the **instantiate** pragma and no template definition is available or a specific definition is provided, an error is issued.

```
template <class T> void f1(T); // No body provided
template <class T> void g1(T); // No body provided

void f1(int) {} // Specific definition
void main()
{
    int i;
    double d;
    f1(i);
    f1(d);
    g1(i);
    g1(d);
}

#pragma instantiate void f1(int) // error - specific
                                // definition
#pragma instantiate void g1(int) // error - no body
                                // provided
```

`f1(double)` and `g1(double)` will not be instantiated (because no bodies were supplied) but no errors will be produced during the compilation (if no bodies are supplied at link time, a linker error will be produced).



A member function name (e.g., `A<int>::f`) can only be used as a pragma argument if it refers to a single user defined member function (i.e., not an overloaded function). Compiler-generated functions are not considered, so a name may refer to a user defined constructor even if a compiler-generated copy constructor of the same name exists. Overloaded member functions can be instantiated by providing the complete member function declaration, as in

```
#pragma instantiate char* A<int>::f(int, char*)
```

The argument to an instantiation pragma may not be a compiler-generated function, an inline function, or a pure virtual function.

## 2.5.4 IMPLICIT INCLUSION

When implicit inclusion is enabled, the C++ compiler is given permission to assume that if it needs a definition to instantiate a template entity declared in a `.h` file it can implicitly include the corresponding `.cc` file to get the source code for the definition. For example, if a template entity `ABC::f` is declared in file `xyz.h`, and an instantiation of `ABC::f` is required in a compilation but no definition of `ABC::f` appears in the source code processed by the compilation, the compiler will look to see if a file `xyz.cc` exists, and if so it will process it as if it were included at the end of the main source file.

To find the template definition file for a given template entity the C++ compiler needs to know the full path name of the file in which the template was declared and whether the file was included using the system include syntax (e.g., `#include <file.h>`). This information is not available for preprocessed source containing `#line` directives. Consequently, the C++ compiler will not attempt implicit inclusion for source code containing `#line` directives.

By default, the list of definition-file suffixes tried is `.cc`, `.cpp`, and `.cxx`.

Implicit inclusion works well alongside automatic instantiation, but the two are independent. They can be enabled or disabled independently, and implicit inclusion is still useful when automatic instantiation is not done.

The implicit inclusion mode can be turned on by the command-line option **--implicit\_include**.

## 2.6 PRECOMPILED HEADERS

It is often desirable to avoid recompiling a set of header files, especially when they introduce many lines of code and the primary source files that `#include` them are relatively small. The C++ compiler provides a mechanism for, in effect, taking a snapshot of the state of the compilation at a particular point and writing it to a disk file before completing the compilation; then, when recompiling the same source file or compiling another file with the same set of header files, it can recognize the "snapshot point", verify that the corresponding precompiled header (PCH) file is reusable, and read it back in. Under the right circumstances, this can produce a dramatic improvement in compilation time; the trade-off is that PCH files can take a lot of disk space.

### 2.6.1 AUTOMATIC PRECOMPILED HEADER PROCESSING

When `--pch` appears on the command line, automatic precompiled header processing is enabled. This means the C++ compiler will automatically look for a qualifying precompiled header file to read in and/or will create one for use on a subsequent compilation.

The PCH file will contain a snapshot of all the code preceding the "header stop" point. The header stop point is typically the first token in the primary source file that does not belong to a preprocessing directive, but it can also be specified directly by `#pragma hdrstop` (see below) if that comes first. For example:

```
#include "xxx.h"
#include "yyy.h"
int i;
```

The header stop point is `int` (the first non-preprocessor token) and the PCH file will contain a snapshot reflecting the inclusion of `xxx.h` and `yyy.h`. If the first non-preprocessor token or the `#pragma hdrstop` appears within a `#if` block, the header stop point is the outermost enclosing `#if`. To illustrate, heres a more complicated example:

```
#include "xxx.h"
#ifndef YYY_H
#define YYY_H 1
#include "yyy.h"
#endif
#if TEST
int i;
#endif
```

Here, the first token that does not belong to a preprocessing directive is again `int`, but the header stop point is the start of the `#if` block containing it. The PCH file will reflect the inclusion of `xxx.h` and conditionally the definition of `YYY_H` and inclusion of `yyy.h`; it will not contain the state produced by `#if TEST`.

A PCH file will be produced only if the header stop point and the code preceding it (mainly, the header files themselves) meet certain requirements:

- The header stop point must appear at file scope — it may not be within an unclosed scope established by a header file. For example, a PCH file will not be created in this case:

```
// xxx.h
class A {

// xxx.C
#include "xxx.h"
int i; };
```

- The header stop point may not be inside a declaration started within a header file, nor (in C++) may it be part of a declaration list of a linkage specification. For example, in the following case the header stop point is `int`, but since it is not the start of a new declaration, no PCH file will be created:

```
// yyy.h
static

// yyy.C
#include "yyy.h"
int i;
```

- Similarly, the header stop point may not be inside a `#if` block or a `#define` started within a header file.

- The processing preceding the header stop must not have produced any errors. (Note: warnings and other diagnostics will not be reproduced when the PCH file is reused.)
- No references to predefined macros `__DATE__` or `__TIME__` may have appeared.
- No use of the `#line` preprocessing directive may have appeared.
- **`#pragma no_pch`** (see below) must not have appeared.
- The code preceding the header stop point must have introduced a sufficient number of declarations to justify the overhead associated with precompiled headers. The minimum number of declarations required is 1.

When the host system does not support memory mapping, so that everything to be saved in the precompiled header file is assigned to preallocated memory (MS-Windows), two additional restrictions apply:

- The total memory needed at the header stop point cannot exceed the size of the block of preallocated memory.
- No single program entity saved can exceed 16384, the preallocation unit.

When a precompiled header file is produced, it contains, in addition to the snapshot of the compiler state, some information that can be checked to determine under what circumstances it can be reused. This includes:

- The compiler version, including the date and time the compiler was built.
- The current directory (i.e., the directory in which the compilation is occurring).
- The command line options.
- The initial sequence of preprocessing directives from the primary source file, including `#include` directives.
- The date and time of the header files specified in `#include` directives.

This information comprises the PCH prefix. The prefix information of a given source file can be compared to the prefix information of a PCH file to determine whether the latter is applicable to the current compilation.

As an illustration, consider two source files:

```
// a.cc
#include "xxx.h"
...           // Start of code
// b.cc
#include "xxx.h"
...           // Start of code
```

When `a.cc` is compiled with `--pch`, a precompiled header file named `a.pch` is created. Then, when `b.cc` is compiled (or when `a.cc` is recompiled), the prefix section of `a.pch` is read in for comparison with the current source file. If the command line options are identical, if `xxx.h` has not been modified, and so forth, then, instead of opening `xxx.h` and processing it line by line, the C++ compiler reads in the rest of `a.pch` and thereby establishes the state for the rest of the compilation.

It may be that more than one PCH file is applicable to a given compilation. If so, the largest (i.e., the one representing the most preprocessing directives from the primary source file) is used. For instance, consider a primary source file that begins with

```
#include "xxx.h"
#include "yyy.h"
#include "zzz.h"
```

If there is one PCH file for `xxx.h` and a second for `xxx.h` *and* `yyy.h`, the latter will be selected (assuming both are applicable to the current compilation). Moreover, after the PCH file for the first two headers is read in and the third is compiled, a new PCH file for all three headers may be created.

When a precompiled header file is created, it takes the name of the primary source file, with the suffix replaced by an implementation-specified suffix (`pch` by default). Unless `--pch_dir` is specified (see below), it is created in the directory of the primary source file.

When a precompiled header file is created or used, a message such as

```
"test.cc": creating precompiled header file "test.pch"
```

is issued. The user may suppress the message by using the command-line option **--no\_pch\_messages**.

In automatic mode (i.e., when **--pch** is used) the C++ compiler will deem a precompiled header file obsolete and delete it under the following circumstances:

- if the precompiled header file is based on at least one out-of-date header file but is otherwise applicable for the current compilation; or
- if the precompiled header file has the same base name as the source file being compiled (e.g., `xxx.pch` and `xxx.cc`) but is not applicable for the current compilation (e.g., because of different command-line options).

This handles some common cases; other PCH file clean-up must be dealt with by other means (e.g., by the user).

Support for precompiled header processing is not available when multiple source files are specified in a single compilation: an error will be issued and the compilation aborted if the command line includes a request for precompiled header processing and specifies more than one primary source file.

## 2.6.2 **MANUAL PRECOMPILED HEADER PROCESSING**

Command-line option **--create\_pch** *file-name* specifies that a precompiled header file of the specified name should be created.

Command-line option **--use\_pch** *file-name* specifies that the indicated precompiled header file should be used for this compilation; if it is invalid (i.e., if its prefix does not match the prefix for the current primary source file), a warning will be issued and the PCH file will not be used.

When either of these options is used in conjunction with **--pch\_dir**, the indicated file name (which may be a path name) is tacked on to the directory name, unless the file name is an absolute path name.

The **--create\_pch**, **--use\_pch**, and **--pch** options may not be used together. If more than one of these options is specified, only the last one will apply. Nevertheless, most of the description of automatic PCH processing applies to one or the other of these modes — header stop points are determined the same way, PCH file applicability is determined the same way, and so forth.

### 2.6.3 OTHER WAYS TO CONTROL PRECOMPILED HEADERS

There are several ways in which the user can control and/or tune how precompiled headers are created and used.

- **#pragma hdrstop** may be inserted in the primary source file at a point prior to the first token that does not belong to a preprocessing directive. It enables you to specify where the set of header files subject to precompilation ends. For example,

```
#include "xxx.h"
#include "yyy.h"
#pragma hdrstop
#include "zzz.h"
```

Here, the precompiled header file will include processing state for `xxx.h` and `yyy.h` but not `zzz.h`. (This is useful if the user decides that the information added by what follows the **#pragma hdrstop** does not justify the creation of another PCH file.)

- **#pragma no\_pch** may be used to suppress precompiled header processing for a given source file.
- Command-line option **--pch\_dir** *directory-name* is used to specify the directory in which to search for and/or create a PCH file.

Moreover, when the host system does not support memory mapping and preallocated memory is used instead, then one of the command-line options **--pch**, **--create\_pch**, or **--use\_pch**, if it appears at all, must be the *first* option on the command line.

### 2.6.4 PERFORMANCE ISSUES

The relative overhead incurred in writing out and reading back in a precompiled header file is quite small for reasonably large header files.

In general, it does not cost much to write a precompiled header file out even if it does not end up being used, and if it *is* used it almost always produces a significant speedup in compilation. The problem is that the precompiled header files can be quite large (from a minimum of about 250K bytes to several megabytes or more), and so one probably does not want many of them sitting around.

Thus, despite the faster recompilations, precompiled header processing is not likely to be justified for an arbitrary set of files with nonuniform initial sequences of preprocessing directives. Rather, the greatest benefit occurs when a number of source files can share the same PCH file. The more sharing, the less disk space is consumed. With sharing, the disadvantage of large precompiled header files can be minimized, without giving up the advantage of a significant speedup in compilation times.

Consequently, to take full advantage of header file precompilation, users should expect to reorder the `#include` sections of their source files and/or to group `#include` directives within a commonly used header file.

Below is an example of how this can be done. A common idiom is this:

```
#include "comnfile.h"
#pragma hdrstop
#include ...
```

where `comnfile.h` pulls in, directly and indirectly, a few dozen header files; the `#pragma hdrstop` is inserted to get better sharing with fewer PCH files. The PCH file produced for `comnfile.h` can be a bit over a megabyte in size. Another idiom, used by the source files involved in declaration processing, is this:

```
#include "comnfile.h"
#include "decl_hdrs.h"
#pragma hdrstop
#include ...
```

`decl_hdrs.h` pulls in another dozen header files, and a second, somewhat larger, PCH file is created. In all, the source files of a particular program can share just a few precompiled header files. If disk space were at a premium, you could decide to make `comnfile.h` pull in *all* the header files used — then, a single PCH file could be used in building the program.

Different environments and different projects will have different needs, but in general, users should be aware that making the best use of the precompiled header support will require some experimentation and probably some minor changes to source code.



# LANGUAGE

# CHAPTER

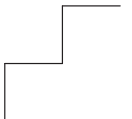
# 3

## COMPILER USE

---



TASKING



---

# 3

# CHAPTER

---

### 3.1 INVOCATION

The invocation syntax of the C++ compiler is:

**cpxxx** *file* [*options*]

where **cpxxx** represents the possible 68K/ColdFire targets. For a list of all possible targets see section *Derivatives Overview* in chapter *Tutorial* of the Getting Started Manual.



When you use a **UNIX** shell (Bourne shell, C-shell), arguments containing special characters (such as '(' and '?') must be enclosed with " " or escaped. The **-?** option (in the C-shell) becomes: **"-?"** or **-\\?**.

The C++ compiler accepts a C++ source file name and command line options in random order. A C++ source file must have a .cc, .cxx or .cpp suffix.

A keyword option specification consists of two hyphens followed by the option keyword (e.g., **--strict**). If an option requires an argument, the argument may be separated from the keyword by white space, or the keyword may be immediately followed by **=option**. When the second form is used there may not be any white space on either side of the equals sign.

The priority of the options is left-to-right: when two options conflict, the first (most left) one takes effect. The **-D** and **-U** options are not considered conflicting options, so they are processed left-to-right for each source file. You can overrule the default output file name with the **--gen\_c\_file\_name** option.

A summary of the options is given below. The next section describes the options in more detail.



You can use C Compiler options where applicable.

| Option                                                            | Description                                         |
|-------------------------------------------------------------------|-----------------------------------------------------|
| <b>--alternative_tokens</b><br><b>--no_alternative_tokens</b>     | Enable or disable recognition of alternative tokens |
| <b>--anachronisms</b><br><b>--no_anachronisms</b>                 | Enable or disable anachronisms                      |
| <b>--array_new_and_delete</b><br><b>--no_array_new_and_delete</b> | Enable or disable support for array new and delete  |

| Option                                                                                                                                                                                                                 | Description                                                                                              |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <b>--auto_instantiation</b><br><b>--no_auto_instantiation</b>                                                                                                                                                          | Enable or disable automatic instantiation of templates                                                   |
| <b>--bool</b><br><b>--no_bool</b>                                                                                                                                                                                      | Enable or disable recognition of <code>bool</code>                                                       |
| <b>--brief_diagnostics</b><br><b>--no_brief_diagnostics</b>                                                                                                                                                            | Enable or disable a shorter form of diagnostic output                                                    |
| <b>--cfront_2.1</b>                                                                                                                                                                                                    | Compile C++ compatible with <code>cfront</code> version 2.1                                              |
| <b>--cfront_3.0</b>                                                                                                                                                                                                    | Compile C++ compatible with <code>cfront</code> version 3.0                                              |
| <b>--comments</b>                                                                                                                                                                                                      | Keep comments in the preprocessed output                                                                 |
| <b>--create_pch</b> <i>file</i>                                                                                                                                                                                        | Create a precompiled header file with the specified name                                                 |
| <b>--define_macro</b> <i>macro</i> [= <i>def</i> ]<br><b>-D</b> <i>macro</i> [= <i>def</i> ]                                                                                                                           | Define preprocessor <i>macro</i>                                                                         |
| <b>--dependencies</b><br><b>-M</b>                                                                                                                                                                                     | Preprocess only. Emit dependencies for make                                                              |
| <b>--diag_suppress</b> <i>tag</i> [, <i>tag</i> ]...<br><b>--diag_remark</b> <i>tag</i> [, <i>tag</i> ]...<br><b>--diag_warning</b> <i>tag</i> [, <i>tag</i> ]...<br><b>--diag_error</b> <i>tag</i> [, <i>tag</i> ]... | Override normal error severity                                                                           |
| <b>--display_error_number</b>                                                                                                                                                                                          | Display error number in diagnostic messages                                                              |
| <b>--distinct_template_signatures</b><br><b>--no_distinct_template_signatures</b>                                                                                                                                      | Disallow or allow normal functions as template instantiation                                             |
| <b>--embedded_c++</b>                                                                                                                                                                                                  | Enable the diagnostics of noncompliance with the "Embedded C++" subset                                   |
| <b>--enum1s</b><br><b>--enum1u</b><br><b>--enum2s</b><br><b>--enum4s</b>                                                                                                                                               | Treat <code>enum</code> as signed char, unsigned char, signed 16-bit int (default), or signed 32-bit int |
| <b>--enum_overloading</b><br><b>--no_enum_overloading</b>                                                                                                                                                              | Enable or disable operator functions to overload builtin operators on <code>enum</code> -typed operands  |

| Option                                        | Description                                                                                                                     |
|-----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>--error_limit</b> <i>number</i>            | Specify maximum <i>number</i> of errors                                                                                         |
| <b>--error_output</b> <i>efile</i>            | Send diagnostics to error list file                                                                                             |
| <b>--exceptions</b>                           |                                                                                                                                 |
| <b>--no_exceptions</b>                        |                                                                                                                                 |
| <b>-x</b>                                     | Enable or disable support for exception handling                                                                                |
| <b>--explicit</b>                             |                                                                                                                                 |
| <b>--no_explicit</b>                          | Enable or disable support for the <code>explicit</code> specifier on constructor declarations                                   |
| <b>--extern_inline</b>                        |                                                                                                                                 |
| <b>--no_extern_inline</b>                     | Enable or disable inline function with external C++ linkage                                                                     |
| <b>--force_vtbl</b>                           | Force definition of virtual function tables                                                                                     |
| <b>--for_init_diff_warning</b>                |                                                                                                                                 |
| <b>--no_for_init_diff_warning</b>             | Enable or disable warning when old-style <code>for</code> -scoping is used                                                      |
| <b>--gen_c_file_name</b> <i>file</i>          | Specify name of generated C output <i>file</i>                                                                                  |
| <b>--guiding_decls</b>                        |                                                                                                                                 |
| <b>--no_guiding_decls</b>                     | Enable or disable recognition of "guiding declarations" of template functions                                                   |
| <b>--implicit_extern_c_type_conversion</b>    |                                                                                                                                 |
| <b>--no_implicit_extern_c_type_conversion</b> | Enable or disable implicit type conversion between external C and C++ function pointers                                         |
| <b>--implicit_include</b>                     |                                                                                                                                 |
| <b>--no_implicit_include</b>                  |                                                                                                                                 |
| <b>-B</b>                                     | Enable or disable implicit inclusion of source files as a method of finding definitions of template entities to be instantiated |
| <b>--implicit_typename</b>                    |                                                                                                                                 |
| <b>--no_implicit_typename</b>                 | Enable or disable implicit determination, from context, whether a template parameter dependent name is a type or nontype        |

| Option                               | Description                                                                                                                    |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>--inlining</b>                    |                                                                                                                                |
| <b>--no_inlining</b>                 | Enable or disable minimal inlining of function calls                                                                           |
| <b>--instantiate <i>mode</i></b>     | Control instantiation of external template entities                                                                            |
| <b>--list <i>lfile</i></b>           | Generate raw list file <i>lfile</i>                                                                                            |
| <b>--long_lifetime_temps</b>         |                                                                                                                                |
| <b>--short_lifetime_temps</b>        | Select lifetime for temporaries                                                                                                |
| <b>--namespaces</b>                  |                                                                                                                                |
| <b>--no_namespaces</b>               | Enable or disable the support for namespaces                                                                                   |
| <b>--new_for_init</b>                | New-style <code>for</code> -scoping rules                                                                                      |
| <b>--no_code_gen</b>                 | Do syntax checking only                                                                                                        |
| <b>--no_line_commands</b>            | Preprocess only. Remove line control information and comments                                                                  |
| <b>--nonconst_ref_anachronism</b>    |                                                                                                                                |
| <b>--no_nonconst_ref_anachronism</b> | Enable or disable the anachronism of allowing a reference to <code>nonconst</code> to bind to a class rvalue of the right type |
| <b>--no_preproc_only</b>             | Specify that a full compilation should be done (not just preprocessing)                                                        |
| <b>--no_use_before_set_warnings</b>  | Suppress warnings on local automatic variables that are used before their values are set                                       |
| <b>--no_warnings</b>                 |                                                                                                                                |
| <b>-w</b>                            | Suppress all warning messages                                                                                                  |
| <b>--old_for_init</b>                | Old-style <code>for</code> -scoping rules                                                                                      |
| <b>--old_line_commands</b>           | Put out line control information in the form <code># <i>nnn</i></code> instead of <code>#line <i>nnn</i></code>                |
| <b>--old_specializations</b>         |                                                                                                                                |
| <b>--no_old_specializations</b>      | Enable or disable old-style template specialization                                                                            |
| <b>--old_style_preprocessing</b>     | Forces pcc style preprocessing                                                                                                 |
| <b>-opfile <i>file</i></b>           | Read command line arguments from <i>file</i>                                                                                   |
| <b>--output <i>file</i></b>          | Specify name of preprocess or intermediate output <i>file</i>                                                                  |
| <b>--pch</b>                         | Automatically use and/or create a precompiled header file                                                                      |

| Option                                                                    | Description                                                                                                                             |
|---------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>--pch_dir</b> <i>dir</i>                                               | Specify directory <i>dir</i> in which to search for and/or create a precompiled header file                                             |
| <b>--pch_messages</b><br><b>--no_pch_messages</b>                         | Enable or disable the display of a message indicating that a precompiled header file was created or used in the current compilation     |
| <b>--preprocess</b><br><b>-E</b>                                          | Preprocess only. Keep line control information and remove comments                                                                      |
| <b>--remarks</b>                                                          | Issue remarks                                                                                                                           |
| <b>--remove_unneeded_entities</b><br><b>--no_remove_unneeded_entities</b> | Enable or disable the removal of unneeded entities from the generated intermediate C file                                               |
| <b>--rtti</b><br><b>--no_rtti</b>                                         | Enable or disable support for RTTI (run-time type information)                                                                          |
| <b>--special_subscript_cost</b><br><b>--no_special_subscript_cost</b>     | Enable or disable a special nonstandard weighting of the conversion to the integral operand of the [ ] operator in overload resolution. |
| <b>--strict</b>                                                           | Strict ANSI C++. Issue errors on non-ANSI features                                                                                      |
| <b>--strict_warnings</b>                                                  | Strict ANSI C++. Issue warnings on non-ANSI features                                                                                    |
| <b>--suppress_vtbl</b>                                                    | Suppress definition of virtual function tables                                                                                          |
| <b>--trace_includes</b><br><b>-H</b>                                      | Preprocess only. Generate list of included files                                                                                        |
| <b>--typename</b><br><b>--no_typename</b>                                 | Enable or disable recognition of <code>typename</code>                                                                                  |
| <b>--undefine_macro</b> <i>macro</i><br><b>-U</b> <i>macro</i>            | Remove preprocessor <i>macro</i>                                                                                                        |
| <b>--use_pch</b> <i>file</i>                                              | Use a precompiled header file of the specified name                                                                                     |



| Option                                                    | Description                                                                                              |
|-----------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <b>--using_std</b><br><b>--no_using_std</b>               | Enable or disable implicit use of the <code>std</code> namespace when standard header files are included |
| <b>--wchar_t_keyword</b><br><b>--no_wchar_t_keyword</b>   | Enable or disable recognition of <code>wchar_t</code> as a keyword                                       |
| <b>--wrap_diagnostics</b><br><b>--no_wrap_diagnostics</b> | Enable or disable wrapping of diagnostic messages                                                        |
| <b>--xref xfile</b><br><b>-X xfile</b>                    | Generate cross-reference file <i>xfile</i>                                                               |

Table 3-1: Compiler options (alphabetical)

| Description                                                                                       | Option                                                     |
|---------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <b>Include options</b>                                                                            |                                                            |
| Read command line arguments from <i>file</i>                                                      | <b>--opfile file</b>                                       |
| <b>Preprocess options</b>                                                                         |                                                            |
| Preprocess only. Keep line control information and remove comments                                | <b>--preprocess</b><br><b>-E</b>                           |
| Preprocess only. Remove line control information and comments                                     | <b>--no_line_commands</b>                                  |
| Keep comments in the preprocessed output                                                          | <b>--comments</b>                                          |
| Put out line control information in the form <code># nnn</code> instead of <code>#line nnn</code> | <b>--old_line_commands</b>                                 |
| Forces pcc style preprocessing                                                                    | <b>--old_style_preprocessing</b>                           |
| Preprocess only. Emit dependencies for make                                                       | <b>--dependencies</b><br><b>-M</b>                         |
| Preprocess only. Generate list of included files                                                  | <b>--trace_includes</b><br><b>-H</b>                       |
| Define preprocessor <i>macro</i>                                                                  | <b>--define_macro macro[=def]</b><br><b>-D macro[=def]</b> |
| Remove preprocessor <i>macro</i>                                                                  | <b>--undefine_macro macro</b><br><b>-U macro</b>           |

| Description                                                                                   | Option                                                                                                  |
|-----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| Do syntax checking only                                                                       | <code>--no_code_gen</code>                                                                              |
| Specify that a full compilation should be done (not just preprocessing)                       | <code>--no_preproc_only</code>                                                                          |
| <b>Language control options</b>                                                               |                                                                                                         |
| Strict ANSI C++. Issue errors on non-ANSI features                                            | <code>--strict</code>                                                                                   |
| Strict ANSI C++. Issue warnings on non-ANSI features                                          | <code>--strict_warnings</code>                                                                          |
| Compile C++ compatible with cfront version 2.1                                                | <code>--cfront_2.1</code>                                                                               |
| Compile C++ compatible with cfront version 3.0                                                | <code>--cfront_3.0</code>                                                                               |
| Enable or disable support for exception handling                                              | <code>--exceptions</code><br><code>--no_exceptions</code><br><code>-x</code>                            |
| Enable the diagnostics of noncompliance with the "Embedded C++" subset                        | <code>--embedded_c++</code>                                                                             |
| Enable or disable operator functions to overload builtin operators on enum-typed operands     | <code>--enum_overloading</code><br><code>--no_enum_overloading</code>                                   |
| Treat enum as signed char, unsigned char, signed 16-bit int (default), or signed 32-bit int   | <code>--enum1s</code><br><code>--enum1u</code><br><code>--enum2s</code><br><code>--enum4s</code>        |
| Enable or disable support for the <code>explicit</code> specifier on constructor declarations | <code>--explicit</code><br><code>--no_explicit</code>                                                   |
| Enable or disable inline function with external C++ linkage                                   | <code>--extern_inline</code><br><code>--no_extern_inline</code>                                         |
| Enable or disable implicit type conversion between external C and C++ function pointers       | <code>--implicit_extern_c_type_conversion</code><br><code>--no_implicit_extern_c_type_conversion</code> |
| Suppress definition of virtual function tables                                                | <code>--suppress_vtbl</code>                                                                            |
| Force definition of virtual function tables                                                   | <code>--force_vtbl</code>                                                                               |
| Enable or disable anachronisms                                                                | <code>--anachronisms</code><br><code>--no_anachronisms</code>                                           |

| Description                                                                                                                                         | Option                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Enable or disable the anachronism of allowing a reference to <code>nonconst</code> to bind to a class rvalue of the right type                      | <code>--nonconst_ref_anachronism</code><br><code>--no_nonconst_ref_anachronism</code> |
| Enable or disable support for array new and delete                                                                                                  | <code>--array_new_and_delete</code><br><code>--no_array_new_and_delete</code>         |
| Enable or disable support for namespaces                                                                                                            | <code>--namespaces</code><br><code>--no_namespaces</code>                             |
| New-style <code>for</code> -scoping rules                                                                                                           | <code>--new_for_init</code>                                                           |
| Old-style <code>for</code> -scoping rules                                                                                                           | <code>--old_for_init</code>                                                           |
| Enable or disable implicit use of the <code>std</code> namespace when standard header files are included                                            | <code>--using_std</code><br><code>--no_using_std</code>                               |
| Enable or disable support for RTTI (run-time type information)                                                                                      | <code>--rtti</code><br><code>--no_rtti</code>                                         |
| Enable or disable recognition of <code>bool</code>                                                                                                  | <code>--bool</code><br><code>--no_bool</code>                                         |
| Enable or disable recognition of <code>typename</code>                                                                                              | <code>--typename</code><br><code>--no_typename</code>                                 |
| Enable or disable implicit determination, from context, whether a template parameter dependent name is a type or nontype                            | <code>--implicit_typename</code><br><code>--no_implicit_typename</code>               |
| Enable or disable a special nonstandard weighting of the conversion to the integral operand of the <code>[]</code> operator in overload resolution. | <code>--special_subscript_cost</code><br><code>--no_special_subscript_cost</code>     |
| Enable or disable recognition of <code>wchar_t</code> as a keyword                                                                                  | <code>--wchar_t_keyword</code><br><code>--no_wchar_t_keyword</code>                   |
| Select lifetime for temporaries                                                                                                                     | <code>--long_lifetime_temps</code><br><code>--short_lifetime_temps</code>             |
| Enable or disable recognition of alternative tokens                                                                                                 | <code>--alternative_tokens</code><br><code>--no_alternative_tokens</code>             |
| Enable or disable minimal inlining of function calls                                                                                                | <code>--inlining</code><br><code>--no_inlining</code>                                 |
| Enable or disable the removal of unneeded entities from the generated intermediate C file                                                           | <code>--remove_unneeded_entities</code><br><code>--no_remove_unneeded_entities</code> |

| Description                                                                                                                         | Option                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| <b>Template instantiation options</b>                                                                                               |                                                                                               |
| Control instantiation of external template entities                                                                                 | <code>--instantiate <i>mode</i></code>                                                        |
| Enable or disable automatic instantiation of templates                                                                              | <code>--auto_instantiation</code><br><code>--no_auto_instantiation</code>                     |
| Enable or disable implicit inclusion of source files as a method of finding definitions of template entities to be instantiated     | <code>--implicit_include</code><br><code>--no_implicit_include</code><br><code>-B</code>      |
| Dis-allow or allow normal functions as template instantiation                                                                       | <code>--distinct_template_signatures</code><br><code>--no_distinct_template_signatures</code> |
| Enable or disable recognition of "guiding declarations" of template functions                                                       | <code>--guiding_decls</code><br><code>--no_guiding_decls</code>                               |
| Enable or disable old-style template specialization                                                                                 | <code>--old_specializations</code><br><code>--no_old_specializations</code>                   |
| <b>Precompiled header options</b>                                                                                                   |                                                                                               |
| Automatically use and/or create a precompiled header file                                                                           | <code>--pch</code>                                                                            |
| Create a precompiled header file with the specified name                                                                            | <code>--create_pch <i>file</i></code>                                                         |
| Use a precompiled header file of the specified name                                                                                 | <code>--use_pch <i>file</i></code>                                                            |
| Specify directory <i>dir</i> in which to search for and/or create a precompiled header file                                         | <code>--pch_dir <i>dir</i></code>                                                             |
| Enable or disable the display of a message indicating that a precompiled header file was created or used in the current compilation | <code>--pch_messages</code><br><code>--no_pch_messages</code>                                 |
| <b>Output file options</b>                                                                                                          |                                                                                               |
| Specify name of preprocess or intermediate output <i>file</i>                                                                       | <code>--output <i>file</i></code>                                                             |
| Specify name of generated C output <i>file</i>                                                                                      | <code>--gen_c_file_name <i>file</i></code>                                                    |
| <b>Diagnostic options</b>                                                                                                           |                                                                                               |
| Send diagnostics to error list file                                                                                                 | <code>--error_output <i>efile</i></code>                                                      |
| Generate raw list file <i>lfile</i>                                                                                                 | <code>--list <i>lfile</i></code>                                                              |

| Description                                                                              | Option                                                                                                                                                                     |
|------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Generate cross-reference file <i>xfile</i>                                               | <code>--xref xfile</code><br><code>-X xfile</code>                                                                                                                         |
| Override normal error severity                                                           | <code>--diag_suppress tag[,tag]...</code><br><code>--diag_remark tag[,tag]...</code><br><code>--diag_warning tag[,tag]...</code><br><code>--diag_error tag[,tag]...</code> |
| Display error number in diagnostic messages                                              | <code>--display_error_number</code>                                                                                                                                        |
| Specify maximum <i>number</i> of errors                                                  | <code>--error_limit number</code>                                                                                                                                          |
| Issue remarks                                                                            | <code>--remarks</code>                                                                                                                                                     |
| Suppress all warning messages                                                            | <code>--no_warnings</code><br><code>-w</code>                                                                                                                              |
| Suppress warnings on local automatic variables that are used before their values are set | <code>--no_use_before_set_warnings</code>                                                                                                                                  |
| Enable or disable a shorter form of diagnostic output                                    | <code>--brief_diagnostics</code><br><code>--no_brief_diagnostics</code>                                                                                                    |
| Enable or disable wrapping of diagnostic messages                                        | <code>--wrap_diagnostics</code><br><code>--no_wrap_diagnostics</code>                                                                                                      |
| Enable or disable warning when old-style <code>for</code> -scoping is used               | <code>--for_init_diff_warning</code><br><code>--no_for_init_diff_warning</code>                                                                                            |

Table 3-2: Compiler options (functional)

3.1.1 DETAILED DESCRIPTION OF THE COMPILER OPTIONS

Option letters are listed below. If the same option is used more than once, the first (most left) occurrence is used. The placement of command line options is of no importance except for the `-I` option. Some options also have a "no\_" form. These options are described together.

## --alternative\_tokens

### Option:

**--alternative\_tokens**  
**--no\_alternative\_tokens**

### Default:

**--alternative\_tokens**

### Description:

Enable or disable recognition of alternative tokens. This controls recognition of the digraph tokens in C++, and controls recognition of the operator keywords (e.g., **not**, **and**, **bitand**, etc.).

### Example:

To disable operator keywords (e.g., "not", "and") and digraphs, enter:

```
cpxxx test.cc --no_alternative_tokens
```

## --anachronisms

### Option:

**--anachronisms**  
**--no\_anachronisms**

### Default:

**--no\_anachronisms**

### Description:

Enable or disable anachronisms.

### Example:

```
cpxxx test.cc --anachronisms
```



**--nonconst\_ref\_anachronisms,**  
**--cfront\_2.1 / --cfront\_3.0**

Section *Anachronisms Accepted* in chapter *Language Implementation*.

## **--array\_new\_and\_delete**

### **Option:**

**--array\_new\_and\_delete**  
**--no\_array\_new\_and\_delete**

### **Default:**

**--array\_new\_and\_delete**

### **Description:**

Enable or disable support for array new and delete.

### **Example:**

```
cpxxx test.cc --no_array_new_and_delete
```



## --auto\_instantiation

### Option:

**--auto\_instantiation**  
**--no\_auto\_instantiation**

### Default:

**--auto\_instantiation**

### Description:

Enable or disable automatic instantiation of templates.

### Example:

```
cpxxx test.cc --no_auto_instantiation
```



**--instantiate**

Section *Template Instantiation* in chapter *Language Implementation*.

## **--bool**

### **Option:**

**--bool**  
**--no\_bool**

### **Default:**

**--bool**

### **Description:**

Enable or disable recognition of the `bool` keyword.

### **Example:**

```
cpxxx test.cc --no_bool
```

## --brief\_diagnostics

### Option:

**--brief\_diagnostics**  
**--no\_brief\_diagnostics**

### Default:

**--brief\_diagnostics**

### Description:

Enable or disable a mode in which a shorter form of the diagnostic output is used. When enabled, the original source line is not displayed and the error message text is not wrapped when too long to fit on a single line.

### Example:

```
cpxxx test.cc --no_brief_diagnostics
```



**--wrap\_diagnostics**

Chapter *Compiler Diagnostics* and Appendix *Error Messages*.

## **--cfront\_version**

### **Option:**

**--cfront\_2.1**  
**--cfront\_3.0**

### **Default:**

Normal C++ mode.

### **Description:**

**--cfront\_2.1** or **--cfront\_3.0** enable compilation of C++ with compatibility with cfront version 2.1 or 3.0 respectively. This causes the compiler to accept language constructs that, while not part of the C++ language definition, are accepted by the AT&T C++ Language System (cfront) release 2.1 or 3.0 respectively. These options also enable acceptance of anachronisms.

### **Example:**

To compile C++ compatible with cfront version 3.0, enter:

```
cpxxx test.cc --cfront_3.0
```



### **--anachronisms**

Section *Extensions Accepted in Cfront 2.1 and 3.0 Compatibility Mode* in chapter *Language Implementation*.

## --comments

### Option:

**--comments**

### Description:

Keep comments in the preprocessed output. This should be specified after either **--preprocess** or **--no\_line\_commands**; it does not of itself request preprocessing output.

### Example:

To do preprocessing only, with comments and with line control information, enter:

```
cpxxx test.cc -E --comments
```



```
--preprocess / -E, --no_line_commands
```

## --create\_pch

### Option:

**--create\_pch** *filename*

### Arguments:

A filename specifying the precompiled header file to create.

### Description:

If other conditions are satisfied (see the *Precompiled Headers* section), create a precompiled header file with the specified name. If **--pch** (automatic PCH mode) or **--use\_pch** appears on the command line following this option, its effect is erased.

### Example:

To create a precompiled header file with the name `test.pch`, enter:

```
cpxxx test.cc --create_pch test.pch
```



**--pch, --use\_pch**

Section *Precompiled Headers* in chapter *Language Implementation*.

## --define\_macro / -D

### Option:

**-D** *macro*[=*def*]  
**--define\_macro** *macro*[=*def*]

### Arguments:

The macro you want to define and optionally its definition.

### Description:

Define *macro* to the preprocessor, as in #define. If *def* is not given ('=' is absent), '1' is assumed. Any number of symbols can be defined. The definition can be tested by the preprocessor with #if, #ifdef and #ifndef, for conditional compilations.

### Example:

```
cpxxx test.cc -D NORAM -D PI=3.1416
```



**--undefine\_macro / -U**

## --dependencies / -M

### Option:

**-M**  
**--dependencies**

### Description:

Do preprocessing only. Instead of the normal preprocessing output, generate on the preprocessing output file a list of dependency lines suitable for input to a 'make' utility.



When implicit inclusion of templates is enabled, the output may indicate false (but safe) dependencies unless **--no\_proproc\_only** is also used.

### Examples:

```
cpxxx test.cc -M
```

```
test.ic: test.cc
```



**--preprocess / -E, --no\_line\_commands**



## --diag\_

### Option:

- diag\_suppress** *tag[,tag]...*
- diag\_remark** *tag[,tag]...*
- diag\_warning** *tag[,tag]...*
- diag\_error** *tag[,tag]...*

### Arguments:

A mnemonic error tag or an error number.

### Description:

Override the normal error severity of the specified diagnostic messages. The message(s) may be specified using a mnemonic error tag or using an error number. The error tag names and error numbers are listed in the *Error Messages* appendix.

### Example:

When you want diagnostic error 20 to be a warning, enter:

```
cpxxx test.cc --diag_warning 20
```



Chapter *Compiler Diagnostics* and Appendix *Error Messages*.

## --display\_error\_number

### Option:

**--display\_error\_number**

### Description:

Display the error message number in any diagnostic messages that are generated. The option may be used to determine the error number to be used when overriding the severity of a diagnostic message. The error numbers are listed in the *Error Messages* appendix.

Normally, diagnostics are written to stderr in the following form:

*"filename", line line\_num: message*

With **--display\_error\_number** this form will be:

*"filename", line line\_num: severity #err\_num: message*

or:

*"filename", line line\_num: severity #err\_num-D: message*

If the severity may be overridden, the error number will include the suffix **-D** (for discretionary); otherwise no suffix will be present.

### Example:

```
cpxxx test.cc --display_error_number
```

```
"test.cc", line 7: error #64-D: declaration does not  
declare anything
```



Chapter *Compiler Diagnostics* and Appendix *Error Messages*.

## **--distinct\_template\_signatures**

### **Option:**

**--distinct\_template\_signatures**  
**--no\_distinct\_template\_signatures**

### **Default:**

**--distinct\_template\_signatures**

### **Description:**

Control whether the signatures for template functions can match those for non-template functions when the functions appear in different compilation units. The default is **--distinct\_template\_signatures**, under which a normal function cannot be used to satisfy the need for a template instance; e.g., a function "void f(int)" could not be used to satisfy the need for an instantiation of a template "void f(T)" with T set to int.

**--no\_distinct\_template\_signatures** provides the older language behavior, under which a non-template function can match a template function.

### **Example:**

```
cpxxx test.cc --no_distinct_template_signatures
```

## **--embedded\_c++**

### **Option:**

**--embedded\_c++**

### **Description:**

Enable the diagnostics of noncompliance with the “Embedded C++” subset (from which templates, exceptions, namespaces, new-style casts, RTTI, multiple inheritance, virtual base classes, and mutable are excluded).

### **Example:**

To enable the diagnostics of noncompliance with the “Embedded C++” subset, enter:

```
cpxxx test.cc --embedded_c++
```

## **--enum**

### **Option:**

- enum1s**
- enum1u**
- enum2s**
- enum4s**

### **Default:**

- enum2s**

### **Description:**

Implement enums as another type:

- enum1s** treat enum as signed char
- enum1u** treat enum as unsigned char
- enum2s** treat enum as signed 16-bit int (default)
- enum4s** treat enum as signed 32-bit int

### **Example:**

To treat an enum as a signed char, enter:

```
cpxxx test.cc --enum1s
```

## **--enum\_overloading**

### **Option:**

**--enum\_overloading**  
**--no\_enum\_overloading**

### **Default:**

**--enum\_overloading**

### **Description:**

Enable or disable support for using operator functions to overload builtin operations on enum-typed operands.

### **Example:**

To disable overloading builtin operations on enum-typed operands, enter:

```
cpxxx test.cc --no_enum_overloading
```

## **--error\_limit**

### **Option:**

**--error\_limit** *number*

### **Arguments:**

An error limit number.

### **Default:**

**--error\_limit 100**

### **Description:**

Set the error limit to *number*. The C++ compiler will abandon compilation after this number of errors (remarks and warnings are not counted toward the limit). By default, the limit is 100.

### **Example:**

When you want compilation to stop when 10 errors occurred, enter:

```
cpxxx test.cc --error_limit 10
```

## **--error\_output**

### **Option:**

**--error\_output** *efile*

### **Arguments:**

The name for an error output file.

### **Description:**

Redirect the output that would normally go to stderr (i.e., diagnostic messages) to the file *efile*. This option is useful on systems where output redirection of files is not well supported. If used, this option should probably be specified first in the command line, since otherwise any command-line errors for options preceding the **--error\_output** would be written to stderr before redirection.

### **Example:**

To write errors to the file `test.err` instead of stderr, enter:

```
cpxxx test.cc --error_output test.err
```



## **--exceptions / -x**

### **Option:**

**-x / --exceptions**  
**--no\_exceptions**

### **Default:**

**--no\_exceptions**

### **Description:**

Enable or disable support for exception handling. **-x** is equivalent to **--exceptions**.

### **Example:**

```
cpxxx test.cc --exceptions
```

## **--explicit**

### **Option:**

**--explicit**  
**--no\_explicit**

### **Default:**

**--explicit**

### **Description:**

Enable or disable support for the `explicit` specifier on constructor declarations.

### **Example:**

To disable support for the `explicit` specifier on constructor declarations, enter:

```
cpxxx test.cc --no_explicit
```

## **--extern\_inline**

### **Option:**

**--extern\_inline**  
**--no\_extern\_inline**

### **Default:**

**--extern\_inline**

### **Description:**

Enable or disable support for inline functions with external linkage in C++. When inline functions are allowed to have external linkage (as required by the standard), then `extern` and `inline` are compatible specifiers on a non-member function declaration; the default linkage when `inline` appears alone is external (that is, `inline` means `extern inline` on non-member functions); and an `inline` member function takes on the linkage of its class (which is usually external). However, when inline functions have only internal linkage (as specified in the ARM), then `extern` and `inline` are incompatible; the default linkage when `inline` appears alone is internal (that is, `inline` means `static inline` on non-member functions); and `inline` member functions have internal linkage no matter what the linkage of their class.

### **Example:**

```
cpxxx test.cc --no_extern_inline
```

## **--for\_init\_diff\_warning**

### **Option:**

**--for\_init\_diff\_warning**  
**--no\_for\_init\_diff\_warning**

### **Default:**

**--for\_init\_diff\_warning**

### **Description:**

Enable or disable a warning that is issued when programs compiled under the new for-init scoping rules would have had different behavior under the old rules. The diagnostic is only put out when the new rules are used.

### **Example:**

```
cpxxx test.cc --no_for_init_diff_warning
```



**--new\_for\_init / --old\_for\_init**

## **--force\_vtbl**

### **Option:**

**--force\_vtbl**

### **Description:**

Force definition of virtual function tables in cases where the heuristic used by the C++ compiler to decide on definition of virtual function tables provides no guidance. See **--suppress\_vtbl**.

### **Example:**

```
cpxxx test.cc --force_vtbl
```



**--suppress\_vtbl**

## **--gen\_c\_file\_name**

### **Option:**

**--gen\_c\_file\_name** *file*

### **Arguments:**

An output filename.

### **Description:**

This option specifies the file name to be used for the generated C output.

### **Example:**

To specify the file `out.ic` as the output file instead of `test.ic`, enter:

```
cpxxx test.cc --gen_c_file_name out.ic
```

## --guiding\_decls

### Option:

**--guiding\_decls**  
**--no\_guiding\_decls**

### Default:

**--guiding\_decls**

### Description:

Enable or disable recognition of “guiding declarations” of template functions. A guiding declaration is a function declaration that matches an instance of a function template but has no explicit definition (since its definition derives from the function template). For example:

```
template <class T> void f(T) { ... }
void f(int);
```

When regarded as a guiding declaration, `f(int)` is an instance of the template; otherwise, it is an independent function for which a definition must be supplied. If **--no\_guiding\_decls** is combined with **--old\_specializations**, a specialization of a non-member template function is not recognized — it is treated as a definition of an independent function.

### Example:

```
cpxxx test.cc --no_guiding_decls
```



**--old\_specializations**

## **--implicit\_extern\_c\_type\_conversion**

### **Option:**

**--implicit\_extern\_c\_type\_conversion**  
**--no\_implicit\_extern\_c\_type\_conversion**

### **Default:**

**--implicit\_extern\_c\_type\_conversion**

### **Description:**

Enable or disable an extension to permit implicit type conversion in C++ between a pointer to an `extern "C"` function and a pointer to an `extern "C++"` function. This extension is allowed in environments where C and C++ functions share the same calling conventions.

### **Example:**

```
cpxxx test.cc --no_implicit_extern_c_type_conversion
```



## **--implicit\_include / -B**

### **Option:**

**-B / --implicit\_include**  
**--no\_implicit\_include**

### **Default:**

**--no\_implicit\_include**

### **Description:**

Enable or disable implicit inclusion of source files as a method of finding definitions of template entities to be instantiated. **-B** is equivalent to **--implicit\_include**.

### **Example:**

```
cpxxx test.cc --implicit_include
```



**--instantiate**

Section *Template Instantiation* in chapter *Language Implementation*.

## **--implicit\_typename**

### **Option:**

**--implicit\_typename**  
**--no\_implicit\_typename**

### **Default:**

**--implicit\_typename**

### **Description:**

Enable or disable implicit determination, from context, whether a template parameter dependent name is a type or nontype.

### **Example:**

```
cpxxx test.cc --no_implicit_typename
```



**--typename**

## **--inlining**

### **Option:**

**--inlining**  
**--no\_inlining**

### **Default:**

**--inlining**

### **Description:**

Enable or disable minimal inlining of function calls.

### **Example:**

To disable function call inlining, enter:

```
cpxxx test.cc --no_inlining
```

## --**instantiate**

### Option:

**--instantiate** *mode*

### Pragma:

**instantiate** *mode*

### Arguments:

The instantiation mode, which can be one of:

**none**  
**used**  
**all**  
**local**

### Default:

**--instantiate none**

### Description:

Control instantiation of external template entities. External template entities are external (i.e., noninline and nonstatic) template functions and template static data members. The instantiation mode determines the template entities for which code should be generated based on the template definition:

|              |                                                                                                                                     |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>none</b>  | Instantiate no template entities. This is the default.                                                                              |
| <b>used</b>  | Instantiate only the template entities that are used in this compilation.                                                           |
| <b>all</b>   | Instantiate all template entities whether or not they are used.                                                                     |
| <b>local</b> | Instantiate only the template entities that are used in this compilation, and force those entities to be local to this compilation. |

### Example:

To specify to instantiate only the template entities that are used in this compilation, enter:

```
cpxxx --instantiate used test.cc
```

**--auto\_instantiation**

Section *Template Instantiation* in chapter *Language Implementation*.

## --list

### Option:

**--list** *lfile*

### Arguments:

The name of the list file.

### Description:

Generate raw listing information in the file *lfile*. The raw listing file contains raw source lines, information on transitions into and out of include files, and diagnostics generated by the C++ compiler. Each line of the listing file begins with a key character that identifies the type of line, as follows:

**N:** a normal line of source; the rest of the line is the text of the line.

**X:** the expanded form of a normal line of source; the rest of the line is the text of the line. This line appears following the N line, and only if the line contains non-trivial modifications (comments are considered trivial modifications; macro expansions, line splices, and trigraphs are considered non-trivial modifications).

**S:** a line of source skipped by an `#if` or the like; the rest of the line is text. Note that the `#else`, `#elif`, or `#endif` that ends a skip is marked with an N.

**L:** an indication of a change in source position. The line has a format similar to the `#` line-identifying directive output by `cpp`, that is to say

**L** *line\_number* "*file-name*" *key*

where *key* is,

- 1 for entry into an include file;
- 2 for exit from an include file;

and omitted otherwise.

The first line in the raw listing file is always an L line identifying the primary input file. L lines are also output for `#line` directives (*key* is omitted). L lines indicate the source position of the following source line in the raw listing file.

**R, W, E, or C:** an indication of a diagnostic (R for remark, W for warning, E for error, and C for catastrophic error). The line has the form

*S "file-name" line\_number column-number message-text*

where *S* is R, W, E, or C, as explained above. Errors at the end of file indicate the last line of the primary source file and a column number of zero. Command line errors are catastrophes with an empty file name (" ") and a line and column number of zero. Internal errors are catastrophes with position information as usual, and message-text beginning with (internal error). When a diagnostic displays a list (e.g., all the contending routines when there is ambiguity on an overloaded call), the initial diagnostic line is followed by one or more lines with the same overall format (code letter, file name, line number, column number, and message text), but in which the code letter is the lower case version of the code letter in the initial line. The source position in such lines is the same as that in the corresponding initial line.

#### Example:

To write raw listing information to the file `test.lst`, enter:

```
cpxxx test.cc --list test.lst
```

## **--long\_lifetime\_temps / --short\_lifetime\_temps**

### **Option:**

**--long\_lifetime\_temps**  
**--short\_lifetime\_temps**

### **Default:**

**--long\_lifetime\_temps** (cfront)  
**--short\_lifetime\_temps** (standard C++)

### **Description:**

Select the lifetime for temporaries: short means to end of full expression; long means to the earliest of end of scope, end of switch clause, or the next label. Short is standard C++, and long is what cfront uses (the cfront compatibility modes select long by default).

### **Example:**

```
cpxxx test.cc --long_lifetime_temps
```



## --namespaces

### Option:

**--namespaces**  
**--no\_namespaces**

### Default:

**--namespaces**

### Description:

Enable or disable support for namespaces.

### Example:

```
cpxxx test.cc --no_namespaces
```



**--using\_std**

Section *Namespace Support* in chapter *Language Implementation*.

## **--new\_for\_init / --old\_for\_init**

### **Option:**

**--new\_for\_init**  
**--old\_for\_init**

### **Default:**

**--new\_for\_init**

### **Description:**

Control the scope of a declaration in a `for-init-statement`. The old (cfront-compatible) scoping rules mean the declaration is in the scope to which the `for` statement itself belongs; the new (standard-conforming) rules in effect wrap the entire `for` statement in its own implicitly generated scope.

### **Example:**

```
cpxxx test.cc --old_for_init
```

## **--no\_code\_gen**

### **Option:**

**--no\_code\_gen**

### **Description:**

Do syntax-checking only. Do not generate a C file.

### **Example:**

```
cpxxx test.cc --no_code_gen
```

## **--no\_line\_commands**

### **Option:**

**--no\_line\_commands**

### **Description:**

Do preprocessing only. Write preprocessed text to the preprocessing output file, with comments removed and without line control information.

### **Examples:**

```
cpxxx test.cc --no_line_commands --output preout
```



**--comments, --preprocess / -E, --dependencies / -M**

## --nonconst\_ref\_anachronism

### Option:

**--nonconst\_ref\_anachronism**  
**--no\_nonconst\_ref\_anachronism**

### Default:

**--nonconst\_ref\_anachronism**

### Description:

Enable or disable the anachronism of allowing a reference to nonconst to bind to a class rvalue of the right type. This anachronism is also enabled by the **--anachronisms** option and the cfront-compatibility options.

### Example:

```
cpxxx test.cc --no_nonconst_ref_anachronism
```



**--anachronisms, --cfront\_2.1 / --cfront\_3.0**

Section *Anachronisms Accepted* in chapter *Language Implementation*.

## **--no\_preproc\_only**

### **Option:**

**--no\_preproc\_only**

### **Description:**

May be used in conjunction with the options that normally cause the C++ compiler to do preprocessing only (e.g., **--preprocess**, etc.) to specify that a full compilation should be done (not just preprocessing). When used with the implicit inclusion option, this makes it possible to generate a preprocessed output file that includes any implicitly included files.

### **Examples:**

```
cpxxx test.cc -E -B --no_preproc_only
```



**--preprocess / -E,**  
**--implicit\_include / -B, --no\_line\_commands**

## **--no\_use\_before\_set\_warnings**

### **Option:**

**--no\_use\_before\_set\_warnings**

### **Description:**

Suppress warnings on local automatic variables that are used before their values are set.

### **Example:**

```
cpxxx test.cc --no_use_before_set_warnings
```



```
--no_warnings / -w
```

## **--no\_warnings / -w**

### **Option:**

**-w**  
**--no\_warnings**

### **Description:**

Suppress all warning messages. Error messages are still issued.

### **Example:**

To suppress all warnings, enter:

```
cpxxx test.cc -w
```



## **-opfile**

### **Option:**

**-opfile** *file*

### **Arguments:**

A filename for command line processing.

### **Description:**

Use *file* for command line processing. *file* can contain arbitrary command line options and filenames. It must be the last argument of the command line.

### **Example:**

To read arguments from file `config.opt`, enter:

```
cpxxx test.cc --exceptions -opfile config.opt
```

## **--output**

### **Option:**

**--output** *file*

### **Arguments:**

An output filename specifying the preprocessing output file.

### **Default:**

No preprocessing output file is generated.

### **Description:**

Use *file* as output filename for the preprocessing output file.

### **Example:**

To use the file `my.pre` as the preprocessing output file, enter:

```
cpxxx test.cc -E --output my.pre
```

## --old\_line\_commands

### Option:

**--old\_line\_commands**

### Description:

When generating source output, put out #line directives in the form used by the Reiser cpp, i.e., # *nnn* instead of #line *nnn*.

### Example:

To do preprocessing only, without comments and with old style line control information, enter:

```
cpxxx test.cc -E --old_line_commands
```



```
--preprocess / -E, --no_line_commands
```

## **--old\_specializations**

### **Option:**

**--old\_specializations**  
**--no\_old\_specializations**

### **Default:**

**--old\_specializations**

### **Description:**

Enable or disable acceptance of old-style template specializations (i.e., specializations that do not use the `template<>` syntax).

### **Example:**

```
cpxxx test.cc --no_old_specializations
```

## **--old\_style\_preprocessing**

### **Option:**

**--old\_style\_preprocessing**

### **Description:**

Forces pcc style preprocessing when compiling. This may be used when compiling an ANSI C++ program on a system in which the system header files require pcc style preprocessing.

### **Example:**

To force pcc style preprocessing, enter:

```
cpxxx test.cc -E --old_style_preprocessing
```



```
--preprocess / -E, --no_line_commands
```

## --pch

### Option:

**--pch**

### Description:

Automatically use and/or create a precompiled header file. For details, see the *Precompiled Headers* section in chapter *Language Implementation*. If **--use\_pch** or **--create\_pch** (manual PCH mode) appears on the command line following this option, its effect is erased.

### Example:

```
cpxxx test.cc --pch
```



**--use\_pch, --create\_pch**

Section *Precompiled Headers* in chapter *Language Implementation*.

## --pch\_dir

### Option:

**--pch\_dir** *dir\_name*

### Arguments:

The name of the directory to search for and/or create a precompiled header file.

### Description:

Specify the directory in which to search for and/or create a precompiled header file. This option may be used with automatic PCH mode (**--pch**) or manual PCH mode (**--create\_pch** or **--use\_pch**).

### Example:

To use the directory `/usr/include/pch` to automatically create precompiled header files, enter:

```
cpxxx test.cc --pch_dir /usr/include/pch --pch
```



**--pch, --use\_pch, --create\_pch**

Section *Precompiled Headers* in chapter *Language Implementation*.

## --pch\_messages

### Option:

**--pch\_messages**  
**--no\_pch\_messages**

### Default:

**pch\_messages**

### Description:

Enable or disable the display of a message indicating that a precompiled header file was created or used in the current compilation.

### Example:

```
cpxxx test.cc --create_pch test.pch --pch_messages
```

```
"test.cc": creating precompiled header file "test.pch"
```



**--pch, --use\_pch, --create\_pch**

Section *Precompiled Headers* in chapter *Language Implementation*.



## **--preprocess / -E**

### **Option:**

**-E**  
**--preprocess**

### **Description:**

Do preprocessing only. Write preprocessed text to the preprocessing output file, with comments removed and with line control information.

### **Example:**

```
cpxxx test.cc -E
```



**--comments,**  
**--dependencies / -M,**  
**--no\_line\_commands**

## **--remarks**

### **Option:**

**--remarks**

### **Description:**

Issue remarks, which are diagnostic messages even milder than warnings.

### **Example:**

To enable the display of remarks, enter:

```
cpxxx test.cc --remarks
```

## **--remove\_unneeded\_entities**

### **Option:**

**--remove\_unneeded\_entities**  
**--no\_remove\_unneeded\_entities**

### **Default:**

**--remove\_unneeded\_entities**

### **Description:**

Enable or disable an optimization to remove unneeded entities from the generated intermediate C file. Something may be referenced but unneeded if it is referenced only by something that is itself unneeded; certain entities, such as global variables and routines defined in the translation unit, are always considered to be needed.

### **Example:**

```
cpxxx test.cc --no_remove_unneeded_entities
```

## **--rtti**

### **Option:**

**--rtti**  
**--no\_rtti**

### **Default:**

**--no\_rtti**

### **Description:**

Enable or disable support for RTTI (run-time type information) features: `dynamic_cast`, `typeid`.

### **Example:**

```
cpxxx test.cc --rtti
```

## **--special\_subscript\_cost**

### **Option:**

**--special\_subscript\_cost**  
**--no\_special\_subscript\_cost**

### **Default:**

**--no\_special\_subscript\_cost**

### **Description:**

Enable or disable a special nonstandard weighting of the conversion to the integral operand of the `[]` operator in overload resolution.

This is a compatibility feature that may be useful with some existing code. The special cost is enabled by default in cfront 3.0 mode. With this feature enabled, the following code compiles without error:

```
struct A {
    A();
    operator int *();
    int operator[](unsigned);
};
void main() {
    A a;
    a[0]; // Ambiguous, but allowed with this option
        // operator[] is chosen
}
```

### **Example:**

**cpxxx test.cc --special\_subscript\_cost**

## **--strict**

## **--strict\_warnings**

### **Option:**

**--strict**

**--strict\_warnings**

### **Description:**

Enable strict ANSI mode, which provides diagnostic messages when non-ANSI features are used, and disables features that conflict with ANSI C or C++. ANSI violations can be issued as either warnings or errors depending on which command line option is used. The **--strict** options issue errors and the **--strict\_warnings** options issue warnings. The error threshold is set so that the requested diagnostics will be listed.

### **Example:**

To enable strict ANSI mode, with error diagnostic messages, enter:

```
cpxxx test.cc --strict
```

## **--suppress\_vtbl**

### **Option:**

**--suppress\_vtbl**

### **Description:**

Suppress definition of virtual function tables in cases where the heuristic used by the C++ compiler to decide on definition of virtual function tables provides no guidance. The virtual function table for a class is defined in a compilation if the compilation contains a definition of the first non-inline non-pure virtual function of the class. For classes that contain no such function, the default behavior is to define the virtual function table (but to define it as a local static entity). The **--suppress\_vtbl** option suppresses the definition of the virtual function tables for such classes, and the **--force\_vtbl** option forces the definition of the virtual function table for such classes. **--force\_vtbl** differs from the default behavior in that it does not force the definition to be local.

### **Example:**

```
cpxxx test.cc --suppress_vtbl
```



```
--force_vtbl
```

## **--trace\_includes / -H**

### **Option:**

**-H**  
**--trace\_includes**

### **Description:**

Do preprocessing only. Instead of the normal preprocessing output, generate on the preprocessing output file a list of the names of files #included.

### **Examples:**

```
cpxxx test.cc -H
```

```
iostream.h  
string.h
```



```
--preprocess / -E, --no_line_commands
```



## **--typename**

### **Option:**

**--typename**  
**--no\_typename**

### **Default:**

**--typename**

### **Description:**

Enable or disable recognition of the `typename` keyword.

### **Example:**

```
cpxxx test.cc --no_typename
```



**--implicit\_typename**

## --undefine\_macro / -U

### Option:

**-U** *name*  
**--undefine\_macro** *name*

### Arguments:

The name macro you want to undefine.

### Description:

Remove any initial definition of identifier *name* as in #undef, unless it is a predefined ANSI standard macro. ANSI specifies the following predefined symbols to exist, which cannot be removed:

|                          |                                                                                                                                                                                                                |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__FILE__</code>    | "current source filename"                                                                                                                                                                                      |
| <code>__LINE__</code>    | current source line number (int type)                                                                                                                                                                          |
| <code>__TIME__</code>    | "hh:mm:ss"                                                                                                                                                                                                     |
| <code>__DATE__</code>    | "Mmm dd yyyy"                                                                                                                                                                                                  |
| <code>__STDC__</code>    | level of ANSI standard. This macro is set to 1 when the option to disable language extensions ( <b>-A</b> ) is effective. Whenever language extensions are excepted, <code>__STDC__</code> is set to 0 (zero). |
| <code>__cplusplus</code> | is defined when compiling a C++ program                                                                                                                                                                        |

When **cpxxx** is invoked, also the following predefined symbols exist:

`c_plusplus` is defined in addition to the standard `__cplusplus`

`__SIGNED_CHARS__`  
 is defined when plain char is signed.

`__WCHAR_T` is defined when `wchar_t` is a keyword.

`__BOOL` is defined when `bool` is a keyword.

`__ARRAY_OPERATORS`  
 is defined when array new and delete are enabled.

These symbols can be turned off with the **-U** option.

**Example:**

```
cpxxx test.cc -U cplusplus
```



```
-D / --define_macro
```

## --use\_pch

### Option:

**--use\_pch** *filename*

### Arguments:

The filename to use as a precompiled header file.

### Description:

Use a precompiled header file of the specified name as part of the current compilation. If **--pch** (automatic PCH mode) or **--create\_pch** appears on the command line following this option, its effect is erased.

### Example:

To use the precompiled header file with the name `test.pch`, enter:

```
cpxxx test.cc --use_pch test.pch
```



**--pch, --create\_pch**

Section *Precompiled Headers* in chapter *Language Implementation*.

## --using\_std

### Option:

**--using\_std**  
**--no\_using\_std**

### Default:

**using\_std**

### Description:

Enable or disable implicit use of the `std` namespace when standard header files are included.

### Example:

```
cpxxx test.cc --using_std
```



### --namespaces

Section *Namespace Support* in chapter *Language Implementation*.

## **--wchar\_t\_keyword**

### **Option:**

**--wchar\_t\_keyword**  
**--no\_wchar\_t\_keyword**

### **Default:**

**--wchar\_t\_keyword**

### **Description:**

Enable or disable recognition of `wchar_t` as a keyword.

### **Example:**

```
cpxxx test.cc --no_wchar_t_keyword
```

## **--wrap\_diagnostics**

### **Option:**

**--wrap\_diagnostics**  
**--no\_wrap\_diagnostics**

### **Default:**

**--wrap\_diagnostics**

### **Description:**

Enable or disable a mode in which the error message text is not wrapped when too long to fit on a single line.

### **Example:**

```
cpxxx test.cc --no_wrap_diagnostics
```



**--brief\_diagnostics**

Chapter *Compiler Diagnostics* and Appendix *Error Messages*.

## --xref / -X

### Option:

**-X** *xfile*  
**--xref** *xfile*

### Arguments:

The name of the cross-reference file.

### Description:

Generate cross-reference information in the file *xfile*. For each reference to an identifier in the source program, a line of the form

*symbol\_id name X file-name line-number column-number*

is written, where *X* is

- D** for definition;
- d** for declaration (that is, a declaration that is not a definition);
- M** for modification;
- A** for address taken;
- U** for used;
- C** for changed (but actually meaning used and modified in a single operation, such as an increment);
- R** for any other kind of reference, or
- E** for an error in which the kind of reference is indeterminate.

*symbol-id* is a unique decimal number for the symbol. The fields of the above line are separated by tab characters.



## 3.2 LINKER

The linker used for C++, and mixed language programs, is **ldriver**. It is invoked in the same way as the **llink** linker, with the following differences:

- You must specify the **c++** option
- You must specify the **-cc cxxxxxx** option, where **cxxxxxx** is the name of the appropriate C compiler

If the command line becomes too long, you may find it useful to use the **-opfile <file>** feature. The **-opfile <file>** feature allows you to specify a file containing command-line input for the linker.

A linker for C++ has more to do than a linker for C. **ldriver** handles template instantiation and causes external objects to be constructed and destructed properly. To accomplish its tasks, **ldriver** invokes several programs. These include **llink**, and may also include the C and/or the C++ compilers.

See the C Compiler / Assembler manual for more information about **llink** options which can be used with **ldriver**.

### 3.3 PRAGMAS

According to ANSI (3.8.6) a preprocessing directive of the form:

```
#pragma pragma-token-list new-line
```

causes the compiler to behave in an implementation-defined manner. The compiler ignores pragmas which are not mentioned in the list below. Pragmas give directions to the code generator of the compiler. Besides the pragmas there are two other possibilities to steer the code generator: command line options and keywords. The compiler acknowledges these three groups using the following rule:

Command line options can be overruled by keywords and pragmas. Keywords can be overruled by pragmas. So the pragma has the highest priority.

This approach makes it possible to set a default optimization level for a source module, which can be overridden temporarily within the source by a pragma.

**cpxxx** supports the following pragmas and all pragmas that are described in the *C Compiler/Assembler User's Manual*.

**instantiate**  
**do\_not\_instantiate**  
**can\_instantiate**

These are template instantiation pragmas. They are described in detail in the section *Template Instantiation* in chapter *Language Implementation*.

**hdrstop**  
**no\_pch**

These are precompiled header pragmas. They are described in detail in the section *Precompiled Headers* in chapter *Language Implementation*.

**once**

When placed at the beginning of a header file, indicates that the file is written in such a way that including it several times has the same effect as including it once. Thus, if the C++ compiler sees **#pragma once** at the start of a header file, it will skip over it if the file is **#included** again.

A typical idiom is to place an **#ifndef** guard around the body of the file, with a **#define** of the guard variable after the **#ifndef**:

```
#pragma once      // optional
#ifdef FILE_H
#define FILE_H
... body of the header file ...
#endif
```

The **#pragma once** is marked as optional in this example, because the C++ compiler recognizes the `#ifndef` idiom and does the optimization even in its absence. **#pragma once** is accepted for compatibility with other compilers and to allow the programmer to use other guard-code idioms.

**separate** This pragma gives you complete control over the segmentation of global data.

The compiler creates a “data” group containing all data which is accessed by A5 register–relative addressing. By default, global and static data items are allocated in one of two data segments: “idata”, for initialized data, or “udata”, for uninitialized data, and so are restricted to a total size of 64K bytes. Separate Data Items are placed in their own segments, and can be placed independently in memory using the `locate` function of the linking locator. They are not subject to the total size restriction imposed on typical non-global data.

```
#pragma separate variable_name
                  [segment segname]
                  [class classname]
```

The `#pragma separate` directive causes the data item named *variable\_name* to be separate. The `#pragma separate` directive for *variable\_name* must proceed the definition of *variable\_name* in the source program. The compiler puts this data in the specified class and segment, if that information is included in the statement.

A class is a named collection of segments that share a common logical attribute, such as being executable code or data. We often use the notational convention of bracketing a name in curly braces to indicate that it is a class name. This convention is also accepted within the locator command language. Refer to your C Compiler manual for more information.

The class name of a segment is best thought of as an abstract attribute of that segment; it is a descriptive comment describing the meaning or intended usage of that segment. Users can make up their own class names and assign them whatever significance they want.

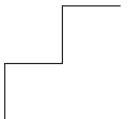


# CHAPTER

# 4

## COMPILER DIAGNOSTICS

---



---

# 4

# CHAPTER

---

## 4.1 DIAGNOSTIC MESSAGES

Diagnostic messages have an associated *severity*, as follows:

- Catastrophic errors, also called 'fatal errors', indicate problems of such severity that the compilation cannot continue. For example: command-line errors, internal errors, and missing include files. If multiple source files are being compiled, any source files after the current one will not be compiled.
- Errors indicate violations of the syntax or semantic rules of the C++ language. Compilation continues, but object code is not generated.
- Warnings indicate something valid but questionable. Compilation continues and object code is generated (if no errors are detected).
- Remarks indicate something that is valid and probably intended, but which a careful programmer may want to check. These diagnostics are not issued by default. Compilation continues and object code is generated (if no errors are detected).

Diagnostics are written to `stderr` with a form like the following:

```
"test.cc", line 5: a break statement may only be used within a loop  
or switch
```



Note that the message identifies the file and line involved.

Long messages are wrapped to additional lines when necessary.

The command line option **--no\_brief\_diagnostics** may be used to request a longer form of the diagnostic output in which the original source line is displayed and the error message text is wrapped when too long to fit on a single line.

The command line option **--display\_error\_number** may be used to request that the error number be included in the diagnostic message. When displayed, the error number also indicates whether the error may have its severity overridden on the command line (with one of the **--diag\_severity** options). If the severity may be overridden, the error number will include the suffix **-D** (for discretionary); otherwise no suffix will be present.

```
"Test_name.cc", line 7: error #64-D: declaration does not  
declare anything
```



```
"Test_name.cc", line 9: error #77: this declaration has no storage
    class or type specifier
```

Because an error is determined to be discretionary based on the error severity associated with a specific context, a given error may be discretionary in some cases and not in others.

For some messages, a list of entities is useful; they are listed following the initial error message:

```
"test.cc", line 4: error: more than one instance of overloaded
    function "f" matches the argument list:
        function "f(int)"
        function "f(float)"
    argument types are: (double)
```

In some cases, some additional context information is provided; specifically, such context information is useful when the C++ compiler issues a diagnostic while doing a template instantiation or while generating a constructor, destructor, or assignment operator function. For example:

```
"test.cc", line 7: error: "A::A()" is inaccessible
    detected during implicit generation of "B::B()" at line
7
```

Without the context information, it is very hard to figure out what the error refers to.



For a list of error messages and error numbers, see Appendix *Error Messages*.

## 4.2 TERMINATION MESSAGES

**cpxxx** writes sign-off messages to `stderr` if errors are detected. For example, one of the following forms of message

```
n errors detected in the compilation of "ifile".
```

```
1 catastrophic error detected in the compilation of "ifile".
```

```
n errors and 1 catastrophic error detected in the compilation of
"ifile".
```

is written to indicate the detection of errors in the compilation. No message is written if no errors were detected.

The message

```
cpxxx: Compilation of 'xxx.cpp' failed, status=x
```

is written at the end of a compilation that was prematurely terminated because of errors.

### **4.3 RESPONSE TO SIGNALS**

The signals `SIGINT` (caused by a user interrupt, like **^C**) and `SIGTERM` (caused by a **kill** command) are trapped by the C++ compiler and cause abnormal termination.

### **4.4 RETURN VALUES**

**cpxxx** returns an exit status to the operating system environment for testing.

*For example,*

in a PC BATCH-file you can examine the exit status of the program executed with `ERRORLEVEL`:

```
cpxxx %1.cc  
IF NOT ERRORLEVEL 1 GOTO STOP_BATCH
```

In a Bourne shell script, the exit status can be found in the `$?` variable, for example:

```
cpxxx test.cpp  
case $? in  
0)          echo ok ;;  
-1|2|4)    echo error ;;  
esac
```

The exit status of **cpxxx** indicates the highest severity diagnostic detected and is one of the numbers of the following list:

- 1 Abnormal termination
- 0 Compilation successful, no errors
- 2 There were user errors, but terminated normally



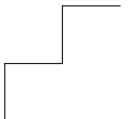
# DIAGNOSTICS

# APPENDIX

## ERROR MESSAGES

---

# A



---

**A**

**APPENDIX**

---

## 1 INTRODUCTION

This appendix lists all diagnostic messages, starting with the error number and the error tag name, followed by the message itself. The error number and/or error tag can be used in **--diag\_severity** options to override the normal error severity.

The C++ compiler produces error messages on standard error output. With the **--error\_output** option you can redirect the error messages to an error list file.

Normally, diagnostics are written to `stderr` in the following form:

*"filename", line line\_num: message*

With **--display\_error\_number** this form will be:

*"filename", line line\_num: severity #err\_num: message*

or:

*"filename", line line\_num: severity #err\_num-D: message*

Where *severity* can be one of: remark, warning, error, catastrophic error, command-line error or internal error.

If the severity may be overridden, the error number will include the suffix **-D** (for discretionary); otherwise no suffix will be present.

In a raw listing file (**-L** option) diagnostic messages have the following layout, starting with the severity (R: remark, W: warning, E: error, C: catastrophe):

[R|W|E|C] *"filename" line\_number column\_number error\_message*



For more detailed information see chapter *Compiler Diagnostics*.

All diagnostic messages are listed below.

## **2 MESSAGES**

- 0001 last\_line\_incomplete:  
last line of file ends without a newline
- 0002 last\_line\_backslash:  
last line of file ends with a backslash
- 0003 include\_recursion:  
#include file "xxxx" includes itself
- 0004 out\_of\_memory:  
out of memory
- 0005 source\_file\_could\_not\_be\_opened:  
could not open source file "xxxx"
- 0006 comment\_unclosed\_at\_eof:  
comment unclosed at end of file
- 0007 bad\_token:  
unrecognized token
- 0008 unclosed\_string:  
missing closing quote
- 0009 nested\_comment:  
nested comment is not allowed
- 0010 bad\_use\_of\_sharp:  
"#" not expected here
- 0011 bad\_pp\_directive\_keyword:  
unrecognized preprocessing directive
- 0012 end\_of\_flush:  
parsing restarts here after previous syntax error
- 0013 exp\_file\_name:  
expected a file name

- 0014 `extra_text_in_pp_directive:`  
extra text after expected end of preprocessing directive
- 0016 `illegal_source_file_name:`  
"xxxx" is not a valid source file name
- 0017 `exp_rbracket:`  
expected a "]"
- 0018 `exp_rparen:`  
expected a ")"
- 0019 `extra_chars_on_number:`  
extra text after expected end of number
- 0020 `undefined_identifier:`  
identifier "xxxx" is undefined
- 0021 `useless_type_qualifiers:`  
type qualifiers are meaningless in this declaration
- 0022 `bad_hex_digit:`  
invalid hexadecimal number
- 0023 `integer_too_large:`  
integer constant is too large
- 0024 `bad_octal_digit:`  
invalid octal digit
- 0025 `zero_length_string:`  
quoted string should contain at least one character
- 0026 `too_many_characters:`  
too many characters in character constant
- 0027 `bad_character_value:`  
character value is out of range
- 0028 `expr_not_constant:`  
expression must have a constant value



- 0029 exp\_primary\_expr:  
expected an expression
- 0030 bad\_float\_value:  
floating constant is out of range
- 0031 expr\_not\_integral:  
expression must have integral type
- 0032 expr\_not\_arithmetic:  
expression must have arithmetic type
- 0033 exp\_line\_number:  
expected a line number
- 0034 bad\_line\_number:  
invalid line number
- 0035 error\_directive:  
#error directive: xxxx
- 0036 missing\_pp\_if:  
the #if for this directive is missing
- 0037 missing\_endif:  
the #endif for this directive is missing
- 0038 pp\_else\_already\_appeared:  
directive is not allowed -- an #else has already appeared
- 0039 divide\_by\_zero:  
division by zero
- 0040 exp\_identifier:  
expected an identifier
- 0041 expr\_not\_scalar:  
expression must have arithmetic or pointer type
- 0042 incompatible\_operands:  
operand types are incompatible ("*type*" and "*type*")

- 0044 `expr_not_pointer:`  
expression must have pointer type
- 0045 `cannot_undef_predef_macro:`  
`#undef` may not be used on this predefined name
- 0046 `cannot_redef_predef_macro:`  
this predefined name may not be redefined
- 0047 `bad_macro_redef:`  
incompatible redefinition of macro *"entity"* (declared at line *xxxx*)
- 0049 `duplicate_macro_param_name:`  
duplicate macro parameter name
- 0050 `paste_cannot_be_first:`  
`"##"` may not be first in a macro definition
- 0051 `paste_cannot_be_last:`  
`"##"` may not be last in a macro definition
- 0052 `exp_macro_param:`  
expected a macro parameter name
- 0053 `exp_colon:`  
expected a `":"`
- 0054 `too_few_macro_args:`  
too few arguments in macro invocation
- 0055 `too_many_macro_args:`  
too many arguments in macro invocation
- 0056 `sizeof_function:`  
operand of `sizeof` may not be a function
- 0057 `bad_constant_operator:`  
this operator is not allowed in a constant expression
- 0058 `bad_pp_operator:`  
this operator is not allowed in a preprocessing expression

- 0059 `bad_constant_function_call`:  
function call is not allowed in a constant expression
- 0060 `bad_integral_operator`:  
this operator is not allowed in an integral constant expression
- 0061 `integer_overflow`:  
integer operation result is out of range
- 0062 `negative_shift_count`:  
shift count is negative
- 0063 `shift_count_too_large`:  
shift count is too large
- 0064 `useless_decl`:  
declaration does not declare anything
- 0065 `exp_semicolon`:  
expected a ";"
- 0066 `enum_value_out_of_int_range`:  
enumeration value is out of "int" range
- 0067 `exp_rbrace`:  
expected a "}"
- 0068 `integer_sign_change`:  
integer conversion resulted in a change of sign
- 0069 `integer_truncated`:  
integer conversion resulted in truncation
- 0070 `incomplete_type_not_allowed`:  
incomplete type is not allowed
- 0071 `sizeof_bit_field`:  
operand of sizeof may not be a bit field
- 0075 `bad_indirection_operand`:  
operand of "\*" must be a pointer

- 0076 `empty_macro_argument`:  
argument to macro is empty
- 0077 `missing_decl_specifiers`:  
this declaration has no storage class or type specifier
- 0078 `initializer_in_param`:  
a parameter declaration may not have an initializer
- 0079 `exp_type_specifier`:  
expected a type specifier
- 0080 `storage_class_not_allowed`:  
a storage class may not be specified here
- 0081 `mult_storage_classes`:  
more than one storage class may not be specified
- 0082 `storage_class_not_first`:  
storage class is not first
- 0083 `dupl_type_qualifier`:  
type qualifier specified more than once
- 0084 `bad_combination_of_type_specifiers`:  
invalid combination of type specifiers
- 0085 `bad_param_storage_class`:  
invalid storage class for a parameter
- 0086 `bad_function_storage_class`:  
invalid storage class for a function
- 0087 `type_specifier_not_allowed`:  
a type specifier may not be used here
- 0088 `array_of_function`:  
array of functions is not allowed
- 0089 `array_of_void`:  
array of void is not allowed

- 0090 `function_returning_function`:  
function returning function is not allowed
- 0091 `function_returning_array`:  
function returning array is not allowed
- 0092 `param_id_list_needs_function_def`:  
identifier-list parameters may only be used in a function definition
- 0093 `function_type_must_come_from_declarator`:  
function type may not come from a typedef
- 0094 `array_size_must_be_positive`:  
the size of an array must be greater than zero
- 0095 `array_size_too_large`:  
array is too large
- 0096 `empty_translation_unit`:  
a translation unit must contain at least one declaration
- 0097 `bad_function_return_type`:  
a function may not return a value of this type
- 0098 `bad_array_element_type`:  
an array may not have elements of this type
- 0099 `decl_should_be_of_param`:  
a declaration here must declare a parameter
- 0100 `dupl_param_name`:  
duplicate parameter name
- 0101 `id_already_declared`:  
"xxxx" has already been declared in the current scope
- 0102 `nonstd_forward_decl_enum`:  
forward declaration of enum type is nonstandard
- 0103 `class_too_large`:  
class is too large

- 0104 `struct_too_large`:  
struct or union is too large
- 0105 `bad_bit_field_size`:  
invalid size for bit field
- 0106 `bad_bit_field_type`:  
invalid type for a bit field
- 0107 `zero_length_bit_field_must_be_unnamed`:  
zero-length bit field must be unnamed
- 0108 `signed_one_bit_field`:  
signed bit field of length 1
- 0109 `expr_not_ptr_to_function`:  
expression must have (pointer-to-) function type
- 0110 `exp_definition_of_tag`:  
expected either a definition or a tag name
- 0111 `code_is_unreachable`:  
statement is unreachable
- 0112 `exp_while`:  
expected "while"
- 0114 `never_defined`:  
*entity-kind "entity"* was referenced but not defined
- 0115 `continue_must_be_in_loop`:  
a continue statement may only be used within a loop
- 0116 `break_must_be_in_loop_or_switch`:  
a break statement may only be used within a loop or switch
- 0117 `no_value_returned_in_non_void_function`:  
non-void *entity-kind "entity"* (declared at line `xxxx`) should return a value

- 0118 `value_returned_in_void_function`:  
a void function may not return a value
- 0119 `cast_to_bad_type`:  
cast to type "*type*" is not allowed
- 0120 `bad_return_value_type`:  
return value type does not match the function type
- 0121 `case_label_must_be_in_switch`:  
a case label may only be used within a switch
- 0122 `default_label_must_be_in_switch`:  
a default label may only be used within a switch
- 0123 `case_label_appears_more_than_once`:  
case label value has already appeared in this switch
- 0124 `default_label_appears_more_than_once`:  
default label has already appeared in this switch
- 0125 `exp_lparen`:  
expected a "("
- 0126 `expr_not_an_lvalue`:  
expression must be an lvalue
- 0127 `exp_statement`:  
expected a statement
- 0128 `loop_not_reachable`:  
loop is not reachable from preceding code
- 0129 `block_scope_function_must_be_extern`:  
a block-scope function may only have extern storage class
- 0130 `exp_lbrace`:  
expected a "{"
- 0131 `expr_not_ptr_to_class`:  
expression must have pointer-to-class type

- 0132 `expr_not_ptr_to_struct_or_union`:  
expression must have pointer-to-struct-or-union type
- 0133 `exp_member_name`:  
expected a member name
- 0134 `exp_field_name`:  
expected a field name
- 0135 `not_a_member`:  
*entity-kind "entity"* has no member *"xxxx"*
- 0136 `not_a_field`:  
*entity-kind "entity"* has no field *"xxxx"*
- 0137 `expr_not_a_modifiable_lvalue`:  
expression must be a modifiable lvalue
- 0138 `address_of_register_variable`:  
taking the address of a register variable is not allowed
- 0139 `address_of_bit_field`:  
taking the address of a bit field is not allowed
- 0140 `too_many_arguments`:  
too many arguments in function call
- 0141 `all_proto_params_must_be_named`:  
unnamed prototyped parameters not allowed when body is present
- 0142 `expr_not_pointer_to_object`:  
expression must have pointer-to-object type
- 0143 `program_too_large`:  
program too large or complicated to compile
- 0144 `bad_initializer_type`:  
a value of type *"type"* cannot be used to initialize an entity of type *"type"*



- 0145 cannot\_initialize:  
*entity-kind "entity"* may not be initialized
- 0146 too\_many\_initializer\_values:  
too many initializer values
- 0147 not\_compatible\_with\_previous\_decl:  
declaration is incompatible with *entity-kind "entity"* (declared at line  
xxxx)
- 0148 already\_initialized:  
*entity-kind "entity"* has already been initialized
- 0149 bad\_file\_scope\_storage\_class:  
a global-scope declaration may not have this storage class
- 0150 type\_cannot\_be\_param\_name:  
a type name may not be redeclared as a parameter
- 0151 typedef\_cannot\_be\_param\_name:  
a typedef name may not be redeclared as a parameter
- 0152 non\_zero\_int\_conv\_to\_pointer:  
conversion of nonzero integer to pointer
- 0153 expr\_not\_class:  
expression must have class type
- 0154 expr\_not\_struct\_or\_union:  
expression must have struct or union type
- 0155 old\_fashioned\_assignment\_operator:  
old-fashioned assignment operator
- 0156 old\_fashioned\_initializer:  
old-fashioned initializer
- 0157 expr\_not\_integral\_constant:  
expression must be an integral constant expression

- 0158 `expr_not_an_lvalue_or_function_designator`:  
expression must be an lvalue or a function designator
- 0159 `decl_incompatible_with_previous_use`:  
declaration is incompatible with previous "*entity*" (declared at line *xxxx*)
- 0160 `external_name_clash`:  
name conflicts with previously used external name "*xxxx*"
- 0161 `unrecognized_pragma`:  
unrecognized `#pragma`
- 0163 `cannot_open_temp_file`:  
could not open temporary file "*xxxx*"
- 0164 `temp_file_dir_name_too_long`:  
name of directory for temporary files is too long ("*xxxx*")
- 0165 `too_few_arguments`:  
too few arguments in function call
- 0166 `bad_float_constant`:  
invalid floating constant
- 0167 `incompatible_param`:  
argument of type "*type*" is incompatible with parameter of type "*type*"
- 0168 `function_type_not_allowed`:  
a function type is not allowed here
- 0169 `exp_declaration`:  
expected a declaration
- 0170 `pointer_outside_base_object`:  
pointer points outside of underlying object
- 0171 `bad_cast`:  
invalid type conversion

- 0172 `linkage_conflict`:  
external/internal linkage conflict with previous declaration
- 0173 `float_to_integer_conversion`:  
floating-point value does not fit in required integral type
- 0174 `expr_has_no_effect`:  
expression has no effect
- 0175 `subscript_out_of_range`:  
subscript out of range
- 0177 `declared_but_not_referenced`:  
*entity-kind "entity"* was declared but never referenced
- 0178 `pcc_address_of_array`:  
"&" applied to an array has no effect
- 0179 `mod_by_zero`:  
right operand of "%" is zero
- 0180 `old_style_incompatible_param`:  
argument is incompatible with formal parameter
- 0181 `printf_arg_mismatch`:  
argument is incompatible with corresponding format string conversion
- 0182 `empty_include_search_path`:  
could not open source file "xxxx" (no directories in search list)
- 0183 `cast_not_integral`:  
type of cast must be integral
- 0184 `cast_not_scalar`:  
type of cast must be arithmetic or pointer
- 0185 `initialization_not_reachable`:  
dynamic initialization in unreachable code

- 0186 unsigned\_compare\_with\_zero:  
pointless comparison of unsigned integer with zero
- 0187 assign\_where\_compare\_meant:  
use of "=" where "==" may have been intended
- 0188 mixed\_enum\_type:  
enumerated type mixed with another type
- 0189 file\_write\_error:  
error while writing xxxx file
- 0190 bad\_il\_file:  
invalid intermediate language file
- 0191 cast\_to\_qualified\_type:  
type qualifier is meaningless on cast type
- 0192 unrecognized\_char\_escape:  
unrecognized character escape sequence
- 0193 undefined\_preproc\_id:  
zero used for undefined preprocessing identifier
- 0194 exp\_asm\_string:  
expected an asm string
- 0195 asm\_func\_must\_be\_prototyped:  
an asm function must be prototyped
- 0196 bad\_asm\_func\_ellipsis:  
an asm function may not have an ellipsis
- 0219 file\_delete\_error:  
error while deleting file "xxxx"
- 0220 integer\_to\_float\_conversion:  
integral value does not fit in required floating-point type
- 0221 float\_to\_float\_conversion:  
floating-point value does not fit in required floating-point type

- 0222 `bad_float_operation_result`:  
floating-point operation result is out of range
- 0223 `implicit_func_decl`:  
function declared implicitly
- 0224 `too_few_printf_args`:  
the format string requires additional arguments
- 0225 `too_many_printf_args`:  
the format string ends before this argument
- 0226 `bad_printf_format_string`:  
invalid format string conversion
- 0227 `macro_recursion`:  
macro recursion
- 0228 `nonstd_extra_comma`:  
trailing comma is nonstandard
- 0229 `enum_bit_field_too_small`:  
bit field cannot contain all values of the enumerated type
- 0230 `nonstd_bit_field_type`:  
nonstandard type for a bit field
- 0231 `decl_in_prototype_scope`:  
declaration is not visible outside of function
- 0232 `decl_of_void_ignored`:  
old-fashioned typedef of "void" ignored
- 0233 `old_fashioned_field_selection`:  
left operand is not a struct or union containing this field
- 0234 `old_fashioned_ptr_field_selection`:  
pointer does not point to struct or union containing this field
- 0235 `var_retained_incomp_type`:  
variable "xxxx" was declared with a never-completed type

- 0236 `boolean_controlling_expr_is_constant`:  
controlling expression is constant
- 0237 `switch_selector_expr_is_constant`:  
selector expression is constant
- 0238 `bad_param_specifier`:  
invalid specifier on a parameter
- 0239 `bad_specifier_outside_class_decl`:  
invalid specifier outside a class declaration
- 0240 `dupl_decl_specifier`:  
duplicate specifier in declaration
- 0241 `base_class_not_allowed_for_union`:  
a union is not allowed to have a base class
- 0242 `access_already_specified`:  
multiple access control specifiers are not allowed
- 0243 `missing_class_definition`:  
class or struct definition is missing
- 0244 `name_not_member_of_class_or_base_classes`:  
qualified name is not a member of class "*type*" or its base classes
- 0245 `member_ref_requires_object`:  
a nonstatic member reference must be relative to a specific object
- 0246 `nonstatic_member_def_not_allowed`:  
a nonstatic data member may not be defined outside its class
- 0247 `already_defined`:  
*entity-kind "entity"* has already been defined
- 0248 `pointer_to_reference`:  
pointer to reference is not allowed
- 0249 `reference_to_reference`:  
reference to reference is not allowed

- 0250 reference\_to\_void:  
reference to void is not allowed
- 0251 array\_of\_reference:  
array of reference is not allowed
- 0252 missing\_initializer\_on\_reference:  
reference *entity-kind* "entity" requires an initializer
- 0253 exp\_comma:  
expected a ","
- 0254 type\_identifier\_not\_allowed:  
type name is not allowed
- 0255 type\_definition\_not\_allowed:  
type definition is not allowed
- 0256 bad\_type\_name\_redeclaration:  
invalid redeclaration of type name "entity" (declared at line .xxxx)
- 0257 missing\_initializer\_on\_const:  
const *entity-kind* "entity" requires an initializer
- 0258 this\_used\_incorrectly:  
"this" may only be used inside a nonstatic member function
- 0259 constant\_value\_not\_known:  
constant value is not known
- 0260 missing\_type\_specifier:  
explicit type is missing ("int" assumed)
- 0261 missing\_access\_specifier:  
access control not specified ("xxxx" by default)
- 0262 not\_a\_class\_or\_struct\_name:  
not a class or struct name
- 0263 dupl\_base\_class\_name:  
duplicate base class name

- 0264 `bad_base_class`:  
invalid base class
- 0265 `no_access_to_name`:  
*entity-kind* "entity" is inaccessible
- 0266 `ambiguous_name`:  
"entity" is ambiguous
- 0267 `old_style_parameter_list`:  
old-style parameter list (anachronism)
- 0268 `declaration_after_statements`:  
declaration may not appear after executable statement in block
- 0269 `inaccessible_base_class`:  
implicit conversion to inaccessible base class "type" is not allowed
- 0274 `improperly_terminated_macro_call`:  
improperly terminated macro invocation
- 0276 `id_must_be_class_or_namespace_name`:  
name followed by "::" must be a class or namespace name
- 0277 `bad_friend_decl`:  
invalid friend declaration
- 0278 `value_returned_in_constructor`:  
a constructor or destructor may not return a value
- 0279 `bad_destructor_decl`:  
invalid destructor declaration
- 0280 `class_and_member_name_conflict`:  
invalid declaration of a member with the same name as its class
- 0281 `global_qualifier_not_allowed`:  
global-scope qualifier (leading "::") is not allowed
- 0282 `name_not_found_in_file_scope`:  
the global scope has no "xxx"



- 0283 `qualified_name_not_allowed`:  
qualified name is not allowed
- 0284 `null_reference`:  
NULL reference is not allowed
- 0285 `brace_initialization_not_allowed`:  
initialization with "{...}" is not allowed for object of type "*type*"
- 0286 `ambiguous_base_class`:  
base class "*type*" is ambiguous
- 0287 `ambiguous_derived_class`:  
derived class "*type*" contains more than one instance of class "*type*"
- 0288 `derived_class_from_virtual_base`:  
cannot convert pointer to base class "*type*" to pointer to derived class "*type*" -- base class is virtual
- 0289 `no_matching_constructor`:  
no instance of constructor "*entity*" matches the argument list
- 0290 `ambiguous_copy_constructor`:  
copy constructor for class "*type*" is ambiguous
- 0291 `no_default_constructor`:  
no default constructor exists for class "*type*"
- 0292 `not_a_field_or_base_class`:  
"*xxxx*" is not a nonstatic data member or base class of class "*type*"
- 0293 `indirect_nonvirtual_base_class_not_allowed`:  
indirect nonvirtual base class is not allowed
- 0294 `bad_union_field`:  
invalid union member -- class "*type*" has a disallowed member function
- 0296 `bad_rvalue_array`:  
invalid use of non-lvalue array

- 0297 `exp_operator`:  
expected an operator
- 0298 `inherited_member_not_allowed`:  
inherited member is not allowed
- 0299 `indeterminate_overloaded_function`:  
cannot determine which instance of *entity-kind "entity"* is intended
- 0300 `bound_function_must_be_called`:  
a pointer to a bound function may only be used to call the function
- 0301 `duplicate_typedef`:  
typedef name has already been declared (with same type)
- 0302 `function_redefinition`:  
*entity-kind "entity"* has already been defined
- 0304 `no_matching_function`:  
no instance of *entity-kind "entity"* matches the argument list
- 0305 `type_def_not_allowed_in_func_type_decl`:  
type definition is not allowed in function return type declaration
- 0306 `default_arg_not_at_end`:  
default argument not at end of parameter list
- 0307 `default_arg_already_defined`:  
redefinition of default argument
- 0308 `ambiguous_overloaded_function`:  
more than one instance of *entity-kind "entity"* matches the argument list:
- 0309 `ambiguous_constructor`:  
more than one instance of constructor *"entity"* matches the argument list:
- 0310 `bad_default_arg_type`:  
default argument of type *"type"* is incompatible with parameter of type *"type"*

- 0311 `return_type_cannot_distinguish_functions`:  
cannot overload functions distinguished by return type alone
- 0312 `no_user_defined_conversion`:  
no suitable user-defined conversion from "*type*" to "*type*" exists
- 0313 `function_qualifier_not_allowed`:  
type qualifier is not allowed on this function
- 0314 `virtual_static_not_allowed`:  
only nonstatic member functions may be virtual
- 0315 `unqual_function_with_qual_object`:  
the object has type qualifiers that are not compatible with the member function
- 0316 `too_many_virtual_functions`:  
program too large to compile (too many virtual functions)
- 0317 `bad_return_type_on_virtual_function_override`:  
return type is not identical to nor covariant with return type "*type*" of overridden virtual function *entity-kind* "*entity*"
- 0318 `ambiguous_virtual_function_override`:  
override of virtual *entity-kind* "*entity*" is ambiguous
- 0319 `pure_specifier_on_nonvirtual_function`:  
pure specifier ("= 0") allowed only on virtual functions
- 0320 `bad_pure_specifier`:  
badly-formed pure specifier (only "= 0" is allowed)
- 0321 `bad_data_member_initialization`:  
data member initializer is not allowed
- 0322 `abstract_class_object_not_allowed`:  
object of abstract class type "*type*" is not allowed:
- 0323 `function_returning_abstract_class`:  
function returning abstract class "*type*" is not allowed:

- 0324 `duplicate_friend_decl:`  
duplicate friend declaration
- 0325 `inline_and_nonfunction:`  
inline specifier allowed on function declarations only
- 0326 `inline_not_allowed:`  
"inline" is not allowed
- 0327 `bad_storage_class_with_inline:`  
invalid storage class for an inline function
- 0328 `bad_member_storage_class:`  
invalid storage class for a class member
- 0329 `local_class_function_def_missing:`  
local class member *entity-kind* "entity" requires a definition
- 0330 `inaccessible_special_function:`  
*entity-kind* "entity" is inaccessible
- 0332 `missing_const_copy_constructor:`  
class "type" has no copy constructor to copy a const object
- 0333 `definition_of_implicitly_declared_function:`  
defining an implicitly declared member function is not allowed
- 0334 `no_suitable_copy_constructor:`  
class "type" has no suitable copy constructor
- 0335 `linkage_specifier_not_allowed:`  
linkage specification is not allowed
- 0336 `bad_linkage_specifier:`  
unknown external linkage specification
- 0337 `incompatible_linkage_specifier:`  
linkage specification is incompatible with previous "entity"  
(declared at line xxx)

- 0338 overloaded\_function\_linkage:  
more than one instance of overloaded function "*entity*" has "C" linkage
- 0339 ambiguous\_default\_constructor:  
class "*type*" has more than one default constructor
- 0340 temp\_used\_for\_ref\_init:  
value copied to temporary, reference to temporary used
- 0341 nonmember\_operator\_not\_allowed:  
"operator:xxxx" must be a member function
- 0342 static\_member\_operator\_not\_allowed:  
operator may not be a static member function
- 0343 too\_many\_args\_for\_conversion:  
no arguments allowed on user-defined conversion
- 0344 too\_many\_args\_for\_operator:  
too many parameters for this operator function
- 0345 too\_few\_args\_for\_operator:  
too few parameters for this operator function
- 0346 no\_params\_with\_class\_type:  
nonmember operator requires a parameter with class type
- 0347 default\_arg\_expr\_not\_allowed:  
default argument is not allowed
- 0348 ambiguous\_user\_defined\_conversion:  
more than one user-defined conversion from "*type*" to "*type*" applies:
- 0349 no\_matching\_operator\_function:  
no operator "xxxx" matches these operands
- 0350 ambiguous\_operator\_function:  
more than one operator "xxxx" matches these operands:

- 0351 `bad_arg_type_for_operator_new`:  
first parameter of allocation function must be of type "size\_t"
- 0352 `bad_return_type_for_op_new`:  
allocation function requires "void \*" return type
- 0353 `bad_return_type_for_op_delete`:  
deallocation function requires "void" return type
- 0354 `bad_first_arg_type_for_operator_delete`:  
first parameter of deallocation function must be of type "void \*"
- 0356 `type_must_be_object_type`:  
type must be an object type
- 0357 `base_class_already_initialized`:  
base class "*type*" has already been initialized
- 0358 `base_class_init_anachronism`:  
base class name required -- "*type*" assumed (anachronism)
- 0359 `member_already_initialized`:  
*entity-kind* "*entity*" has already been initialized
- 0360 `missing_base_class_or_member_name`:  
name of member or base class is missing
- 0361 `assignment_to_this`:  
assignment to "this" (anachronism)
- 0362 `overload_anachronism`:  
"overload" keyword used (anachronism)
- 0363 `anon_union_member_access`:  
invalid anonymous union -- nonpublic member is not allowed
- 0364 `anon_union_member_function`:  
invalid anonymous union -- member function is not allowed

- 0365 `anon_union_storage_class`:  
anonymous union at global or namespace scope must be declared static
- 0366 `missing_initializer_on_fields`:  
*entity-kind "entity"* provides no initializer for:
- 0367 `cannot_initialize_fields`:  
implicitly generated constructor for class *"type"* cannot initialize:
- 0368 `no_ctor_but_const_or_ref_member`:  
*entity-kind "entity"* defines no constructor to initialize the following:
- 0369 `var_with_uninitialized_member`:  
*entity-kind "entity"* has an uninitialized const or reference member
- 0370 `var_with_uninitialized_field`:  
*entity-kind "entity"* has an uninitialized const field
- 0371 `missing_const_assignment_operator`:  
class *"type"* has no assignment operator to copy a const object
- 0372 `no_suitable_assignment_operator`:  
class *"type"* has no suitable assignment operator
- 0373 `ambiguous_assignment_operator`:  
ambiguous assignment operator for class *"type"*
- 0375 `missing_typedef_name`:  
declaration requires a typedef name
- 0377 `virtual_not_allowed`:  
"virtual" is not allowed
- 0378 `static_not_allowed`:  
"static" is not allowed
- 0379 `bound_function_cast_anachronism`:  
cast of bound function to normal function pointer (anachronism)

- 0380 `expr_not_ptr_to_member`:  
expression must have pointer-to-member type
- 0381 `extra_semicolon`:  
extra ";" ignored
- 0382 `nonstd_const_member`:  
nonstandard member constant declaration (standard form is a static const integral member)
- 0384 `no_matching_new_function`:  
no instance of overloaded "*entity*" matches the argument list
- 0386 `no_match_for_addr_of_overloaded_function`:  
no instance of *entity-kind* "*entity*" matches the required type
- 0387 `delete_count_anachronism`:  
delete array size expression used (anachronism)
- 0388 `bad_return_type_for_op_arrow`:  
"operator->" for class "*type*" returns invalid type "*type*"
- 0389 `cast_to_abstract_class`:  
a cast to abstract class "*type*" is not allowed:
- 0390 `bad_use_of_main`:  
function "main" may not be called or have its address taken
- 0391 `initializer_not_allowed_on_array_new`:  
a new-initializer may not be specified for an array
- 0392 `member_function_redecl_outside_class`:  
member function "*entity*" may not be redeclared outside its class
- 0393 `ptr_to_incomplete_class_type_not_allowed`:  
pointer to incomplete class type is not allowed
- 0394 `ref_to_nested_function_var`:  
reference to local variable of enclosing function is not allowed



- 0395 `single_arg_postfix_incr_decr_anachronism`:  
single-argument function used for postfix `"xxxx"` (anachronism)
- 0397 `bad_default_assignment`:  
implicitly generated assignment operator cannot copy:
- 0398 `nonstd_array_cast`:  
cast to array type is nonstandard (treated as cast to *"type"*)
- 0399 `class_with_op_new_but_no_op_delete`:  
*entity-kind "entity"* has an operator `newxxxx()` but no default operator `deletexxxx()`
- 0400 `class_with_op_delete_but_no_op_new`:  
*entity-kind "entity"* has a default operator `deletexxxx()` but no operator `newxxxx()`
- 0401 `base_class_with_nonvirtual_dtor`:  
destructor for base class *"type"* is not virtual
- 0403 `member_function_redeclaration`:  
*entity-kind "entity"* has already been declared
- 0404 `inline_main`:  
function `"main"` may not be declared inline
- 0405 `class_and_member_function_name_conflict`:  
member function with the same name as its class must be a constructor
- 0406 `nested_class_anachronism`:  
using nested *entity-kind "entity"* (anachronism)
- 0407 `too_many_params_for_destructor`:  
a destructor may not have parameters
- 0408 `bad_constructor_param`:  
copy constructor for class *"type"* may not have a parameter of type *"type"*

- 0409 incomplete\_function\_return\_type:  
*entity-kind "entity"* returns incomplete type *"type"*
- 0410 protected\_access\_problem:  
protected *entity-kind "entity"* is not accessible through a *"type"* pointer or object
- 0411 param\_not\_allowed:  
a parameter is not allowed
- 0412 asm\_decl\_not\_allowed:  
an "asm" declaration is not allowed here
- 0413 no\_conversion\_function:  
no suitable conversion function from *"type"* to *"type"* exists
- 0414 delete\_of\_incomplete\_class:  
delete of pointer to incomplete class
- 0415 no\_constructor\_for\_conversion:  
no suitable constructor exists to convert from *"type"* to *"type"*
- 0416 ambiguous\_constructor\_for\_conversion:  
more than one constructor applies to convert from *"type"* to *"type"*:
- 0417 ambiguous\_conversion\_function:  
more than one conversion function from *"type"* to *"type"* applies:
- 0418 ambiguous\_conversion\_to\_builtin:  
more than one conversion function from *"type"* to a built-in type applies:
- 0424 addr\_of\_constructor\_or\_destructor:  
a constructor or destructor may not have its address taken
- 0425 dollar\_used\_in\_identifier:  
dollar sign ("\$\$") used in identifier
- 0426 nonconst\_ref\_init\_anachronism:  
temporary used for initial value of reference to non-const (anachronism)

- 0427 `qualifier_in_member_declaration`:  
qualified name is not allowed in member declaration
- 0428 `mixed_enum_type_anachronism`:  
enumerated type mixed with another type (anachronism)
- 0429 `new_array_size_must_be_nonnegative`:  
the size of an array in "new" must be non-negative
- 0430 `return_ref_init_requires_temp`:  
returning reference to local temporary
- 0432 `enum_not_allowed`:  
"enum" declaration is not allowed
- 0433 `qualifier_dropped_in_ref_init`:  
qualifiers dropped in binding reference of type "*type*" to initializer of type "*type*"
- 0434 `bad_nonconst_ref_init`:  
a reference of type "*type*" (not const-qualified) cannot be initialized with a value of type "*type*"
- 0435 `delete_of_function_pointer`:  
a pointer to function may not be deleted
- 0436 `bad_conversion_function_decl`:  
conversion function must be a nonstatic member function
- 0437 `bad_template_declaration_scope`:  
template declaration is not allowed here
- 0438 `exp_lt`:  
expected a "<"
- 0439 `exp_gt`:  
expected a ">"
- 0440 `missing_template_param`:  
template parameter declaration is missing

- 0441 `missing_template_arg_list`:  
argument list for *entity-kind* "entity" is missing
- 0442 `too_few_template_args`:  
too few arguments for *entity-kind* "entity"
- 0443 `too_many_template_args`:  
too many arguments for *entity-kind* "entity"
- 0445 `not_used_in_template_function_params`:  
*entity-kind* "entity" is not used in declaring the parameter types of  
*entity-kind* "entity"
- 0446 `cfront_multiple_nested_types`:  
two nested types have the same name: "entity" and "entity"  
(declared at line .xxxx) (cfront compatibility)
- 0447 `cfront_global_defined_after_nested_type`:  
global "entity" was declared after nested "entity" (declared at line  
.xxxx) (cfront compatibility)
- 0449 `ambiguous_ptr_to_overloaded_function`:  
more than one instance of *entity-kind* "entity" matches the required  
type
- 0450 `nonstd_long_long`:  
the type "long long" is nonstandard
- 0451 `nonstd_friend_decl`:  
omission of "xxxx" is nonstandard
- 0452 `return_type_on_conversion_function`:  
return type may not be specified on a conversion function
- 0456 `runaway_recursive_instantiation`:  
excessive recursion at instantiation of *entity-kind* "entity"
- 0457 `bad_template_declaration`:  
"xxxx" is not a function or static data member

- 0458 `bad_nontype_template_arg`:  
argument of type "*type*" is incompatible with template parameter of type "*type*"
- 0459 `init_needing_temp_not_allowed`:  
initialization requiring a temporary or conversion is not allowed
- 0460 `decl_hides_function_parameter`:  
declaration of "xxxx" hides function parameter
- 0461 `nonconst_ref_init_from_rvalue`:  
initial value of reference to non-const must be an lvalue
- 0463 `template_not_allowed`:  
"template" is not allowed
- 0464 `not_a_class_template`:  
"*type*" is not a class template
- 0466 `function_template_named_main`:  
"main" is not a valid name for a function template
- 0467 `union_nonunion_mismatch`:  
invalid reference to *entity-kind* "*entity*" (union/nonunion mismatch)
- 0468 `local_type_in_template_arg`:  
a template argument may not reference a local type
- 0469 `tag_kind_incompatible_with_declaration`:  
tag kind of xxxx is incompatible with declaration of *entity-kind* "*entity*" (declared at line xxxx)
- 0470 `name_not_tag_in_file_scope`:  
the global scope has no tag named "xxxx"
- 0471 `not_a_tag_member`:  
*entity-kind* "*entity*" has no tag member named "xxxx"
- 0472 `ptr_to_member_typedef`:  
member function typedef (allowed for cfront compatibility)

- 0473 `bad_use_of_member_function_typedef`:  
*entity-kind "entity"* may be used only in pointer-to-member declaration
- 0475 `nonexternal_entity_in_template_arg`:  
a template argument may not reference a non-external entity
- 0476 `id_must_be_class_or_type_name`:  
name followed by "::~" must be a class name or a type name
- 0477 `destructor_name_mismatch`:  
destructor name does not match name of class *"type"*
- 0478 `destructor_type_mismatch`:  
type used as destructor name does not match type *"type"*
- 0479 `called_function_redeclared_inline`:  
*entity-kind "entity"* redeclared "inline" after being called
- 0481 `bad_storage_class_on_template_decl`:  
invalid storage class for a template declaration
- 0482 `no_access_to_type_cfront_mode`:  
*entity-kind "entity"* is an inaccessible type (allowed for cfront compatibility)
- 0483 `return_type_not_allowed`:  
a return type is not allowed
- 0484 `invalid_instantiation_argument`:  
invalid explicit instantiation declaration
- 0485 `not_instantiatable_entity`:  
*entity-kind "entity"* is not an entity that can be instantiated
- 0486 `compiler_generated_function_cannot_be_instantiated`:  
compiler generated *entity-kind "entity"* cannot be explicitly instantiated
- 0487 `inline_function_cannot_be_instantiated`:  
inline *entity-kind "entity"* cannot be explicitly instantiated

- 0488 pure\_virtual\_function\_cannot\_be\_instantiated:  
pure virtual *entity-kind "entity"* cannot be explicitly instantiated
- 0489 instantiation\_requested\_no\_definition\_supplied:  
*entity-kind "entity"* cannot be instantiated -- no template definition was supplied
- 0490 instantiation\_requested\_and\_specialized:  
*entity-kind "entity"* cannot be instantiated -- it has been explicitly specialized
- 0491 no\_constructor:  
class *"type"* has no constructor
- 0493 no\_match\_for\_type\_of\_overloaded\_function:  
no instance of *entity-kind "entity"* matches the specified type
- 0494 nonstd\_void\_param\_list:  
declaring a void parameter list with a typedef is nonstandard
- 0495 cfront\_name\_lookup\_bug:  
global *entity-kind "entity"* used instead of *entity-kind "entity"* (cfront compatibility)
- 0496 redeclaration\_of\_template\_param\_name:  
template parameter *"xxxx"* may not be redeclared in this scope
- 0497 decl\_hides\_template\_parameter:  
declaration of *"xxxx"* hides template parameter
- 0498 must\_be\_prototype\_instantiation:  
template argument list must match the parameter list
- 0499 conversion\_to\_type\_not\_allowed:  
conversion function to convert from *"type"* to *"type"* is not allowed
- 0500 bad\_extra\_arg\_for\_postfix\_operator:  
extra parameter of postfix *"operatorxxxx"* must be of type *"int"*
- 0501 function\_type\_required:  
an operator name must be declared as a function

- 0502 operator\_name\_not\_allowed:  
operator name is not allowed
- 0503 bad\_scope\_for\_specialization:  
*entity-kind "entity"* cannot be specialized in the current scope
- 0504 nonstd\_member\_function\_address:  
nonstandard form for taking the address of a member function
- 0505 too\_few\_template\_params:  
too few template parameters -- does not match previous declaration
- 0506 too\_many\_template\_params:  
too many template parameters -- does not match previous declaration
- 0507 template\_operator\_delete:  
function template for operator delete(void \*) is not allowed
- 0508 class\_template\_same\_name\_as\_tmpl\_param:  
class template and template parameter may not have the same name
- 0510 unnamed\_type\_in\_template\_arg:  
a template argument may not reference an unnamed type
- 0511 enum\_type\_not\_allowed:  
enumerated type is not allowed
- 0512 qualified\_reference\_type:  
type qualifier on a reference type is not allowed
- 0513 incompatible\_assignment\_operands:  
a value of type *"type"* cannot be assigned to an entity of type *"type"*
- 0514 unsigned\_compare\_with\_negative:  
pointless comparison of unsigned integer with a negative constant
- 0515 converting\_to\_incomplete\_class:  
cannot convert to incomplete class *"type"*



- 0516 `missing_initializer_on_unnamed_const`:  
const object requires an initializer
- 0517 `unnamed_object_with_uninitialized_field`:  
object has an uninitialized const or reference member
- 0518 `nonstd_pp_directive`:  
nonstandard preprocessing directive
- 0519 `unexpected_template_arg_list`:  
*entity-kind* "*entity*" may not have a template argument list
- 0520 `missing_initializer_list`:  
initialization with "{...}" expected for aggregate object
- 0521 `incompatible_ptr_to_member_selection_operands`:  
pointer-to-member selection class types are incompatible ("*type*" and "*type*")
- 0522 `self_friendship`:  
pointless friend declaration
- 0523 `period_used_as_qualifier`:  
"." used in place of "::" to form a qualified name (cfront anachronism)
- 0524 `const_function_anachronism`:  
non-const function called for const object (anachronism)
- 0525 `dependent_stmt_is_declaration`:  
a dependent statement may not be a declaration
- 0526 `void_param_not_allowed`:  
a parameter may not have void type
- 0529 `bad_tmpl_arg_expr_operator`:  
this operator is not allowed in a template argument expression
- 0530 `missing_handler`:  
try block requires at least one handler

- 0531 `missing_exception_declaration`:  
handler requires an exception declaration
- 0532 `masked_by_default_handler`:  
handler is masked by default handler
- 0533 `masked_by_handler`:  
handler is potentially masked by previous handler for type *"type"*
- 0534 `local_type_used_in_exception`:  
use of a local type to specify an exception
- 0535 `redundant_exception_specification_type`:  
redundant type in exception specification
- 0536 `incompatible_exception_specification`:  
exception specification is incompatible with that of previous  
*entity-kind "entity"* (declared at line *xxxx*):
- 0540 `no_exception_support`:  
support for exception handling is disabled
- 0541 `omitted_exception_specification`:  
omission of exception specification is incompatible with previous  
*entity-kind "entity"* (declared at line *xxxx*)
- 0542 `cannot_create_instantiation_request_file`:  
could not create instantiation request file *"xxxx"*
- 0543 `non_arith_operation_in_tmpl_arg`:  
non-arithmetic operation not allowed in nontype template argument
- 0544 `local_type_in_nonlocal_var`:  
use of a local type to declare a nonlocal variable
- 0545 `local_type_in_function`:  
use of a local type to declare a function
- 0546 `branch_past_initialization`:  
transfer of control bypasses initialization of:

- 0548 `branch_into_handler`:  
transfer of control into an exception handler
- 0549 `used_before_set`:  
*entity-kind "entity"* is used before its value is set
- 0550 `set_but_not_used`:  
*entity-kind "entity"* was set but never used
- 0551 `bad_scope_for_definition`:  
*entity-kind "entity"* cannot be defined in the current scope
- 0552 `exception_specification_not_allowed`:  
exception specification is not allowed
- 0553 `template_and_instance_linkage_conflict`:  
external/internal linkage conflict for *entity-kind "entity"* (declared at line `xxxx`)
- 0554 `conversion_function_not_usable`:  
*entity-kind "entity"* will not be called for implicit or explicit conversions
- 0555 `tag_kind_incompatible_with_template_parameter`:  
tag kind of `xxxx` is incompatible with template parameter of type *"type"*
- 0556 `template_operator_new`:  
function template for operator `new(size_t)` is not allowed
- 0558 `bad_member_type_in_ptr_to_member`:  
pointer to member of type *"type"* is not allowed
- 0559 `ellipsis_on_operator_function`:  
ellipsis is not allowed in operator function parameter list
- 0560 `unimplemented_keyword`:  
*"entity"* is reserved for future use as a keyword
- 0561 `cl_invalid_macro_definition`:  
invalid macro definition:

- 0562 `cl_invalid_macro_undefinition:`  
invalid macro undefinition:
- 0563 `cl_invalid_preprocessor_output_file:`  
invalid preprocessor output file
- 0564 `cl_cannot_open_preprocessor_output_file:`  
cannot open preprocessor output file
- 0565 `cl_il_file_must_be_specified:`  
IL file name must be specified if input is
- 0566 `cl_invalid_il_output_file:`  
invalid IL output file
- 0567 `cl_cannot_open_il_output_file:`  
cannot open IL output file
- 0568 `cl_invalid_C_output_file:`  
invalid C output file
- 0569 `cl_cannot_open_C_output_file:`  
cannot open C output file
- 0570 `cl_error_in_debug_option_argument:`  
error in debug option argument
- 0571 `cl_invalid_option:`  
invalid option:
- 0572 `cl_back_end_requires_il_file:`  
back end requires name of IL file
- 0573 `cl_could_not_open_il_file:`  
could not open IL file
- 0574 `cl_invalid_number:`  
invalid number:
- 0575 `cl_incorrect_host_id:`  
incorrect host CPU id

- 0576 `cl_invalid_instantiation_mode:`  
invalid instantiation mode:
- 0578 `cl_invalid_error_limit:`  
invalid error limit:
- 0579 `cl_invalid_raw_listing_output_file:`  
invalid raw-listing output file
- 0580 `cl_cannot_open_raw_listing_output_file:`  
cannot open raw-listing output file
- 0581 `cl_invalid_xref_output_file:`  
invalid cross-reference output file
- 0582 `cl_cannot_open_xref_output_file:`  
cannot open cross-reference output file
- 0583 `cl_invalid_error_output_file:`  
invalid error output file
- 0584 `cl_cannot_open_error_output_file:`  
cannot open error output file
- 0585 `cl_vtbl_option_only_in_cplusplus:`  
virtual function tables can only be suppressed when compiling C++
- 0586 `cl_anachronism_option_only_in_cplusplus:`  
anachronism option can be used only when compiling C++
- 0587 `cl_instantiation_option_only_in_cplusplus:`  
instantiation mode option can be used only when compiling C++
- 0588 `cl_auto_instantiation_option_only_in_cplusplus:`  
automatic instantiation mode can be used only when compiling C++
- 0589 `cl_implicit_inclusion_option_only_in_cplusplus:`  
implicit template inclusion mode can be used only when compiling C++

- 0590 `cl_exceptions_option_only_in_cplusplus`:  
exception handling option can be used only when compiling C++
- 0591 `cl_strict_ansi_incompatible_with_pcc`:  
strict ANSI mode is incompatible with K&R mode
- 0592 `cl_strict_ansi_incompatible_with_cfront`:  
strict ANSI mode is incompatible with cfront mode
- 0593 `cl_missing_source_file_name`:  
missing source file name
- 0594 `cl_output_file_incompatible_with_multiple_inputs`:  
output files may not be specified when compiling several input files
- 0595 `cl_too_many_arguments`:  
too many arguments on command line
- 0596 `cl_no_output_file_needed`:  
an output file was specified, but none is needed
- 0597 `cl_il_display_requires_il_file_name`:  
IL display requires name of IL file
- 0598 `void_template_parameter`:  
a template parameter may not have void type
- 0599 `too_many_unused_instantiations`:  
excessive recursive instantiation of *entity-kind "entity"* due to  
instantiate-all mode
- 0600 `cl_strict_ansi_incompatible_with_anachronisms`:  
strict ANSI mode is incompatible with allowing anachronisms
- 0601 `void_throw`:  
a throw expression may not have void type
- 0602 `cl_tim_local_conflicts_with_auto_instantiation`:  
local instantiation mode is incompatible with automatic instantiation

- 0603 `abstract_class_param_type`:  
parameter of abstract class type "*type*" is not allowed:
- 0604 `array_of_abstract_class`:  
array of abstract class "*type*" is not allowed:
- 0605 `float_template_parameter`:  
floating-point template parameter is nonstandard
- 0606 `pragma_must_precede_declaration`:  
this pragma must immediately precede a declaration
- 0607 `pragma_must_precede_statement`:  
this pragma must immediately precede a statement
- 0608 `pragma_must_precede_decl_or_stmt`:  
this pragma must immediately precede a declaration or statement
- 0609 `pragma_may_not_be_used_here`:  
this kind of pragma may not be used here
- 0610 `nonoverriding_function_decl`:  
*entity-kind* "*entity*" does not match "*entity*" -- virtual function  
override intended?
- 0611 `partial_override`:  
overloaded virtual function "*entity*" is only partially overridden in  
*entity-kind* "*entity*"
- 0612 `specialization_of_called_inline_template_function`:  
specific definition of inline template function must precede its first  
use
- 0613 `cl_invalid_error_tag`:  
invalid error tag:
- 0614 `cl_invalid_error_number`:  
invalid error number:
- 0615 `param_type_ptr_to_array_of_unknown_bound`:  
parameter type involves pointer to array of unknown bound

- 0616 `param_type_ref_array_of_unknown_bound`:  
parameter type involves reference to array of unknown bound
- 0617 `ptr_to_member_cast_to_ptr_to_function`:  
pointer-to-member-function cast to pointer to function
- 0618 `no_named_fields`:  
struct or union declares no named members
- 0619 `nonstd_unnamed_field`:  
nonstandard unnamed field
- 0620 `nonstd_unnamed_member`:  
nonstandard unnamed member
- 0622 `cl_invalid_pch_output_file`:  
invalid precompiled header output file
- 0623 `cl_cannot_open_pch_output_file`:  
cannot open precompiled header output file
- 0624 `not_a_type_name`:  
"xxxx" is not a type name
- 0625 `cl_cannot_open_pch_input_file`:  
cannot open precompiled header input file
- 0626 `invalid_pch_file`:  
precompiled header file "xxxx" is either invalid or not generated by  
this version of the compiler
- 0627 `pch_curr_directory_changed`:  
precompiled header file "xxxx" was not generated in this directory
- 0628 `pch_header_files_have_changed`:  
header files used to generate precompiled header file "xxxx" have  
changed
- 0629 `pch_cmd_line_option_mismatch`:  
the command line options do not match those used when  
precompiled header file "xxxx" was created



- 0630 pch\_file\_prefix\_mismatch:  
the initial sequence of preprocessing directives is not compatible  
with those of precompiled header file "xxxxx"
- 0631 unable\_to\_get\_mapped\_memory:  
unable to obtain mapped memory
- 0632 using\_pch:  
"xxxxx": using precompiled header file "xxxxx"
- 0633 creating\_pch:  
"xxxxx": creating precompiled header file "xxxxx"
- 0634 memory\_mismatch:  
memory usage conflict with precompiled header file "xxxxx"
- 0635 cl\_invalid\_pch\_size:  
invalid PCH memory size
- 0636 cl\_pch\_must\_be\_first:  
PCH options must appear first in the command line
- 0637 out\_of\_memory\_during\_pch\_allocation:  
insufficient memory for PCH memory allocation
- 0638 cl\_pch\_incompatible\_with\_multiple\_inputs:  
precompiled header files may not be used when compiling several  
input files
- 0639 not\_enough\_preallocated\_memory:  
insufficient preallocated memory for generation of precompiled  
header file (xxxxx bytes required)
- 0640 program\_entity\_too\_large\_for\_pch:  
very large entity in program prevents generation of precompiled  
header file
- 0641 cannot\_chdir:  
"xxxxx" is not a valid directory

- 0642 `cannot_build_temp_file_name:`  
cannot build temporary file name
- 0643 `restrict_not_allowed:`  
"restrict" is not allowed
- 0644 `restrict_pointer_to_function:`  
a pointer or reference to function type may not be qualified by "restrict"
- 0645 `bad_declspec_modifier:`  
"xxxx" is an unrecognized `__declspec` attribute
- 0646 `calling_convention_not_allowed:`  
a calling convention modifier may not be specified here
- 0647 `conflicting_calling_conventions:`  
conflicting calling convention modifiers
- 0648 `cl_strict_ansi_incompatible_with_microsoft:`  
strict ANSI mode is incompatible with Microsoft mode
- 0649 `cl_cfront_incompatible_with_microsoft:`  
cfront mode is incompatible with Microsoft mode
- 0650 `calling_convention_ignored:`  
calling convention specified here is ignored
- 0651 `calling_convention_may_not_precede_nested_declarator:`  
a calling convention may not be followed by a nested declarator
- 0652 `calling_convention_ignored_for_type:`  
calling convention is ignored for this type
- 0654 `decl_modifiers_incompatible_with_previous_decl:`  
declaration modifiers are incompatible with previous declaration
- 0655 `decl_modifiers_invalid_for_this_decl:`  
the modifier "xxxx" is not allowed on this declaration

- 0656 `branch_into_try_block`:  
transfer of control into a try block
- 0657 `incompatible_inline_specifier_on_specific_decl`:  
inline specification is incompatible with previous *"entity"* (declared at line `xxxx`)
- 0658 `template_missing_closing_brace`:  
closing brace of template definition not found
- 0659 `cl_wchar_t_option_only_in_cplusplus`:  
`wchar_t` keyword option can be used only when compiling C++
- 0660 `bad_pack_alignment`:  
invalid packing alignment value
- 0661 `exp_int_constant`:  
expected an integer constant
- 0662 `call_of_pure_virtual`:  
call of pure virtual function
- 0663 `bad_ident_string`:  
invalid source file identifier string
- 0664 `template_friend_definition_not_allowed`:  
a class template cannot be defined in a friend declaration
- 0665 `asm_not_allowed`:  
`"asm"` is not allowed
- 0666 `bad_asm_function_def`:  
`"asm"` must be used with a function definition
- 0667 `nonstd_asm_function`:  
`"asm"` function is nonstandard
- 0668 `nonstd_ellipsis_only_param`:  
ellipsis with no explicit parameters is nonstandard

- 0669 nonstd\_address\_of\_ellipsis:  
" &..." is nonstandard
- 0670 bad\_address\_of\_ellipsis:  
invalid use of " &..."
- 0672 const\_volatile\_ref\_init\_anachronism:  
temporary used for initial value of reference to const volatile  
(anachronism)
- 0673 bad\_const\_volatile\_ref\_init:  
a reference of type "*type*" cannot be initialized with a value of type  
"*type*"
- 0674 const\_volatile\_ref\_init\_from\_rvalue:  
initial value of reference to const volatile must be an lvalue
- 0675 cl\_SVR4\_C\_option\_only\_in\_ansi\_C:  
SVR4 C compatibility option can be used only when compiling ANSI  
C
- 0676 using\_out\_of\_scope\_declaration:  
using out-of-scope declaration of *entity-kind "entity"* (declared at  
line *xxxx*)
- 0677 cl\_strict\_ansi\_incompatible\_with\_SVR4:  
strict ANSI mode is incompatible with SVR4 C mode
- 0678 cannot\_inline\_call:  
call of *entity-kind "entity"* (declared at line *xxxx*) cannot be inlined
- 0679 cannot\_inline:  
*entity-kind "entity"* cannot be inlined
- 0680 cl\_invalid\_pch\_directory:  
invalid PCH directory:
- 0681 exp\_except\_or\_finally:  
expected `__except` or `__finally`

- 0682 `leave_must_be_in_try`:  
a `__leave` statement may only be used within a `__try`
- 0688 `not_found_on_pack_alignment_stack`:  
"xxxx" not found on pack alignment stack
- 0689 `empty_pack_alignment_stack`:  
empty pack alignment stack
- 0690 `cl_rtti_option_only_in_cplusplus`:  
RTTI option can be used only when compiling C++
- 0691 `inaccessible_elided_ctor`:  
*entity-kind "entity"*, required for copy that was eliminated, is inaccessible
- 0692 `uncallable_elided_ctor`:  
*entity-kind "entity"*, required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue
- 0693 `typeid_needs_typeinfo`:  
<typeid> must be included before typeid is used
- 0694 `cannot_cast_away_const`:  
xxxx cannot cast away const or other type qualifiers
- 0695 `bad_dynamic_cast_type`:  
the type in a `dynamic_cast` must be a pointer or reference to a complete class type, or void \*
- 0696 `bad_ptr_dynamic_cast_operand`:  
the operand of a pointer `dynamic_cast` must be a pointer to a complete class type
- 0697 `bad_ref_dynamic_cast_operand`:  
the operand of a reference `dynamic_cast` must be an lvalue of a complete class type
- 0698 `dynamic_cast_operand_must_be_polymorphic`:  
the operand of a runtime `dynamic_cast` must have a polymorphic class type

- 0699 `cl_bool_option_only_in_cplusplus:`  
bool option can be used only when compiling C++
- 0701 `array_type_not_allowed:`  
an array type is not allowed here
- 0702 `exp_assign:`  
expected an "="
- 0703 `exp_declarator_in_condition_decl:`  
expected a declarator in condition declaration
- 0704 `redeclaration_of_condition_decl_name:`  
"xxxx", declared in condition, may not be redeclared in this scope
- 0705 `default_template_arg_not_allowed:`  
default template arguments are not allowed for function templates
- 0706 `exp_comma_or_gt:`  
expected a ",", or ">"
- 0707 `missing_template_param_list:`  
expected a template parameter list
- 0708 `incr_of_bool_deprecated:`  
incrementing a bool value is deprecated
- 0709 `bool_type_not_allowed:`  
bool type is not allowed
- 0710 `base_class_offset_too_large:`  
offset of base class "*entity*" within class "*entity*" is too large
- 0711 `expr_not_bool:`  
expression must have bool type (or be convertible to bool)
- 0712 `cl_array_new_and_delete_option_only_in_cplusplus:`  
array new and delete option can be used only when compiling C++
- 0713 `based_requires_variable_name:`  
*entity-kind* "*entity*" is not a variable name

- 0714 `based_not_allowed_here`:  
    `__based` modifier is not allowed here
- 0715 `based_not_followed_by_star`:  
    `__based` does not precede a pointer operator, `__based` ignored
- 0716 `based_var_must_be_ptr`:  
    variable in `__based` modifier must have pointer type
- 0717 `bad_const_cast_type`:  
    the type in a `const_cast` must be a pointer, reference, or pointer to member to an object type
- 0718 `bad_const_cast`:  
    a `const_cast` can only adjust type qualifiers; it cannot change the underlying type
- 0719 `mutable_not_allowed`:  
    mutable is not allowed
- 0720 `cannot_change_access`:  
    redeclaration of *entity-kind "entity"* is not allowed to alter its access
- 0721 `nonstd_printf_format_string`:  
    nonstandard format string conversion
- 0722 `probable_inadvertent_lbracket_digraph`:  
    use of alternative token "<:" appears to be unintended
- 0723 `probable_inadvertent_sharp_digraph`:  
    use of alternative token "%:" appears to be unintended
- 0724 `namespace_def_not_allowed`:  
    namespace definition is not allowed
- 0725 `missing_namespace_name`:  
    name must be a namespace name
- 0726 `namespace_alias_def_not_allowed`:  
    namespace alias definition is not allowed

- 0727 namespace\_qualified\_name\_required:  
namespace-qualified name is required
- 0728 namespace\_name\_not\_allowed:  
a namespace name is not allowed
- 0729 bad\_combination\_of\_dll\_attributes:  
invalid combination of DLL attributes
- 0730 sym\_not\_a\_class\_template:  
*entity-kind "entity"* is not a class template
- 0731 array\_of\_incomplete\_type:  
array with incomplete element type is nonstandard
- 0732 allocation\_operator\_in\_namespace:  
allocation operator may not be declared in a namespace
- 0733 deallocation\_operator\_in\_namespace:  
deallocation operator may not be declared in a namespace
- 0734 conflicts\_with\_using\_decl:  
*entity-kind "entity"* conflicts with using-declaration of *entity-kind "entity"*
- 0735 using\_decl\_conflicts\_with\_prev\_decl:  
using-declaration of *entity-kind "entity"* conflicts with *entity-kind "entity"* (declared at line xxx)
- 0736 cl\_namespaces\_option\_only\_in\_cplusplus:  
namespaces option can be used only when compiling C++
- 0737 useless\_using\_declaration:  
using-declaration ignored -- it refers to the current namespace
- 0738 class\_qualified\_name\_required:  
a class-qualified name is required
- 0741 using\_declaration\_ignored:  
using-declaration of *entity-kind "entity"* ignored



- 0742 not\_an\_actual\_member:  
*entity-kind "entity"* has no actual member "xxxx"
- 0744 mem\_attr\_incompatible:  
incompatible memory attributes specified
- 0745 mem\_attr\_ignored:  
memory attribute ignored
- 0746 mem\_attr\_may\_not\_precede\_nested\_declarator:  
memory attribute may not be followed by a nested declarator
- 0747 dupl\_mem\_attr:  
memory attribute specified more than once
- 0748 dupl\_calling\_convention:  
calling convention specified more than once
- 0749 type\_qualifier\_not\_allowed:  
a type qualifier is not allowed
- 0750 template\_instance\_already\_used:  
*entity-kind "entity"* (declared at line xxxx) was used before its  
template was declared
- 0751 static\_nonstatic\_with\_same\_param\_types:  
static and nonstatic member functions with same parameter types  
cannot be overloaded
- 0752 no\_prior\_declaration:  
no prior declaration of *entity-kind "entity"*
- 0753 template\_id\_not\_allowed:  
a template-id is not allowed
- 0754 class\_qualified\_name\_not\_allowed:  
a class-qualified name is not allowed
- 0755 bad\_scope\_for\_redeclaration:  
*entity-kind "entity"* may not be redeclared in the current scope

- 0756 `qualifier_in_namespace_member_decl`:  
qualified name is not allowed in namespace member declaration
- 0757 `sym_not_a_type_name`:  
*entity-kind "entity"* is not a type name
- 0758 `explicit_instantiation_not_in_namespace_scope`:  
explicit instantiation is not allowed in the current scope
- 0759 `bad_scope_for_explicit_instantiation`:  
*entity-kind "entity"* cannot be explicitly instantiated in the current scope
- 0760 `multiple_explicit_instantiations`:  
*entity-kind "entity"* explicitly instantiated more than once
- 0761 `typename_not_in_template`:  
typename may only be used within a template
- 0762 `cl_special_subscript_cost_option_only_in_cplusplus`:  
special\_subscript\_cost option can be used only when compiling C++
- 0763 `cl_typename_option_only_in_cplusplus`:  
typename option can be used only when compiling C++
- 0764 `cl_implicit_typename_option_only_in_cplusplus`:  
implicit typename option can be used only when compiling C++
- 0765 `nonstd_character_at_start_of_macro_def`:  
nonstandard character at start of object-like macro definition
- 0766 `exception_spec_override_incompat`:  
exception specification for virtual *entity-kind "entity"* is incompatible with that of overridden *entity-kind "entity"*
- 0767 `pointer_conversion_loses_bits`:  
conversion from pointer to smaller integer

- 0768 generated\_exception\_spec\_override\_incompat:  
exception specification for implicitly declared virtual *entity-kind* "entity" is incompatible with that of overridden *entity-kind* "entity"
- 0769 implicit\_call\_of\_ambiguous\_name:  
"entity", implicitly called from *entity-kind* "entity", is ambiguous
- 0770 cl\_explicit\_option\_only\_in\_cplusplus:  
option "explicit" can be used only when compiling C++
- 0771 explicit\_not\_allowed:  
"explicit" is not allowed
- 0772 conflicts\_with\_predeclared\_type\_info:  
declaration conflicts with "xxxx" (reserved class name)
- 0773 array\_member\_initialization:  
only "()" is allowed as initializer for array *entity-kind* "entity"
- 0774 virtual\_function\_template:  
"virtual" is not allowed in a function template declaration
- 0775 anon\_union\_class\_member\_template:  
invalid anonymous union -- class member template is not allowed
- 0776 template\_depth\_mismatch:  
template nesting depth does not match the previous declaration of *entity-kind* "entity"
- 0777 multiple\_template\_decls\_not\_allowed:  
this declaration cannot have multiple "template <...>" clauses
- 0778 cl\_old\_for\_init\_option\_only\_in\_cplusplus:  
option to control the for-init scope can be used only when compiling C++
- 0779 redeclaration\_of\_for\_init\_decl\_name:  
"xxxx", declared in for-loop initialization, may not be redeclared in this scope

- 0780 `hidden_by_old_for_init`:  
reference is to *entity-kind "entity"* (declared at line `xxxx`) -- under old for-init scoping rules it would have been *entity-kind "entity"* (declared at line `xxxx`)
- 0781 `cl_for_init_diff_warning_option_only_in_cplusplus`:  
option to control warnings on for-init differences can be used only when compiling C++
- 0782 `unnamed_class_virtual_function_def_missing`:  
definition of virtual *entity-kind "entity"* is required here
- 0783 `svr4_token_pasting_comment`:  
empty comment interpreted as token-pasting operator `##`
- 0784 `storage_class_in_friend_decl`:  
a storage class is not allowed in a friend declaration
- 0785 `templ_param_list_not_allowed`:  
template parameter list for *entity* is not allowed in this declaration
- 0786 `bad_member_template_sym`:  
*entity-kind "entity"* is not a valid member class or function template
- 0787 `bad_member_template_decl`:  
not a valid member class or function template declaration
- 0788 `specialization_follows_param_list`:  
a template declaration containing a template parameter list may not be followed by an explicit specialization declaration
- 0789 `specialization_of_referenced_template`:  
explicit specialization of *entity-kind "entity"* must precede the first use of *entity-kind "entity"*
- 0790 `explicit_specialization_not_in_namespace_scope`:  
explicit specialization is not allowed in the current scope
- 0791 `partial_specialization_not_allowed`:  
partial specialization of *entity-kind "entity"* is not allowed

- 0792 `entity_cannot_be_specialized`:  
*entity-kind "entity"* is not an entity that can be explicitly specialized
- 0793 `specialization_of_referenced_entity`:  
explicit specialization of *entity-kind "entity"* must precede its first use
- 0794 `template_param_in_elab_type`:  
template parameter `xxxx` may not be used in an elaborated type specifier
- 0795 `old_specialization_not_allowed`:  
specializing *entity-kind "entity"* requires "template<>" syntax
- 0798 `cl_old_specializations_option_only_in_cplusplus`:  
option "old\_specializations" can be used only when compiling C++
- 0799 `nonstd_old_specialization`:  
specializing *entity-kind "entity"* without "template<>" syntax is nonstandard
- 0800 `bad_linkage_for_decl`:  
this declaration may not have extern "C" linkage
- 0801 `not_a_template_name`:  
"xxxx" is not a class or function template name in the current scope
- 0802 `nonstd_default_arg_on_function_template_redecl`:  
specifying a default argument when redeclaring an unreferenced function template is nonstandard
- 0803 `default_arg_on_function_template_not_allowed`:  
specifying a default argument when redeclaring an already referenced function template is not allowed
- 0804 `pm_derived_class_from_virtual_base`:  
cannot convert pointer to member of base class *"type"* to pointer to member of derived class *"type"* -- base class is virtual

- 0805 `bad_exception_specification_for_specialization`:  
exception specification is incompatible with that of *entity-kind* "*entity*" (declared at line *xxxx*):
- 0806 `omitted_exception_specification_on_specialization`:  
omission of exception specification is incompatible with *entity-kind* "*entity*" (declared at line *xxxx*)
- 0807 `unexpected_end_of_default_arg`:  
the parse of this expression has changed between the point at which it appeared in the program and the point at which the expression was evaluated -- "typename" may be required to resolve the ambiguity
- 0808 `default_init_of_reference`:  
default-initialization of reference is not allowed
- 0809 `uninitialized_field_with_const_member`:  
uninitialized *entity-kind* "*entity*" has a const member
- 0810 `uninitialized_base_class_with_const_member`:  
uninitialized base class "*type*" has a const member
- 0811 `missing_default_constructor_on_const`:  
const *entity-kind* "*entity*" requires an initializer -- class "*type*" has no explicitly declared default constructor
- 0812 `missing_default_constructor_on_unnamed_const`:  
const object requires an initializer -- class "*type*" has no explicitly declared default constructor
- 0813 `cl_impl_extern_c_conv_option_only_in_cplusplus`:  
option "implicit\_extern\_c\_type\_conversion" can be used only when compiling C++
- 0814 `cl_strict_ansi_incompatible_with_long_preserving`:  
strict ANSI mode is incompatible with long preserving rules
- 0815 `useless_type_qualifier_on_return_type`:  
type qualifier on return type is meaningless

- 0816 `type_qualifier_on_void_return_type`:  
in a function definition a type qualifier on a "void" return type is not allowed
- 0817 `static_data_member_not_allowed`:  
static data member declaration is not allowed in this class
- 0818 `invalid_declaration`:  
template instantiation resulted in an invalid function declaration
- 0819 `ellipsis_not_allowed`:  
"..." is not allowed
- 0820 `clExternInlineOptionOnlyInCplusplus`:  
option "extern\_inline" can be used only when compiling C++
- 0821 `extern_inline_never_defined`:  
extern inline *entity-kind* "entity" was referenced but not defined
- 0822 `invalid_destructor_name`:  
invalid destructor name for type "*type*"
- 0823 `nonstandard_destructor_reference`:  
use of *entity-kind* "entity" in a destructor call is nonstandard
- 0824 `ambiguous_destructor`:  
destructor reference is ambiguous -- both *entity-kind* "entity" and *entity-kind* "entity" could be used
- 0825 `virtual_inline_never_defined`:  
virtual inline *entity-kind* "entity" was never defined
- 0826 `unreferenced_function_param`:  
*entity-kind* "entity" was never referenced
- 0827 `union_already_initialized`:  
only one member of a union may be specified in a constructor initializer list
- 0828 `no_array_new_and_delete_support`:  
support for "new[]" and "delete[]" is disabled

- 0829 `double_for_long_double`:  
"double" used for "long double" in generated C code
- 0830 `no_corresponding_delete`:  
*entity-kind "entity"* has no corresponding operator `delete` (to be called if an exception is thrown during initialization of an allocated object)
- 0831 `useless_placement_delete`:  
support for placement delete is disabled
- 0832 `no_appropriate_delete`:  
no appropriate operator delete is visible
- 0833 `ptr_or_ref_to_incomplete_type`:  
pointer or reference to incomplete type is not allowed
- 0834 `bad_partial_specialization`:  
invalid partial specialization -- *entity-kind "entity"* is already fully specialized
- 0835 `incompatible_exception_specs`:  
incompatible exception specifications
- 0836 `returning_ref_to_local_variable`:  
returning reference to local variable
- 0837 `nonstd_implicit_int`:  
omission of explicit type is nonstandard ("int" assumed)
- 0838 `ambiguous_partial_spec`:  
more than one partial specialization matches the template argument list of *entity-kind "entity"*
- 0840 `partial_spec_is_primary_template`:  
a template argument list is not allowed in a declaration of a primary template
- 0841 `default_not_allowed_on_partial_spec`:  
partial specializations may not have default template arguments



- 0842 `not_used_in_partial_spec_arg_list`:  
*entity-kind "entity"* is not used in template argument list of  
*entity-kind "entity"*
- 0843 `partial_spec_param_depends_on_tmpl_param`:  
the type of partial specialization template parameter *entity-kind*  
*"entity"* depends on another template parameter
- 0844 `partial_spec_arg_depends_on_tmpl_param`:  
the template argument list of the partial specialization includes a  
nontype argument whose type depends on a template parameter
- 0845 `partial_spec_after_instantiation`:  
this partial specialization would have been used to instantiate  
*entity-kind "entity"*
- 0846 `partial_spec_after_instantiation_ambiguous`:  
this partial specialization would have been made the instantiation of  
*entity-kind "entity"* ambiguous
- 0847 `expr_not_integral_or_enum`:  
expression must have integral or enum type
- 0848 `expr_not_arithmetic_or_enum`:  
expression must have arithmetic or enum type
- 0849 `expr_not_arithmetic_or_enum_or_pointer`:  
expression must have arithmetic, enum, or pointer type
- 0850 `cast_not_integral_or_enum`:  
type of cast must be integral or enum
- 0851 `cast_not_arithmetic_or_enum_or_pointer`:  
type of cast must be arithmetic, enum, or pointer
- 0852 `expr_not_object_pointer`:  
expression must be a pointer to a complete object type
- 0853 `member_partial_spec_not_in_class`:  
a partial specialization of a member class template must be declared  
in the class of which it is a member

- 0854 `partial_spec_nontype_expr`:  
a partial specialization nontype argument must be the name of a nontype parameter or a constant
- 0855 `different_return_type_on_virtual_function_override`:  
return type is not identical to return type "*type*" of overridden virtual function *entity-kind* "*entity*"
- 0856 `cl_guiding_decls_option_only_in_cplusplus`:  
option "`guiding_decls`" can be used only when compiling C++
- 0857 `member_partial_spec_not_in_namespace`:  
a partial specialization of a class template must be declared in the namespace of which it is a member
- 0858 `pure_virtual_function`:  
*entity-kind* "*entity*" is a pure virtual function
- 0859 `no_overrider_for_pure_virtual_function`:  
pure virtual *entity-kind* "*entity*" has no overrider
- 0860 `decl_modifiers_ignored`:  
`__declspec` attributes ignored
- 0861 `invalid_char`:  
invalid character in input line
- 0862 `incomplete_return_type`:  
function returns incomplete type "*type*"
- 0863 `local_pragma_pack`:  
effect of this "`#pragma pack`" directive is local to *entity-kind* "*entity*"
- 0864 `not_a_template`:  
`xxx` is not a template
- 0865 `friend_partial_specialization`:  
a friend declaration may not declare a partial specialization
- 0866 `exception_specification_ignored`:  
exception specification ignored

- 0867 unexpected\_type\_for\_size\_t:  
declaration of "size\_t" does not match the expected type "*type*"
- 0868 exp\_gt\_not\_shift\_right:  
space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator)
- 0869 bad\_multibyte\_char\_locale:  
could not set locale "xxxx" to allow processing of multibyte characters
- 0870 bad\_multibyte\_char:  
invalid multibyte character sequence
- 0871 bad\_type\_from\_instantiation:  
template instantiation resulted in unexpected function type of "*type*"  
(the meaning of a name may have changed since the template declaration -- the type of the template is "*type*")
- 0872 ambiguous\_guiding\_decl:  
ambiguous guiding declaration -- more than one function template "*entity*" matches type "*type*"
- 0873 non\_integral\_operation\_in\_tmpl\_arg:  
non-integral operation not allowed in nontype template argument
- 0874 cl\_embedded\_cplusplus\_option\_only\_in\_cplusplus:  
option "embedded\_c++" can be used only when compiling C++
- 0875 templates\_in\_embedded\_cplusplus:  
Embedded C++ does not support templates
- 0876 exceptions\_in\_embedded\_cplusplus:  
Embedded C++ does not support exception handling
- 0877 namespaces\_in\_embedded\_cplusplus:  
Embedded C++ does not support namespaces
- 0878 rtti\_in\_embedded\_cplusplus:  
Embedded C++ does not support run time type information

- 0879 `new_cast_in_embedded_cplusplus:`  
Embedded C++ does not support the new cast syntax
- 0880 `using_decl_in_embedded_cplusplus:`  
Embedded C++ does not support using declarations
- 0881 `mutable_in_embedded_cplusplus:`  
Embedded C++ does not support “mutable”
- 0882 `multiple_inheritance_in_embedded_cplusplus:`  
Embedded C++ does not support multiple or virtual inheritance
- 0883 `cl_invalid_microsoft_version:`  
invalid Microsoft version number
- 0884 `inheritance_kind_already_set:`  
pointer-to-member representation has already been set for *entity-kind* “*entity*”
- 0885 `bad_constructor_type:`  
“*type*” cannot be used to designate constructor for “*type*”
- 0886 `bad_suffix:`  
invalid suffix on integral constant
- 0887 `uuidof_requires_uuid_class_type:`  
operand of `__uuidof` must have a class type for which `__declspec(uuid("..."))` has been specified
- 0888 `bad_uuid_string:`  
invalid GUID string in `__declspec(uuid("..."))`
- 0889 `cl_vla_option_only_in_C:`  
option “vla” can be used only when compiling C
- 0890 `vla_with_unspecified_bound_not_allowed:`  
variable length array with unspecified bound is not allowed
- 0891 `explicit_template_args_not_allowed:`  
an explicit template argument list is not allowed on this declaration

- 0892 `variably_modified_type_not_allowed`:  
an entity with linkage cannot have a variably modified type
- 0893 `vla_is_not_auto`  
a variable length array cannot have static storage duration
- 0894 `sym_not_a_template`:  
*entity-kind "entity"* is not a template
- 0896 `expected_template_arg`:  
expected a template argument
- 0897 `explicit_template_args_in_expr`:  
explicit function template argument lists are not supported yet in expression contexts
- 0898 `no_params_with_class_or_enum_type`:  
nonmember operator requires a parameter with class or enum type
- 0899 `cl_enum_overloading_option_only_in_cplusplus`:  
option "enum\_overloading" can be used only when compiling C++
- 0900 `using_declaration_not_allowed`:  
using-declaration of *entity-kind "entity"* is not allowed
- 0901 `destructor_qualifier_type_mismatch`:  
qualifier of destructor name "*type*" does not match type "*type*"
- 0902 `type_qualifier_ignored`:  
type qualifier ignored
- 0903 `cl_nonstandard_qualifier_deduction_option_only_in_cplusplus`:  
option "nonstd\_qualifier\_deduction" can be used only when compiling C++
- 0904 `cannot_define_dllimport_function`:  
a function declared "dllimport" may not be defined
- 0905 `bad_declspec_property`:  
incorrect property specification; correct form is  
`__declspec(property(get=name1,put=name2))`

- 0906 `dupl_get_or_put`:  
property has already been specified
- 0907 `declspec_property_not_allowed`:  
`__declspec(property)` is not allowed on this declaration
- 0908 `no_get_property`:  
member is declared with `__declspec(property)`, but no "get"  
function was specified
- 0909 `get_property_function_missing`:  
the `__declspec(property)` "get" function "xxxx" is missing
- 0910 `no_put_property`:  
member is declared with `__declspec(property)`, but no "put"  
function was specified
- 0911 `put_property_function_missing`:  
the `__declspec(property)` "put" function "xxxx" is missing
- 0912 `dual_lookup_ambiguous_name`:  
ambiguous class member reference -- *entity-kind "entity"* (declared  
at line xxx) used in preference to *entity-kind "entity"* (declared at  
line xxx)
- 0913 `bad_allocate_segname`:  
missing or invalid segment name in `__declspec(allocate("..."))`
- 0914 `declspec_allocate_not_allowed`:  
`__declspec(allocate)` is not allowed on this declaration
- 0915 `dupl_allocate_segname`:  
a segment name has already been specified
- 0916 `pm_virtual_base_from_derived_class`:  
cannot convert pointer to member of derived class "*type*" to pointer  
to member of base class "*type*" -- base class is virtual
- 0917 `cl_invalid_instantiation_directory`:  
invalid directory for instantiation files:

- 0918 `cl_one_instantiation_per_object_option_only_in_cplusplus`:  
option "one\_instantiation\_per\_object" can be used only when  
compiling C++
- 0919 `invalid_output_file`:  
invalid output file: "xxxx"
- 0920 `cannot_open_output_file`:  
cannot open output file: "xxxx"
- 0921 `cl_ii_file_name_incompatible_with_multiple_inputs`:  
an instantiation information file name may not be specified when  
compiling several input files
- 0922 `cl_one_instantiation_per_object_incompatible_with_multiple_inputs`:  
option "one\_instantiation\_per\_object" may not be used when  
compiling several input files
- 0923 `cl_ambiguous_option`:  
more than one command line option matches the abbreviation  
"--xxxx":
- 0925 `cv_qualified_function_type`:  
a type qualifier cannot be applied to a function type
- 0926 `cannot_open_definition_list_file`:  
cannot open definition list file: "xxxx"
- 0927 `cl_late_tiebreaker_option_only_in_cplusplus`:  
late/early tiebreaker option can be used only when compiling C++
- 0928 `bad_va_start`:  
incorrect use of `va_start`
- 0929 `bad_va_arg`:  
incorrect use of `va_arg`
- 0930 `bad_va_end`:  
incorrect use of `va_end`

- 0931 `cl_pending_instantiations_option_only_in_cplusplus`:  
pending instantiations option can be used only when compiling C++
- 0932 `cl_invalid_import_directory`:  
invalid directory for `#import` files:
- 0933 `cl_import_only_in_microsoft`:  
an import directory can be specified only in Microsoft mode
- 0934 `ref_not_allowed_in_union`:  
a member with reference type is not allowed in a union
- 0935 `typedef_not_allowed`:  
"typedef" may not be specified here
- 0936 `redecl_changes_access`:  
redeclaration of *entity-kind* "*entity*" alters its access
- 0937 `qualified_name_required`:  
a class or namespace qualified name is required
- 0938 `opfile_no_file_name`:  
no filename specified for `-opfile` option
- 0939 `opfile_cannot_open_file`:  
cannot open command file *name*
- 0940 `dir_list_not_supported`:  
`#list` directive is ignored in C++ mode





ERRORS

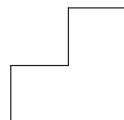
# INDEX

## INDEX

---



**TASKING**



---

# INDEX

---

# Symbols

#define, 3-22  
 #pragma, 3-81  
 #undef, 3-73  
 \_\_ARRAY\_OPERATORS, 3-73  
 \_\_cplusplus, 3-73  
 \_\_DATE\_\_, 3-73  
 \_\_FILE\_\_, 3-73  
 \_\_LINE\_\_, 3-73  
 \_\_SIGNED\_CHARS\_\_, 3-73  
 \_\_STDC\_\_, 3-73  
 \_\_TIME\_\_, 3-73  
 \_\_BOOL, 3-73  
 \_\_WCHAR\_T, 3-73

## A

alternative tokens, 3-13  
 anachronism, 2-7  
 anachronisms, 3-14, 3-19, 3-52  
 ansi standard, 3-73  
 array new and delete, 3-15  
 automatic instantiation, 1-5  
 automatic instantiation method, 2-25

## B

bool keyword, 3-17

## C

C++, language extensions, 2-3  
 C++ dialect, 2-3  
   *accepted*, 2-4  
   *anachronisms accepted*, 2-7  
   *cfront 2.1 and 3.0 extensions*, 2-13  
   *cfront 2.1 extensions*, 2-9  
   *new language features accepted*, 2-4

*new language features not accepted*, 2-6  
   *normal C++ mode extensions*, 2-8  
   *not accepted*, 2-6  
 C++ language features  
   *accepted*, 2-4  
   *not accepted*, 2-6  
 c\_plusplus, 3-73  
 can\_instantiate, 3-81  
 catastrophic error, 4-3  
 cfront, 3-19  
   *2.1 and 3.0 extensions*, 2-13  
   *2.1 extensions*, 2-9  
 command file, 3-56  
 compiler diagnostics, 4-1  
 compiler use, 3-1  
 cross-reference, 3-79

## D

detailed option description, compiler, 3-12-3-79  
 development flow, 1-3  
 diagnostics, 4-1  
   *brief*, 3-18  
   *error severity*, 3-24, 4-3  
   *wrap*, 3-78  
 digraph, 3-13  
 do\_not\_instantiate, 3-81

## E

embedded C++, 3-27  
 entities, remove unneeded, 3-66  
 enum overloading, 3-29  
 error, 4-3  
   error level, 4-5  
   error limit, 3-30  
   error messages, A-1  
   error number, 3-25

error output file, 3-31  
 error severity, 3-24, 4-3  
 exception, 3-32  
 exit status, 4-5  
 explicit specifier, 3-33  
 extensions to C++, 2-3  
 extern C, 3-39  
 extern C++, 3-39  
 extern inline, 3-34

## F

file extensions, 3-3  
 for-init statement, 3-35, 3-49

## G

guiding declarations, 3-38

## H

hdrstop, 3-81  
 header stop, 2-31, 2-36

## I

implicit inclusion, 2-30  
 inline function, 3-34  
 inlining, 3-42  
 instantiate, 3-81  
 instantiation, 2-22  
   *automatic*, 2-25  
 instantiation information file, 1-5  
 instantiation mode, 2-27  
   *all*, 2-27  
   *local*, 2-28  
   *none*, 2-27  
   *used*, 2-27

instantiation pragmas, 2-28  
 introduction, 1-3  
 invocation, 3-3

## K

keyword  
   *bool*, 3-17  
   *typename*, 3-72  
   *wchar\_t*, 3-77

## L

language extensions, 3-69  
 language implementation, 2-1  
 lifetime, 3-47  
 list file, 3-45

## M

messages  
   *diagnostic*, 4-3  
   *termination*, 4-4

## N

namespace, 2-20, 3-48  
   *std*, 3-76  
 no\_pch, 2-36, 3-81

## O

once, 3-81  
 operator, keywords, 3-13  
 options  
   *--alternative\_tokens*, 3-13

--anachronisms, 3-14  
--array\_new\_and\_delete, 3-15  
--auto\_instantiation, 3-16  
--bool, 3-17  
--brief\_diagnostics, 3-18  
--cfront\_2.1, 3-19  
--cfront\_3.0, 3-19  
--comments, 3-20  
--create\_pch, 3-21  
--define\_macro, 3-22  
--dependencies, 3-23  
--diag\_error, 3-24  
--diag\_remark, 3-24  
--diag\_suppress, 3-24  
--diag\_warning, 3-24  
--display\_error\_number, 3-25  
--distinct\_template\_signatures, 3-26  
--embedded\_c++, 3-27  
--enum, 3-28  
--enum\_overloading, 3-29  
--error\_limit, 3-30  
--error\_output, 3-31  
--exceptions, 3-32  
--explicit, 3-33  
--extern\_inline, 3-34  
--for\_init\_diff\_warning, 3-35  
--force\_vtbl, 3-36  
--gen\_c\_file\_name, 3-37  
--guiding\_decls, 3-38  
--implicit\_extern\_c\_type\_conversion, 3-39  
--implicit\_include, 3-40  
--implicit\_typename, 3-41  
--inlining, 3-42  
--instantiate, 3-43  
--list, 3-45  
--long\_lifetime\_temps, 3-47  
--namespaces, 3-48  
--new\_for\_init, 3-49  
--no\_alternative\_tokens, 3-13  
--no\_anachronisms, 3-14  
--no\_array\_new\_and\_delete, 3-15  
--no\_auto\_instantiation, 3-16  
--no\_bool, 3-17  
--no\_brief\_diagnostics, 3-18  
--no\_code\_gen, 3-50  
--no\_distinct\_template\_signatures, 3-26  
--no\_enum\_overloading, 3-29  
--no\_exceptions, 3-32  
--no\_explicit, 3-33  
--no\_extern\_inline, 3-34  
--no\_for\_init\_diff\_warning, 3-35  
--no\_guiding\_decls, 3-38  
--no\_implicit\_extern\_c\_type\_conversion, 3-39  
--no\_implicit\_include, 3-40  
--no\_implicit\_typename, 3-41  
--no\_inlining, 3-42  
--no\_line\_commands, 3-51  
--no\_namespaces, 3-48  
--no\_nonconst\_ref\_anachronism, 3-52  
--no\_old\_specializations, 3-59  
--no\_preproc\_only, 3-53  
--no\_remove\_unneeded\_entities, 3-66  
--no\_rtti, 3-67  
--no\_special\_subscript\_cost, 3-68  
--no\_typename, 3-72  
--no\_use\_before\_set\_warnings, 3-54  
--no\_using\_std, 3-76  
--no\_warnings, 3-55  
--no\_wchar\_t\_keyword, 3-77  
--no\_wrap\_diagnostics, 3-78  
--nonconst\_ref\_anachronism, 3-52  
--old\_for\_init, 3-49  
--old\_line\_commands, 3-58  
--old\_specializations, 3-59  
--old\_style\_preprocessing, 3-60  
--output, 3-57  
--pch, 3-61  
--pch\_dir, 3-62  
--pch\_messages, 3-63  
--preprocess, 3-64  
--remarks, 3-65

*--remove\_unneeded\_entities*, 3-66  
*--rtti*, 3-67  
*--short\_lifetime\_temps*, 3-47  
*--special\_subscript\_cost*, 3-68  
*--strict*, 3-69  
*--strict\_warnings*, 3-69  
*--suppress\_vtbl*, 3-70  
*--trace\_includes*, 3-71  
*--typename*, 3-72  
*--undefine\_macro*, 3-73  
*--use\_pch*, 3-75  
*--using\_std*, 3-76  
*--wchar\_t\_keyword*, 3-77  
*--wrap\_diagnostics*, 3-78  
*--xref*, 3-79  
*-B*, 3-40  
*-D*, 3-22  
*-E*, 3-64  
*-H*, 3-71  
*-M*, 3-23  
*-opfile*, 3-56  
*-U*, 3-73  
*-w*, 3-55  
*-X*, 3-79  
*-x*, 3-32  
*detailed description*, 3-12  
*overview*, 3-3  
*overview in functional order*, 3-8  
*priority*, 3-3  
 options file, 3-56  
 output file, 3-37, 3-57  
 overview, 1-1

## P

pch mode  
   *automatic*, 2-31, 3-61  
   *manual*, 2-35, 3-21, 3-75  
 pragma  
   *can\_instantiate*, 2-28, 3-81  
   *do\_not\_instantiate*, 2-28, 3-81  
   *hdrstop*, 2-31, 2-36, 3-81

*instantiate*, 2-28, 3-81  
*no\_pch*, 2-36, 3-81  
*once*, 3-81  
*separate*, 3-82  
 pragmas, 3-81  
 precompiled header, 2-31  
   *automatic*, 2-31, 3-61  
   *create*, 2-35, 3-21  
   *directory*, 2-35, 2-36, 3-62  
   *manual*, 2-35  
   *messages*, 3-63  
   *performance*, 2-36  
   *pragmas*, 2-36  
   *prefix*, 2-34  
   *use*, 2-35, 3-75  
 predefined symbols, 3-73  
 prelinker, 1-5  
 prelinker prelk56, 2-25

## R

raw listing, 3-45  
 remark, 4-3  
 remarks, 3-65  
 return values, 4-5  
 run-time type information, 3-67

## S

*separate*, 3-82  
 signals, 4-5  
 stack, 2-22  
 symbols, predefined, 3-73  
 syntax checking, 3-50

## T

template, 2-22  
   *distinct signatures*, 3-26

*guiding declarations*, 3-38  
*specialization*, 3-59  
template instantiation, 2-22  
  *#pragma directives*, 2-28  
  *automatic*, 2-23, 3-16  
  *implicit inclusion*, 2-30, 3-40  
  *instantiation modes*, 2-27, 3-43  
tool chain, 1-3  
  *prelinker*, 1-5  
typename keyword, 3-72

## V

virtual function table, 3-36, 3-70

## W

warning, 4-3  
warnings (suppress), 3-54, 3-55  
wchar\_t keyword, 3-77



