Contents

#pragma

CLint++   supports preprocessor directives and special comments to control the detailed reporting of errors and warnings in source text.

You can disable a specific warning for a single statement by bracketing the statement with **#pragma** directives like this:

```
            #pragma lint -w-pua
            if (j = 0)
            #pragma lint .wpua
```

or

```
            /*lint -w-pua*/
            if (j = 0)
            /*lint .wpua*/
```

which will suppress the warning otherwise generated for ***possible unintended assignment***. Care must be taken to not insert these directives in the middle of a statement - they may not take effect when intended. The following **#pragma** directives and comments are supported:

#pragma argsused
/*argsused*/
/*ARGSUSED*/

If you precede a function which does not use all its declared arguments with this directive, complaints about unused arguments will be suppressed for that function.

#pragma notreached
/*notreached*/
/*NOTREACHED*/

If you precede an intentionally unreachable statement with this directive, complaints about unreachable code will be suppressed.

#pragma lint -wxxx
/*lint -wxxx*/

You can enable any of the warning options with this directive. The previous value of the setting is saved to possibly be restored using **lint .wxxx**.

#pragma lint -w-xxx
/*lint -w-xxx*/

You can disable any of the warning options with this directive. The previous value of the setting is saved to possibly be restored using **lint .wxxx**.

#pragma lint .wxxx
/*lint .wxxx*/

This restores the warning option in effect prior to the most recent use of **lint -wxxx** or **lint -w-xxx**. If used before either of these, it has no effect.

#pragma message *text*
This generate a message exactly as #message does.

Banner dialog

For the Shareware version of CLint++, you are presented with this dialog. It appears on every use of the Shareware version to remind you of the terms of the Shareware license. Those terms are:

LICENSE AGREEMENT

CLINT++ (c) 1991-1997 R&D ASSOCIATES - ALL RIGHTS RESERVED

This is a Shareware product. It is NOT free! You are entitled to use the software for a trial period of 30 days, after which for continued use, you must pay a registration fee. You will receive the latest version, a printed manual, and a year's technical support. We encourage you to distribute this software amongst your friends, but the same conditions will apply. At most two copies may be concurrently used on a network to evaluate the network features.

Use of this product constitutes your acceptance of these terms and conditions and your agreement to abide by them.

DISCLAIMER

R&D ASSOCIATES PROVIDE THIS SOFTWARE "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

In the interests of continual product improvement we reserve the right to make changes to the documentation or software at any time without any obligation to notify any person of these changes.

SITE LICENSES

R&D Associates will grant by negotiation site licenses to use the Software for commercial purposes by a number of users. Reduced license fees are available dependent on the number of users.

Ordering and Support

We distribute with CLint++ the file ORDER.DOC. This is a plain ASCII text file you can print directly to your printer, or view and/or edit with CLint++ itself. Orders must be placed in writing with an enclosed check or money order to:

In the USA or Canada:

> R&D Associates, 356 D. W. Highway Suite 232,
> Merrimack, NH 03054, USA

In the UK or Europe:

> R&D Associates, 16 High Street, Rainham,
> Kent, ME8 7JE, ENGLAND

Corporate customers may, by prior arrangement, place orders against corporate Purchase Orders which will be invoiced with the delivery.

CLint++ costs $40 ($25 if upgrading) in the USA or Canada, or £25 (£16 if upgrading) in the UK or Europe. Customers outside Europe or the USA must add $5 if ordering from our USA office, or £8 (£5 with EuroCheque) if ordering from our UK office to cover postage, packing, and bank charges. Upgrade orders must be accompanied by the serial number(s) (use Help | About) of the copy(s) being upgraded.

R&D Associates can be contacted on CompuServe by mailing user ID:

> 100013,1042 (USA)

or

> 100111,1162 (Europe)

What's New

This version of CLint++ has several new features:

32 bit CLint

New with 4.38/5.38 is the ability for Windows 95/NT customers to order CLint as a completely 32 bit program. This provides full support for long file names, and a considerable increase in speed. It is also required to support Microsoft Visual C++ 4.2 or higher. Win32s for Windows 3.1/3.11 is **not** supported.

New compiler support

Added support for Borland C++ 32bit compiler modes for all compilers from Borland C++ 4.0 through Borland C++ 5.1.

Added support for Microsoft Visual C++ 32bit compilers modes for all compilers from Visual C++ 1.0 through Visual C++ 4.1. From 4.38/5.38 (32 bit version only) support for Visual C++ 4.2 or higher.

Added support for Symantec C++ 7.x compilers, and their **long long** data type.

Added support for the Keil 8051 C compiler. This correctly understands all of their non-ANSI extensions to C.

Added support for the Microtec 68K C++ compiler, and their **packed** keyword.

Printing

You have the choice of using color or hatched barcharts when printing. Previously, colors were printed regardless of the printer, resulting in hard to read charts.

From a Query or Intray window, **Print Details** now offers to print one report per page (as before), or to pack as many reports as possible on each page. The report layout has been changed to print in fewer lines.

Previous faults on certain printers (PostScript under Windows 3.1) have been rectified.

Editing

The greatly improved editor allowing direct editing of arbitrary binary files. You can for example, edit a string in an executable file, save it, and then execute the changed version (but if you do, use **overtype** mode to avoid changing the length).

Support for Unix style files. Editing and adding lines in files with LineFeed only line ends occurs correctly - previously CLint++ changed all line ends to Carriage Return -LineFeed.

Support for files containing lines of unlimited length. Previously, CLint++ would GP fault on lines exceeding 2048 characters.

This comes with a Hex edit/view mode, and the ability to show line numbers, and/or original line numbers in any editor window, and Binary mode which shows all characters including line ends.

File | Compare now remembers the directory of your most recent use of Browse, which can make getting other files from that directory much faster.

File | Open when CLint++ has no open files now remembers the directory of your most recent use of File | Open, which can make opening other files from that directory much faster.

Context help

All context menus which pop-up when right-clicking in any window have a Help entry providing immediate context help for that window.

Function searching

Added ultra-fast minimalist parsing of files using Search | Find Functions for the purpose of finding all function definitions in your code for all files in a Lint File (CLN file). Previously, only open editor windows could be searched, and the search results were not persistent. Certain types of function definition (functions returning pointers to functions) are not detected however.

This creates a file with the same name as the Lint File, but with a TAG extension containing the definitions. Automatic dependency analysis occurs on each use of Find Functions, and only those files which have been modified since the list use are re-parsed.

Defect tracking

The defect tracking feature now supports the operators "<", "<=", ">", and ">=" when constructing queries. You can now for example, search for all defects in a range of dates with ease.

Parsing

Added automatic library (CLB) file construction. When you choose a compiler mode or variant for which CLint++ needs a new CLB file, it offers to construct one on the fly. Be aware that this construction may (almost certainly will) fail if you lie and point to header files for a compiler other than the one CLint++ is expecting. This feature allows CLint++ to ship with no CLB files except CGEN.CLB for the generic ANSI C compiler.

Added recent ANSI/ISO C++ extensions **namespace** and **using**, together with support for the new **#include** semantics. These extensions fully support the Rogue Wave implementation of the Standard Template Library which ships with Borland C++ 5.1 in both 16 and 32 bit modes.

Added the ability to selectively enable or disable portions of the ANSI/ISO C++ features like **bool** or the new scoping rules for **for**, **if**, **while** and so on locally declared variables, as might be required to accommodate differing compilers.

Added the ability to warn of the use of **static**, which is now deprecated by the ANSI/ISO standard.

Added a compiler vendor database which allows rapid addition and support of new compiler vendors. Since the supported compiler list is no longer in the executable file, it is easy to ship a small number of short files to add support for a new vendor.

System Administration

CLint++ provides means for coordinating the work of groups. You can:

Mail other users, or all users on a project,

Establish report databases (for requirements, defects etc.),

Query, sort, create, and print database reports.

Before you can do this, you must as System Administrator create the Network Directory, create the user Work Groups (equivalent to job titles), create the User accounts, and create one or more Project databases.

Once setup, CLint++ allows members of your group or department to mail each other, track defects, requirements, hazards (or any other data relevant to your development effort), obtain database reports, create new reports, and of course check C/C++ code and manage version control.

CLint++ uses a network independent method of managing databases and shared access to them. It will therefore work on any network which supports mounting remote network drives on a local PC. It is not required that all users see the directory using the same logical path: any convenient mapping will work.

CLint++ supports passwords for network access. These are not related to any network passwords which may be in use, and are stored encrypted in the user database, but for ease of use we advise that you use existing passwords. Because many actions are possibly hazardous to your data, you should use passwords for at least the Administrator and Manager accounts, and ideally everyone to reduce the possibility of fraud. However, once a non-empty password is entered, CLint++ allows users to suppress further queries for a password on subsequent network accesses. This feature should be used with caution if you leave CLint++ running when not at your desk.

A key property of any user is that they are a member of a Work Group. These groups are specified by the administrator, granting overall permissions to create or modify to members of the group. Individuals can have lesser permissions set by the administrator, but never greater than their Work Group allows. When CLint++ first builds the user database, it creates the groups Administrator, Manager, Engineer, and Tester. Administrators have complete permission to perform any action. By default, Managers can do anything except configure database fields, Engineers can originate reports, and modify reports assigned to them, and Testers can originate reports. The administrator can and should customize these titles and properties for a given site.

The suggested work flow for handling (for example) defect reporting is:

Someone originates a report describing a defect. They are the **Originator**. They may add supporting attachments giving evidence of the defect, and enter data into any custom fields defined for the defect database. The report starts with the disposition **Open**. A notification of this creation is mailed to a Manager, and any other uses who choose to be notified of creation.

The Manager on receiving notification examines the report in the Intray Window. She allocates the report to an Engineer for action, possibly adding supplementary comments - this is the act of **Assigning** a report: the Manager is the **Assignor**, the recipient is the **Assignee**. A notification of this assignment is sent to the Engineer, and any other users who choose to be notified of assignment.

The Engineer on receiving notification examines the report in the Intray Window, and performs work to rectify the defect. On completion, he attaches any reports describing the work done, and re-assigns the report back to the Manager (or possibly to a QA group), which notifies them that the re-work is complete.

When the work is seen to be complete, the report disposition is changed to **Closed** by the Manager. The **Originator** is notified of this closure, and any other users who choose to be notified of this.

This work flow works equally well for handling requirements and hazards.

The duties of the Administrator are to create and maintain databases, and possibly also to create and maintain user properties.

The CLint++ distribution must be loaded individually on the hard disks of each user - it must ***not*** be accessed by users from a shared network drive. Each copy of CLint++ is unique and serialized. CLint++ enforces that no more than one instance of a serial number is accessing network drives at any time. Should CLint++ detect two or more different instances of itself with the same serial number, all violating instances are terminated giving each user the opportunity to save their files. These requirements are relaxed for the Shareware distribution: at most two identical instances may be accessing shared network drives to allow evaluation.

CLint++'s database facility is not intended to be secure. The permissions scheme is designed to prevent casual or unintentional modification to records which are not authorized. It makes no attempt to hide records from any user (the purpose of the database is to share data between users), and user passwords are visible to and may be modified by the System Administrator. However, when passwords are properly used, the probability of unintended corruption of records is minimal, as is the possibility of System Administrator permissions being obtained by unauthorized users.

As with all database applications, you should back up the databases regularly in case of server, disk, or program failure.

Creating a Network Directory

Before you can use CLint++ for mail or report database management, you must create a directory on a shared network drive visible to all users. This creation is only possible when CLint++ is run with the ***/a*** command switch, which you can do by highlighting CLINTW.EXE in **File Manager**, then choosing File | Run, and appending ***/a*** to the command line. CLint++ must *not* already be running for this to be effective.

The Administrator should not be the same user name or ID as any real user of the system. When you perform the following actions, use slightly modified versions of your name and any ID you think is appropriate.

When CLint++ starts, choose Options **|** Mail Setup, and enter the full path of the network mail directory you require in the **System Admin Directory** box. Enter your full name in the **Full Name** box, enter an ID in the **ID** box (max. 8 characters), and enter an initial password in the **Password** box.

Note: As you are about to create a new Administrator account in your (modified) name, the password box may not be empty.

Choose **Create** when you're done, and the directory you specified, and user database are created. The dialog box then changes to the standard Mail Setup layout.

Create a new Icon for CLint++ in **Program Manager** with the command line modified to read:

```
...\CLINTW.EXE /uADMINID
```

where ADMINID stands for whatever **ID** you chose for the Administrator account. When you need to perform administrator actions, run this special Icon instead of the standard Icon. You will of course have to supply a Password before performing Administrator actions.

After performing any actions you need to do as Administrator - which as a minimum must include creating a normal user account for yourself - exit CLint++, and then restart it normally, and proceed to connect yourself to the network mail directory as in Connecting to a Network Directory.

Connecting to a Network Directory

Before you can use CLint++ for mail or report database management, you must connect to an existing mail directory on a shared network drive visible to you. You may need help from your System Administrator to find out what drive letter and path to use. Also, your Administrator must create an account for you before you try to connect.

Choose Options | Mail Setup, and enter the full path of the network mail directory in the **System Admin Directory** box. Enter your full name in the **Full Name** box, enter an ID in the **ID** box (max. 8 characters) which must match the ID specified by your Administrator, and enter your initial password (obtained from your Administrator) in the **Password** box - your Administrator may have left your password empty.

Choose **OK** when you're done, and you are prompted to re-enter your password in the Password dialog. Do this, choose **OK**, and the dialog box then changes to the standard Mail Setup layout. If you receive messages about 'unknown users', you may have miss-typed your ID. If you can't get a connection, see your Administrator, and have him do it for you.

Creating Work Groups

When the user database was created with Creating a Network Directory, CLint++ created four groups for you:

Administrator

This is the person who created the database. They have permission to do anything, and those permissions can't be changed. Additional users can be assigned to the Administrator group, but this is not recommended.

Manager

A manager can perform any actions other than configuring database fields. The Manager group permissions can be set by the Administrator to meet your local requirements.

Engineer

An Engineer can originate reports, and re-assign reports assigned to them. The Engineer group permissions can be set by the Administrator to meet your local requirements.

Tester

A Tester can originate reports. The Tester group permissions can be set by the Administrator to meet your local requirements.

As Administrator, or any user with Edit Group permission, you can change the titles and permissions of any group, or with Create Group permission, create new groups using Network | Groups.

Creating Users

When the user database was created with Creating a Network Directory, CLint++ created the Administrator user for you.

As Administrator, or any user with Edit User permission, you can change the name, ID, and permissions of any user, or with Create User permission, create new users using Network **|** Users.

Creating Project Databases

Initially, there are no projects in the database. As Administrator, or any user with Create Project permission, select Network | Project, and then choose **New**, which pops up the Edit Project dialog. Enter the **Project ID** (maximum 8 characters) which may be the same as some user ID, but must be unique within the list of projects. Enter the **Full Name** of the project, and then optionally assign some users to the project - but be aware that only assigned users can see the project and receive notifications.

The project itself is a repository for up to four sub project databases, and the assigned list of users.

Close the dialog with **OK**, and choose Subproject which pops up the Sub Project Name dialog. Enter the **Short Name** (maximum 8 characters), which must differ from any other sub projects in this project. Enter the Full Name of the project, and choose **OK**.

The sub project database does not exist until the first action is performed on it, which creates it. These actions may be any of Submit, Import, or Configure, all of which are available when you select the sub project line in the project list. Once created, you can use Intray, Query, and Reports to examine records in the database, and Export to export any or all of the fields in all records.

It is important that you configure the new database to your needs before creating reports, or importing reports. Don't forget to use Configure first!. Although you can configure a database after it contains reports, this could cause confusion amongst users.

Creating Reports

Before you can operate on any reports, you must create some! There are two ways of doing this:

Import

You can import reports from any tool (like Excel) which supports the comma separated (CSV) export format. Often, there will be existing databases stored in this form which are better made 'live' and on-line through CLint++ using **Import** from the Network | Project dialog.

Submit

You can submit reports from the **Submit** button of the Network | Project dialog, or the Submit menu entry of either the Intray Window or Query Window.

Viewing Reports

Once you have some reports (see Creating Reports), you can view them in three ways:

Query Window

You can open the Query Window from the Network **|** Project dialog **Query Window** button. Once open, you can perform queries on the data, sort the window contents, print the window contents, and view any listed report in detail.

Intray Window

You can open the Intray Window from the Network **|** Project dialog **Intray** button; or if you have outstanding notifications with Network | Notifications. Once open, you can sort the window contents, print the window contents, and view any listed report in detail.

Report Window.

You can open the Report Window from the Network **|** Project dialog **Reports** button. Once open, you can view a historic display in one of six graphical styles of reports against time, and print any of these graphs.

Querying Reports

You open the Query Window from the Network | Project dialog **Query Window** button. Once open, you can perform queries on the data by right-clicking on the window, and selecting Queries.

CLint++ for Windows

CLint++ is a tool which performs all the operations a professional programmer requires:

- Full featured programmer's editor with configurable syntax coloring for any language.
- Support for version control tools, with PVCS and MKS RCS directly supported.
- Able to run compilers, Make, and other DOS programs and collect the output.
- Checking of C/C++ source programs far more thoroughly than any compiler.
- Locating all C/C++ function definitions in your code for rapid browsing.
- Handling e-mail between members of teams.
- Managing defect, requirement, or other databases for team members.
- Generation of Makefiles.
- Full support for the ANSI/ISO extension.
- Full support for the Standard Template Library.

When source checking, it reads the source files of your **C** or **C++** programs, and generates reports about possible problems. As it can look at *all* the files in your program, it is able to report far more than any C/C++ compiler. Use **CLint++** while developing your programs, and you'll be assured of type safe linkage, efficient code, and excellent portability.

CLint++ understands all the industry standard extensions to the **ANSI** standard such as **near**, **far**, **huge**, **cdecl**, **pascal**, **asm**, and so on for all the supported compilers. It understands and supports the **Borland** pseudo registers, and segment types, and the **Microsoft** special segment control extensions.

A special feature of **CLint++** is that it does *not* support models. Instead, all type modifiers are unique, that is:

```
char *
char far *
char near *
char huge *
```

and all variations are considered unique pointer types. The advantage is that regardless of the model you use for any project, if **CLint++** produces no warnings, then you can safely change models with no ill effects. At the same time, it understands that **near** pointers may be converted to **far** or **huge** by default. For those of you that have had problems with mixed model programming, this feature alone will justify your purchase!

CLint++ supports the **F1** Help key in all dialogs and windows, and presents concise, relevant help on their features. When in doubt, hit **F1**!

To start with, see Getting Started, Configuring CLint, and System Administrator.

CLint++ for DOS

CLint++ for Windows includes CLint++ for DOS to allow you to perform source checking and report generation from Makefiles and other DOS based tools. This command line program reads and understands fully the CLINT.CFG and *.CLN files made and used by CLint++ for Windows. Both programs use and understand the same switches, but CLint++ for Windows does not accept switches on the command line (other than as described in System Administration).

The command line syntax is:

```
            CLINT [switches] files
```

If filenames are given with extensions, those extensions are used. If not, '**.C**' is assumed as an extension. CLint++ searches for a *.CLN lint file in the current directory if there is not one explicitly given on the command line. If found, the options in the lint file are read into CLint++ prior to parsing the source files. If no filename arguments are present on the command line, and the file PROJECT.CLN is present in the current directory, it is used as a source of filename arguments also.

Filenames may be prefixed with '**+**' which forces them to be considered C++ source regardless of extension, or if prefixed with '**=**' forces them to be considered as C. The default is to consider files with ***CPP*** extensions as C++, otherwise as C. This works also in *.CLN Lint Files.

The switches are:

-3	Support 32 bit **int**egers
-Ab	Support ANSI/ISO **bool** type (C++)
-Ad	Deprecate the use of access declarations (C++)
-Ah	Support ANSI/ISO header names - no extension required (C++)
-Ai	Support ANSI/ISO **if**, **while**, and **switch** local declarations (C++)
-Ak	Support ANSI/ISO keywords (and, or, etc.) (C++)
-An	Support ANSI/ISO **namespace** and **using** keywords (C++)
-At	Deprecate the use of global **static** (C++)
-C	Support nested C comments
-Dname	Specify command line #define
-e0	Remove the error count limit
-eNNN	Specify an error count limit
-E	Generate preprocessor output
-F	Define function types
-G	Allow **sizeof** or casts in #if and #elif
-H	Disable Borland style pseudo registers
-Ipath	Specify include *path*s
-Jname	Define an environment name specifying include paths [not Windows]
-Kb1	allow Microsoft const/volatile declarations
-KC1	Specify that all files are 'C'
-KcN	Specify character constant length *N*
-KiNNN	Specify identifier length *NNN*
-KL1	Support long long data type
-Ks1	support 'asm' statement and function
-Lfile	Generate checking library *file*
-lfile	Use checking library *file*
-M	Generate dependency output
-nname	Define a *name* to ignore when **typedef**ed (C)
-ofile	Specify an output *file*
-p	Generate prototypes
-R	Generate metric reports
-T	Suggest symbols which could be static (obsolete switch, see -wsgs)

-Uname	Delete a previously defined symbol
-u	File or files are not a complete program
-v	Generate variable declarations
-w0	Remove the warning count limit
-waib	Allow any integral type in bitfields (C)
-wall	Enable all warnings
-wans	Warn of ANSI/ISO keywords (C++)
-wasd	Warn of array size for **delete** ignored (C++)
-wbbr	Warn of non ANSI initializer bracing
-wbnc	Warn of base initialization without class (C++)
-wcil	Warn of long constants which need an 'L' suffix
-wcom	Warn about nested comments
-wcon	Warn of constant conditions
-wcpc	Warn about C++ comments in C programs
-wduc	Warn of degenerate unsigned comparisons
-webb	Allow {} as empty body instead of ';'
-webr	Warn of **else** without brace
-weib	Allow enums in bitfields (C++)
-welb	Warn of empty loop, **if**, or **else** bodies not using ';'
-wenu	Warn of problems in enumeration usage (C)
-wfbr	Warn of **for** without brace
-whdr	Warn of problems in user header files
-wiag	Warn of initializing **auto** aggregates
-wibr	Warn of **if** without brace
-winc	Warn about multiply included headers
-wlfb[NNN]	Warn of function bodies exceeding *NNN* lines
-wlib	Warn of redeclared runtime library symbols
-wlnf	Warn of fields longer than 16 bits
-wmxp	Warn of mixing pointers to differing character types
-wnac	Warn of non ANSI printf/scanf conversions
-wNNN	Specify a warning count limit
-wnub	Allow unreachable **break** statements
-wnus	Warn of symbols defined but not used.
-wosf	Warn of old style function definition (C++)
-wpre	Warn of prefix operator used as postfix (C++)
-wprn	Check printf/scanf like function calls
-wpro	Warn of functions with no prototype
-wpsc	Warn of objects previously declared **static**
-wpua	Warn of possibly unintended assignments
-wreg	Warn of **register** usage
-wret	Warn of return problems
-wrtm	Warn of reference temporaries (C++)
-wrvi	Warn of ignored function return values
-wscg	Report **static** symbols with the same name as globals
-wsgs	Suggest symbols which could be **static**
-wshd	Warn of problems in system header files
-wsig	Warn of possible loss of significance
-wspc	Advise on white space usage
-wstv	Warn of **struct**s or **union**s passed or returned by value
-wtni	Warn if function throws type not in specifier (C++)
-wtns	Warn if function throws but no throw specifier (C++)
-wtri	Warn about ANSI trigraphs
-wtyb	Warn of **typedef**ing basic types not in lowercase (C)
-wtye	Warn of **typedef**ing **enum**s (C)
-wtyg	Warn of **typedef**ing complex types (C)
-wtyp	Warn of **typedef**ing pointer types (C)

-wtys	Warn of **typedef**ing **struct**s without leading uppercase name (C)
-wunr	Warn of unreachable code
-wuts	Warn of untyped functions or variables
-wwbr	Warn of **while** without brace
-X	Generate a full symbol report
-Y	Support **extern register** storage class
-Z	Suppress running filename report (not Windows)

-Ipath (set include path)
-ipath

where *path* is a directory name. Case may be important in directory names depending on the operating system you use. It specifies a directory to search for header files. There may be as many of these options as required. The order of search for:

```
        #include "file"
```

is

directory containing the source file,
current directory,
-I directories given on the command line,
-I directories given in a CLN file,
-I directories given in CLINT.CFG.

and for:

```
        #include <file>
```

is

-I directories given on the command line,
-I directories given in a CLN file,
-I directories given in CLINT.CFG.

You can force CLint++ to forget all the **`-I`** options in CLINT.CFG by giving '**`-I-`**' or '**`-i-`**' on the command line prior to any other include paths.

-Dname (define preprocessor symbol)
-dname
-Dname=value
-dname=value

These options define preprocessor macros. Although the option letter is case insensitive, the macro names and values are NOT. The first form defines a macro with default replacement text '**1**'. The second form gives the replacement text explicitly. For your convenience, you may append several definitions to an option with '`;`' like this:

```
-Dname1;name2=4;name3=5;name4
```

Note that the DOS command line can't handle options like:

```
-Dname="1"
```

properly.

-Uname (undefine preprocessor symbol)

This option deletes a preprocessor macro. It is not an error to specify a macro which was not defined. It is an error to attempt to undefine:

'__TIME__', '__DATE__', '__FILE__', or '__LINE__'.

But it can be used to undefine

'__CLINT__' , '_lint', or '__DOS__'

or any symbols defined in **CLINT.CFG**. You can append several names with '`;`' like this:

```
-Uname1;name2
```

The order of `-D` and '`-U`' is important - you may only undefine macros whose definition precedes the '`-U`' option.

-o file (specify output file)
-O file
-ofile
-Ofile

This option specifies a file to receive **CLint++**'s output. By default the output is written to **stdout**. If you use this option, **CLint++** still writes the source file name it is working on to **stderr** to show you how it is proceeding. If more than one of these options occur, the rightmost one is used.

-e0 (remove error count limit)

By default **CLint++** stops reading a file if 15 errors are produced, and proceeds to the next file with the message 'Too many errors'. If you use this option the limit is removed.

-eNNN (set error count limit)

Use this option to specify the error limit you need. The default is 15. If this option and '**-e0**' occur together, the rightmost one is obeyed. The maximum value you can specify is 255.

-w0 (remove warning count limit)
-W0

By default, **CLint++** stops reading a file if 60 warnings are produced, and proceeds to the next file with the message 'Too many warnings'. If you use this option the limit is removed.

-wNNN (set warning count limit)

Use this option to specify the warning limit you need. The default is 60. If this option and **-w0** occur together, the rightmost one is obeyed. The maximum value you can specify is 255.

-wall (enable all warnings)

A number of **CLint++**'s warnings are optional, and OFF by default. This turns all warnings on.

-w-all turns all warnings off.

-wnus (symbols defined but not used)

On by default. This enables warnings about symbols defined but not used.

-wpsc (previously static or previously extern)

On by default. This enables warnings about **static** declarations previously declared **extern** (C only, in C++ this is an error), or non-static declarations previously declared **static**.

-wtri (trigraphs)

On by default. This enables warnings about ANSI trigraphs. This capability was added to the standard to allow users with old style input devices to create characters not present on such keyboards. The standard further states that scanning for and replacement of trigraph sequences precedes all other operations, so trigraph replacement occurs in comments. **CLint++** however does not warn about trigraphs in comments.

??=	replaced by '#'
??/	replaced by '\'
??'	replaced by '^'
??(	replaced by '['
??)	replaced by ']'
??!	replaced by '|'
??<	replaced by '{'
??>	replaced by '}'
??-	replaced by '~'

-wcom (comments)

On by default. This causes CLint++ to object to C comments nested inside C comments like this:

```
/*      /*      */      */
```

The standard specifically states that comment's don't nest. However, if you use the **-C** switch, CLint++ allows comments to nest and disables this warning. CLint++ also supports C++ style comments in C.

-wcpc (C++ comments)

Off by default. This causes CLint++ to object to C++ comments in C files like this:

```
// comment
```

This is provided to help enforce local coding standards which prohibit the use of C++ comments in C code.

-wrvi (return value ignored)

Off by default. This causes CLint++ to note calls of functions which return a value where the result is not used, like this:

```
int f();

void g()
{
f();
}
```

where the return value of the call of **f()** is not used.

-winc (included files)

On by default. This causes CLint++ to object when a header file is directly or indirectly included more than once - this means that if you include a header which re-includes a header you already included, you'll get a warning. This warning is given to try and improve compile times - multiply including headers wastes your time, although with properly written headers there won't be a problem. However, older compilers sometimes have header files which don't comply with the ANSI requirements, and multiple inclusion can cause problems.

-wfbr (for requires brace)

Off by default. When enabled, CLint++ objects to a **for** statement where the inner loop body is not braced like this:

```
for (i = 0; i < 5; i++)
j = i;
```

We consider that all loop bodies should be braced for clarity. CLint++ won't object to an empty loop body if it simply the null statement '**;**'. You may prefer that empty loop bodies use '**{}**', in which case you should enable **`-webb`**.

-wwbr (while requires brace)

Off by default. When enabled, CLint++ objects to a **while** statement where the inner loop body is not braced like this:

```
while (i != 0)
j = i--;
```

We consider that all loop bodies should be braced for clarity. CLint++ won't object to an empty loop body if it simply the null statement '**;**'. You may prefer that empty loop bodies use '**{}**', in which case you should enable **`-webb`**.

-wibr (if requires brace)

Off by default. When enabled, CLint++ objects to an **if** statement where the inner body is not braced like this:

```
for (i = 0; i < 5; i++)
j = i;
```

We consider that all bodies should be braced for clarity. CLint++ won't object to an empty body if it simply the null statement '**;**'. You may prefer that empty bodies use '**{}**', in which case you should enable **`-webb`**.

-webr (else requires brace)

Off by default. When enabled, CLint++ objects to an **else** statement where the inner body is not braced like this:

```
for (i = 0; i < 5; i++)
j = i;
```

We consider that all bodies should be braced for clarity. CLint++ won't object to an empty body if it simply the null statement '**;**'. You may prefer that empty bodies use '**{}**', in which case you should enable **`-webb`**.

-wspc (advise on space usage)

Off by default. When this switch is on, CLint++ objects to **if**, **return**, **switch**, **for**, **while**, and **do** not followed by white space; and to '**=**' (assignment), '**:**' (colon), '**?**' (question mark), '**;**' (semicolon), and '**,**' (comma) not followed by white space unless directly followed by '**)**' or '**;**'. These warnings are intended to highlight coding styles which are dense and hard to read. It particularly applies to constructs like:

```
if(i)
```

which looks enough like a function call to confuse the eye at first glance.

-wiag (initializing auto aggregates)

On by default. CLint++ considers initializing automatic aggregates (structures or arrays) to be wasteful. Be aware that when you do this, the compiler generates a hidden unnamed copy of the object, and then copies that object over your automatic variable on each function entry. If you don't modify the object, consider making it static, which will initialize it once at compile time.

-wtye (typdef of enum)

Off by default. When enabled, CLint++ considers it poor practice to typedef an enumerated type, as this makes it hard to distinguish an enumerated type from other types when reading the source. (C only).

-wtys (typedef of structs)

Off by default. When enabled, CLint++ considers that typdefing structures without at least a leading capital to be misleading. The form **-wtysU** requests CLint++ to object to such typedefs which are not fully in uppercase. (C only).

-wtyp (typedef of pointers)

Off by default. When enabled, CLint++ objects to typedefs of pointer types, on the grounds that too much information is hidden. (C only).

-wtyb (typedef of basic types)

Off by default. When enabled, CLint++ objects to typedefing primitive types like **int** other than in lower case, on the grounds of possible confusion. (C only).

-wtyg (typedef of general types)

Off by default. When enabled, CLint++ objects to overly complex typedefs such as pointers to functions, and arrays, on the grounds of possible confusion. (C only).

-wunr (unreachable code)

On by default. CLint++ considers code which can't be reached to be a problem. In some obscure cases (such as jumping into the body of a loop that can't be entered from the top), CLint++'s analysis is wrong. In such cases you can silence CLint++'s objections by preceding the statement with:

```
#pragma notreached
```

or

```
/*NOTREACHED*/
```

which informs CLint++ that this is intentional.

-wstv (struct/union passed/returned value)

On by default. When you pass or return a structure value, slow code which uses additional stack space at least equal to the size of the object is generated. Also, some earlier compilers could not correctly generate code for recursive functions when this occurred.

-wuts (untyped symbols)

On by default. The default type is **int**, but C++ deprecates the practice of not typing symbols. We warn of this for C also.

-welb (empty loop bodies)

On by default. CLint++ considers that empty bodies of **for**, **while**, **do**, **if**, or **else** statements is wasteful and possibly a bug, unless such empty bodies are the null statement '**;**'. If you prefer to use '**{}**' for empty bodies, use **`-webb`**.

-wcon (constant condition)

On by default. CLint++ considers that conditional expressions which evaluate to constants is wasteful and possibly a bug. It objects to statement like:

```
if (1 >= 0)
```

(which might arise from macro expansion) as a constant condition, and to:

```
unsigned j;
while (j > 0)
```

as a degenerate comparison, since unsigned values can't ever be negative: such comparisons are equivalent to comparison for equality with 0.

-wduc (degenerate unsigned comparison)

On by default. CLint++ considers that conditional expressions like:

```
unsigned j;
while (j > 0)
```

as a degenerate comparison, since unsigned values can't ever be negative: such comparisons are equivalent to comparison for equality with 0. It is ignored unless **`-wcon`** is set.

-wmxp (mixing pointers to different char types)

On by default. CLint++ considers char, unsigned char, and signed char to be different types, so it complains of the use of a pointer to one of these when another was specified like this:

```
char *cp;
unsigned char *ucp;

cp = ucp;
```

Although all char types are the same size, the effect of sign extension when the values are used can cause problems unless you are quite specific about which type you use. You can silence this objection with a cast.

-whdr (warn in user headers)

On by default. CLint++ considers that warnings should be issued about dubious practices in your headers files, on the grounds that headers are your code too.

-wshd (warn in system headers)

Off by default. CLint++ suppresses warning about dubious practices in system header files because unfortunately too many vendor headers would otherwise produce a barrage of objections.

-wprn (check printf/scanf)

On by default. CLint++ warns checks the format strings and arguments of printf(), fprintf(), sprintf(), scanf(), fscanf(), and sscanf() according to the ANSI requirements. All these functions can be a prolific source of problems in your code, and you should never disable this.

-wcil (constant is long)

On by default. CLint++ objects to constants like 1234567 which are outside the range which can be handled by an **int** type. For values in the range 32768 through 65535 which can be placed in an **unsigned int**, you can use the suffix '**U**', otherwise you should use the suffix '**L**'.

-wenu (check enumeration usage)

On by default. CLint++ checks all comparisons and assignments of enumerated types, and objects if the values used are members of different enumerations, or not enumerations. You can suppress these warnings with casts.

-wpro (call requires prototype)

On by default. CLint++ considers that all function calls must be made to functions which have been previously been prototyped. If you have difficulty constructing these prototypes, consider using the **-p** option to generate the prototypes for you.

-wret (check return)

On by default. CLint++ checks that all functions return the values they are supposed to. It objects to missing **return** statements on functions which should return values, and on both returning a value and not returning a value in the same function.

-wlfb[NNN]   (long function bodies)

Off by default. When enabled, CLint++ will inform you of any functions it sees with more than NNN lines (or 100 lines if NNN is not specified). This helps identify large functions which should perhaps be simplified by breaking into subroutines.

-wsig (possible loss of significance)

On by default. CLint++ considers assignment of **long** values to shorter integral types without a cast to be a possible error. The form **-wsig+** considers assignment of any type larger than the destination to be a possible error.

-wlib (symbol hides library definition)

On by default. CLint++ considers that if you define a function or symbol which is defined in your vendor's runtime library, that this is a possible error. Sometimes you must provide a function of the same name which then hides the runtime library definition, in which case you should accept the warning or disable this option.

-wlnf (long field)

On by default. CLint++ considers **int** objects to be 16 bits unless you use the -3 (32 bit integers) switch, and then objects to bit-fields longer than 16 bits. You should be aware that the ANSI standard leaves all aspects of bit fields to the vendor to implement as they see fit, so code using bit fields must be considered highly non-portable.

-wreg (register used)

On by default. CLint++ warns about all **register** declarations because with modern compilers you are almost always better off leaving the compiler to decide how to use registers. Legacy code often used register declarations to try and force primitive compilers to generate slightly faster code, but with modern compilers you will almost always force the compiler (if it honors register requests) to generate worse code.

-wpua (possibly unintended assignment)

On by default. CLint++ considers assignment in a conditional context where the result is not compared with a constant to be a possible error, for example:

```
            if (i = 5)
```

might well have been intended to be:

```
            if (i == 5)
```

However, if such assignments are written as:

```
            if ((i = 5) != 0)
```

then CLint++ considers this to be an intended assignment.

-wbbr (non ANSI initializer bracing)

On by default. The ANSI standard considers it good practice to enclose each substructure element in an initialization in braces (as for example when initializing an array of structures). CLint++ warns if this is not done.

-wnac (non ANSI printf/scanf conversion)

Off by default. Most DOS compilers extend printf() and scanf() conversions to allow the use of 'N' (near) and 'F' (far) when specifying a pointer format. These are non-ANSI conversions, and will cause a complaint if you enable this option.

-webb (empty body is braces)

Off by default. When any of **`-wfbr`**, **`-wwbr`**, **`-wibr`**, and **`-webr`** are used, CLint++ complains about empty loop bodies which are not the null statement ';'. If you prefer '{}' as your empty loop body, enable this option.

-wnub (unreachable break allowed)

Off by default. When **-wunr** is enabled (see earlier), CLint++ warns of unreachable statements. However, your local coding standard may require that all cases in a switch end with break even if not reachable. If you enable this option, such break statements won't provoke an unreachable code warning. For example:

```
switch (j)
{
case 1:
goto label;
break;
}
```

-wsgs (suggest statics)

On by default. After reading all source files, CLint++ checks all symbols for compatible definitions in each of the file, and with the runtime library. During this check, any globally visible symbols used in only one file are noted as being possible **static** symbols. This does not apply to member functions which always have external linkage.

-wscg (statics conflict with globals)

Off by default. After reading all source files, CLint++ checks all symbols for compatible definitions in each of the files, and with the runtime library. During this check, any globally visible symbols with the same name as **static** symbols are detected if this switch is enabled. You should rename such static symbols to avoid possible conflicts.

-waib (allow integral bitfields)

On by default. When enabled, any integral type may be used to declare a bitfield. Note that the ANSI C standard requires that bitfields may only be **int**, **unsigned int**, or **signed int**.

-weib (allow enumeration bitfields)

Off by default. When enabled, any enumerated type may be used to declare a bitfield. Note that the ANSI C standard requires that bitfields may only be **int**, **unsigned int**, or **signed int**.

-wans (warn of ANSI/ISO keywords)

Disabled by default. It can only be enabled if -Ak ***support ANSI/ISO keywords*** is enabled. When enabled use of the ANSI/ISO keywords is reported.

-wasd (warn of array size in delete[])

On by default. The use of array sizes in **delete[]** is now obsolete.

-wbnc (warn of base initialization without class)

On by default. The initialization of base classes in constructors without using the base class name is now obsolete.

-wosf (warn of old style function definition)

On by default. Old-style function definitions in C++ are obsolete, and deprecated in ANSI C.

-wpre (warn of prefix operator used as postfix)

On by default. In older versions of C++ you could not distinguish between the pre- and postfix versions of ++ and --. This feature is now considered obsolete.

-wrtm (warn of reference temporaries)

Off by default. When enabled, warnings are generated when a reference temporary is generated to initialize a reference. This is often a cause of slow program performance, however be aware that many vendor libraries do this ***lots***.

-wtni (warn if function throws type not in specifier)

Off by default. When enabled, warnings are generated about functions which throw types not declared in their prototypes. This is off by default because many vendor header files don't get this right.

-wtns (warn if function throws but no throw specifier)

Off by default. When enabled, warnings are generated about functions which throw types but have no throw specifier. This is off by default because many vendor header files don't get this right.

-whvf (hidden virtual functions)

On by default. When enabled, warnings are generated when a function declaration hides a virtual function of the same name inherited from a base class. Some vendor header files do this (Microsoft MFC headers), so you may want to turn it off for them.

-Ak (support ANSI/ISO keywords)

Disabled by default. When enabled the ANSI/ISO extension keywords are enabled for C++.

-Ab (support ANSI/ISO bool type)

Disabled by default. When enabled the ANSI/ISO **bool** type is enabled for C++, and also the truth constants **true** and **false**.

-An (support ANSI/ISO namespace and using)

Disabled by default. When enabled the ANSI/ISO **namespace** and **using** keywords are enabled for C++. These keywords render the use of global **static** declarations (**-At**), and access declarations (**-Ad**) redundant, and their use is deprecated in ANSI/ISO C++.

-At (deprecate global static)

Disabled by default. When enabled you are warned when global **static** declarations are used. The unnamed **namespace** (**-An**) renders these declarations redundant.

-Ad (deprecate access declarations)

Disabled by default. When enabled you are warned when access declarations are used. The unnamed **namespace** (**-An**) renders these declarations redundant.

-Ah (support ANSI/ISO include names)

Disabled by default. When enabled the ANSI/ISO header style (where extensions for headers may be omitted) is supported. This is required for some implementations of the Standard Template Library (STL).

-Ai (support ANSI/ISO if, while, etc.)

Disabled by default. When enabled, you may declare local variables in the condition part of **if**, **while**, **for**, and **switch**. All such declared names go out of scope when the construct ends. This changes the prior meaning of local declarations in **for**, which continued to exist until the end of the enclosing block.

-L file

This important option instructs **CLint++** to change it's behavior, in that it knows the source files specified will be headers, not program text. Each prototype or external declaration in the source files is written to the **file** in a form to be later read into **CLint++** using the **-l** option. This allows CLint++ to check all usage in your program of vendor supplied library functions. It was used to prepare the CLB library files shipped with CLint++, and is used by the batch files shipped with CLint++ to build or rebuild CLB files.

Note that many errors will be reported if this option is applied to program source, and that the '`-L`' MUST be upper case. The file name case is only important if your operating system requires.

By default only those symbols prototyped or declared **extern** in the specific files are written to the library file as definitions - symbols from included headers are not written to the output. See **-L-** and **-L+** below if you want symbols in included headers defined also.

For **C++**, all non-**inline** member functions, and **static** members are written to the library file as definitions. The assumption is that a library defines all the member functions in its headers.

There are two additional forms for use when preparing libraries from vendor supplied header files:

-L-file

This form specifies that the input is vendor library header files, and that definitions in included headers are to be written to the library file also. Absolute pathnames of the header source files are embedded in the library file for use in error reporting.

-L+file

This form specifies that the input is vendor library header files, and that definitions in included headers are to be written to the library file also. Only the filenames of the headers are embedded in the library file for use in error reporting. This is the form used by us to prepare CLint++ library files.

When preparing a library file of your own, one of the best ways is to create a Lint File using Lint | New Lint File, and than drag all the source files of your project into it, then choose Lint | Generate Library, and save the resulting file as MAKEFILE.MAK. Then from a DOS box in the project's directory type **make lintlib**. If you chose a name other than this, you must type **make -f *filename* lintlib**. The output file will have the name ***project*.CLB**, where *project* is the name of the CLN file you created.

You can drag any *.CLB file into a Lint File Window. This has exactly the same effect as adding the *.CLB file to the project using Lint | Edit Lint File, choosing the ***New*** button in the ***Checking Libraries*** group, and browsing for the file - but much easier.

-lfile

This option instructs **CLint++** to read a definition file previously prepared with the `-L` option after reading all the source files. Declarations and usage in your source files will be compared to those recorded in the definition file, and may result in the warning:

Warning: 'name' in file(NNN) hides a library symbol in 'file'

If this happens, you have defined a function or variable which is also defined in a library. C allows this, but in some cases it may be a bug in your code. You may suppress this warning with the `-w-lib` option.

You may also use `-l` to specify a directory rather than a file, which adds the directory to the list. CLint++ use to search for libraries when a full path is not given.

-p (generate prototypes output)

-p[=]

-P[=]

Write prototypes of all externally visible functions declared in your source files to 'stdout'. If '**=**' is given, suppress the first dimension in array declarations.

-p[=]file

-P[=]file

Write prototypes of all externally visible functions declared in your source files to 'file'. If '**=**' is given, suppress the first dimension in array declarations.

-p[=]+

-P[=]+

Write prototypes of all functions declared in your source files to 'stdout'. Functions declared '**static**' will have '**static**' prototypes. If '**=**' is given, suppress the first dimension in array declarations.

-p[=]+file

-P[=]+file

Write prototypes of all functions declared in your source files to 'file'. Functions declared '**static**' will have '**static**' prototypes. If '**=**' is given, suppress the first dimension in array declarations.

-p[=]-

-P[=]-

Write prototypes of all static functions declared in your source files to 'stdout'. They will all have static prototypes. This is intended as a quick way of 'ANSIfying' old-style C programs. You'd then read the resulting file into the original source file to provide local prototypes. If '**=**' is given, suppress the first dimension in array declarations.

-p[=]-file

-P[=]-file

Write prototypes of all static functions declared in your source files to 'file'. They will all have static prototypes. If '**=**' is given, suppress the first dimension in array declarations.

-v (generate variables output)

-v[=]

-V[=]

Write declarations of all externally visible variables declared in your source files to **stdout**. If '**=**' is given, suppress the first dimension in array declarations.

-v[=]file

-V[=]file

Write declarations of all externally visible variables declared in your source files to **file**. If '**=**' is given, suppress the first dimension in array declarations.

-v$[=]

-V$[=]

Write declarations of all externally visible variables declared in your source files to **stdout** prefixed by **extern**. If '**=**' is given, suppress the first dimension in array declarations.

-v$[=]file

-V$[=]file

Write declarations of all externally visible variables declared in your source files to **file** prefixed by **extern**. If '**=**' is given, suppress the first dimension in array declarations.

-E[file]   (generate preprocessor output)

Use **CLint++** as an **ANSI** C pre-processor. The output is written to **stdout** [or **file**]. The preprocessed output preserves comments and white space. It is useful to see the effect of macro expansion on the text. The output may be given to CLint++ as later input, and will produce the exact same reports that examination of the original files would have done. If multiple files are specified on the command line, each is preprocessed in turn, and the outputs concatenated.

-M (generate dependency output)

-M[file]

-m[file]

Write the non-system dependencies to **stdout** [or **file**]in a form suitable for use by **MAKE**. Only headers included by **#include file** will appear.

-M1[file]

-m1[file]

Write all dependencies to **stdout** [or **file**] in a form suitable for use by **MAKE**. All included files will appear.

-nname (specify typedef name to ignore)
-Nname

This defines a name for which 'typedef' checking should not occur. It can be used to prevent warnings about names which you consider to be inappropriate. **CLint++** already knows the names of all **ANSI** or **Windows** typedefs, and never complains about them. This is most useful in your **PROJECT.CLN** or **CLINT.CFG** files. (C only).

-R (generate effort metric output)

-R[:file]

This causes **CLint++** to write a report to **stdout** [or **file**] for each source file showing the notional complexity of each function in the file (using Halstead's Effort metric), and useful additional data including the use of **goto**s and **continue**s, and number of loops.

-RNNN[:file]

-RwNNN[:file]

Many functions are so small as to be uninteresting. You can suppress effort reports on all functions with less effort than **NNN** with these option. A value of 5 will usually omit uninteresting functions.

-F (specify function properties)

-F0name

CLint++ knows that **exit()**, **_exit()**, **abort()**, and **longjmp()** can't return, so any following statements are unreachable. If you define functions of your own which don't return, use this switch to tell **CLint++** about them for full flow of control checking. This is most useful in your **PROJECT.CLN** or **CLINT.CFG** files.

-F1name

CLint++ knows that **printf()** is a function which has a format argument followed by a variable number of arguments, and will check these against the format. If you create any functions like this using **vsprintf()** or similar functions, use this switch to tell **CLint++** about them for full checking. This is most useful in your **PROJECT.CLN** or **CLINT.CFG** files. In general the digit '1' may be replaced with any digit from 1 to 9 to specify the number of fixed arguments to the function - for example **fprintf()** has 2. CLint++ knows about **printf()**, **fprintf()**, and **sprintf()**.

-F-1name

CLint++ knows that **scanf()** is a function which has a format argument followed by a variable number of arguments, and will check these against the format. If you create any functions like this using **vscanf()** or similar functions, use this switch to tell **CLint++** about them for full checking. This is most useful in your **PROJECT.CLN** or **CLINT.CFG** files. In general the digit '1' may be replaced with any digit from 1 to 9 to specify the number of fixed arguments to the function - for example **fscanf()** has 2. CLint++ knows about **scanf()**, **fscanf()**, and **sscanf()**.

-u (program is incomplete)

This switch informs **CLint++** that the files given are not complete, either they are a library or only part of a program. In any case, complaints about symbols used but not defined, and defined but not used are suppressed, and no recommendations about **static** symbols will occur. Use this switch when checking single files which are part of a larger program.

-T (suggest static symbols)

This switch tells **CLint++** to detect functions and variables which are only used in the file in which they are declared. It then prints a warning about all such variables, observing that they should be declared **static**. It is retained for backward compatibility, **`-wsgs`** is preferred in new projects.

-X[file]   (generate symbol report)

This switch causes CLint++ to write a listing off all globally visible symbols in each file to stdout [or the given file]. The output file has lines formatted as:

```
NNNNN:[function]  declaration . . .
NNNNN:[variable]  declaration . . .
```

where **NNNNN** is the line number inn the source file where the symbol was declared or defined. The declaration is in the same form as written by either the **-p** or **-v** switches. For example,

```
File 3d.c

   67: [function]   void identity(MATRIX *m);
   79: [function]   void mat_mul(MATRIX *mat1, MATRIX *mat2,
MATRIX *mat3);
```

-Y (support extern register)

This switch instructs **CLint++** to allow **extern register** as a new storage class. This storage class is often used by compilers for controllers. SETUP automatically adds this switch to your CLINT.CFG file when you specify the Intel IC96 compiler.

-Z (suppress file display)

By default, CLint++ displays the file it is reading on the last line of the display. When output is being captured, you should use this switch to suppress the many header file names that otherwise clutter the captured output. CLint++ will not show these file names when the stderr stream is a file, so the switch should only be occasionally required.

-Jname (specify include environment variable)
-jname

This option specifies the name of an environment variable which contains a set of semi-colon separated search paths for header files. You can place this option on the command line, or in CLINT.CFG, or PROJECT.CLN. When used, the search order is: the directory containing the source; the command line include paths (from **-I**) if any; and the path(s) given in the **name**d environment variable.

-C (support nested comments)

This option tells CLint++ to support nested comments. It also disables the **`-wcom`** warning option

-3 (32 bit integers)

This option tells CLint++ to treat all **int** types as 32 bit instead of 16. It causes complaints about long fields to occur only if fields longer than 32 are specified, and suppresses a number of other complaints which would occur with 16 bit **int** types. SETUP adds this switch to your CLINT.CFG file if you specify the Watcom or Intel iC86 compilers.

-G (allow sizeof and cast in #if)

This option allows the use of **sizeof()** and casts in **#if** and **#elif** preprocessor directives. Be aware that this is specifically prohibited by the ANSI standard. SETUP adds this switch to your CLINT.CFG file if you specify the Archimedes compiler.

-H (disable pseudo registers)

This option disables the Borland style pseudo registers _AX, _BX etc. SETUP adds this switch to your CLINT.CFG if you use the Mix Power C or Archimedes HC11 compilers.

-KcN (set character constant length)

This option specifies that character constants may be from 1 to **N** characters long. **N** may be in the range 1 to 4. The default is 2.

-KC1 (all files are 'C')

This option specifies that all files will be treated as **C** regardless of their extension. The default is that files with a ***.C** extension are **C**, all other files of the form ***.C??** are **C++**. Specific files may be forced to be **C** by prefixing their names with **'='**, or forced to be **C++** by prefixing them with **'+'**.

-Ks1 (support 'asm')

This option specifies that **asm** in all its forms is to be supported, and is the default. Some compilers expect **asm** to be a function declared for example like:

```
extern asm(char *);
```

which upsets CLint++ unless this option is disabled. Be aware that **asm** is a reserved word in **C++** according to the **ARM**, and so ought never to be disabled for **C++** checking.

-K=1 (allow inheritance of 'operator =')

This option specifies that - contrary to the **ARM** - declarations of **operator =()** in base classes are to be inherited in derived classes. This option is needed to support ***Microsoft C++ 7.0*** and ***Microsoft Visual C++*** which perpetrate this crime. As the commentary in the **ARM** shows, allowing this inheritance is extremely dangerous, but must be allowed to support these compilers. When you specify either of these compilers, CLint++ sets this option for you.

-Kb1 (allow Microsoft const/volatile)

This option specifies that - contrary to the **ARM** and ANSI C- that **const** and **volatile** are legal prefixes to an object name like this:

```
            extern int far volatile x;
```

According to the ARM, this declaration should be written:

```
            extern volatile int far x;
```

This is needed because all ***Microsoft*** and ***Intel*** compilers allow this misuse of **const** and **volatile**. When you specify any of these compilers, this option is set for you.

-KiNNN (set identifier length)

This options specifies the number of significant characters in identifiers. The default is 32. Values in the range 6 to 255 may be used.

-KL1 (support long long data type)

This option makes **long long** a legal data type.

Getting Started

To get started, you must first configure CLint++ as described in Configuring CLint, and optionally System Administration. Then, experiment by dragging some C or C++ files out of File Manager, and try out the Lint and Find Function features. As you'll see, when CLint++ finds errors or dubious constructs in your code, it opens a Message window and places the text of the messages there. If you double click on any such message, CLint++ will place you on the source file line and column at which it found the error. If you select a message and press **F1**, you'll get detailed help describing the problem. If you right click, you can also go to the source line, or get specific help.

CLint++ contains a modern programmer's editor with support for the popular Version Control tools. PVCS and RCS are directly supported, others will require some little work. You can Cut and Paste to or from CLint++ or any other Windows application - and unlike many Windows tools, CLint++ has no limit to how large a block of text you can cut and paste other than available Windows memory. Right clicking in an Edit Window pops up a menu with frequently used commands. CLint++ supports up to 1000 levels of Undo and Redo in each separate window.

CLint++ provides full syntax coloring for C, C++, DOS Batch files, and Makefiles. You can configure the color scheme using any of the standard 16 windows solid colors. You can modify existing or define new keywords for CLint++ to color. You can define entire new document types, the file extensions and filter strings, the color schemes, and keywords for other languages not directly supported such as Pascal, FORTRAN, and COBOL.. See Options | File Type Options for more detail.

You can associate a DOS command line with any document type such as a compiler. Output from the DOS tool is collected and presented in the Message window. For compilers or other tools which produce recognizable warning or error messages, you can go directly to the offending source line by double clicking on the Message window line.

CLint++ provides an email tool to keep members of your team in touch. See Network Menu for details.

CLint++ provides a database tool for defect, requirement, or other on-line report tracking. See System Administrator for more details.

In short, CLint++ is a unified programming tool for the 90's.

Configuring CLint++

You configure CLint++ in many ways. The main configuration options are on the Options menu.

Font	Select the Edit Window display font
Document Options	Select document specific options
Editor Options	Specify default editor options
File Type Options	Specify the properties of your document types.
Key Assignments	Assign CLint++ actions to key combinations
Lint Options	C or C++ source lint options
Version Control	Version control options
Mail Setup	Mail configuration options

In addition, you can specify your page printing options with:

Page Setup	In the File menu

Mail Setup command

The Mail Setup dialog is accessed from the Options menu to specify the location of the network mail directory, and mail and password options.

Admin Directory

This edit box allows you to specify the initial network mail directory when Creating a Network Directory, or the current network mail directory when Connecting to a Network Directory. After these initial actions have been performed, it shows the currently selected directory. Changing this entry later is ineffective.

Full Name

This edit box allows you to specify your name when Creating a Network Directory, otherwise it shows your name from the user database. Changing this entry later is ineffective.

ID

This edit box allows you to specify your network name when Creating a Network Directory or Connecting to a Network Directory, otherwise it shows your ID from the user database. Changing this entry later is ineffective.

Autosave read mail

This box is checked by default. When checked, all mail you don't explicitly save after viewing is saved for you in the mail folder **Autosaved Mail**.

Autosave sent mail

This box is checked by default. When checked, a copy of all mail you send is saved for you in the folder **Sent Mail**.

Discard password

This box is unchecked by default. When checked, your password is discarded from memory after each network transaction, and you'll have to re-enter it on each subsequent action. For most users, this box can be safely left unchecked, but users of Administrator or Manager accounts should not do this if the system is to be reasonably secure.

Create

This button is only present when Creating a Network Directory. It creates the initial user database with the entry for the Administrator.

Font Selection dialog

The Font Selection dialog is accessed from the Options menu to specify the font used by Edit Windows, and from the Page Setup dialog to specify the printer font. In either case, only those monospaced fonts supported by your display or printer are available for selection.

The default font for Edit Windows is **9 point Termina**l.

The default font for printing is **10 point Courier New**.

Font

This combo box allows you to specify the face name of a font. Only those fonts currently meaningful are visible.

Font Style

This combo box allows you to select the appearance of the font. The available selections depend on the Font you choose.

Size

This combo box allows you to specify the point size of the font. The available selections depend on the Font you choose.

Document Options dialog

This dialog is popped up by the Options | Document Options menu entry.

This dialog box lets you specify the specific options for the current document. If you have no document currently open, it can't be selected from the menu.

Tab Spacing (1-16)

You may select the tab stop settings on this range. The default is 4.

Right Margin (40-1000)

You may select the right margin position in the range 40-1000. The default is 80. This specifies where lines are wrapped if wrapping is enabled.

Wrap lines at margin

When set, CLint++ automatically wraps lines as you type when you move past the right margin. It only breaks lines at tabs or spaces.

Auto indent

When set, and when line wrapping is enabled, CLint++ automatically inserts enough white space to line up with the previous line.

Compress blanks to tabs on save

When set, CLint++ optimally converts blanks to tabs according to the current tab setting as your file is saved.

Strip trailing blanks on save

When set, CLint++ strips trailing white space from lines as your file is saved.

Save as default

If you set this and choose OK, CLint++ makes the setting the default for all new documents subsequently opened. Otherwise your changes only affect the current document settings.

Editor Options dialog

This dialog is popped up by the Options | Editor Options menu entry.

It lets you specify options affecting all documents, and to configure the toolbar for the window type which was active when this dialog popped up.

Cursor

Only one of the cursor options may be selected. It takes effect as soon as you choose OK.

Vertical Bar	The cursor is shown as a thin vertical bar.
Block	The cursor is shown as a block covering the whole character.
Underline	The cursor is shown as an underline below the character.

Backup files on save

When checked, CLint++ renames the file you are editing with a BAK extension, and creates a new file with your changes. If you need to return to the previous version, you just delete the new file, and rename the backup file to the original file's name.

Create blank file as default

When checked, CLint++ will offer to create an unnamed document when started without any loaded files or remembered documents.

Undo past earlier save

CLint++ saves up to 1000 of your changes which you can undo with Edit | Undo, and discards this undo information when you save your file. When checked, the undo information is not discarded when you save the file, in which case you can undo changes past the point at which the file was saved.

Restore session

When checked, CLint++ saves the current position and state of all document windows. When you next use CLint++, these windows will be restored exactly as they were when you last exited CLint++. However, the Undo information is not preserved.

Autosave every N minutes

When non-zero, CLint++ writes copies of changed files to the directory specified by your TEMP environment variable, or to C:\TEMP (which it creates if needed). In the unlikely event that CLint+ + or Windows crashes, the files you were editing may be recovered from that directory. CLint++ deletes these files when terminated, or when the Edit Windows they refer to are closed, or when you save your changes.

Enable Toolbar

When checked, a toolbar containing buttons you choose for each window type is drawn beneath the menu bar. The window types are the Edit Window, the Message Window, the Lint File Window, the Query Window, the Intray Window, the Report Window, and a special case for when no widows of any type are open. To specify the toolbar for a given window type, open a window of that type, and select the Options | Editor Options menu entry.

Action

This listbox shows all the available toolbar buttons. You copy a toolbar button to the **Toolbar** list by double-clicking a selection, or selecting one and choosing the **>>** button.

>>

This button inserts the tool highlighted in the **Action** list following the highlighted selection in the **Toolbar** list.

<<

This button deletes the tool highlighted in the Toolbar list.

Toolbar

This list shows the tools for the window type which was active when this **Editor Options** dialog was opened. You can add tools from the **Action** list, delete tools, or move tools up and down the list with the **Up** and **Down** buttons. The window types are the Edit Window, the Message Window, the Lint File Window, the Query Window, the Intray Window, the Report Window, and a special case for when no widows of any type are open.

Up

This button moves the currently highlighted tool in the **Toolbar** list up one place provided it is not the first entry in the list.

Down

This button moves the currently highlights tool in the **Toolbar** list down one place provided it is not the last in the list.

Gap

This button inserts a ***Separator*** (small empty space on the toolbar) following the currently highlighted tool in the **Toolbar** list.

Edit File Properties dialog

This dialog is popped up by the Options | File Type Options menu entry.

It allows you to modify the properties of existing document types, or create new ones.

Document Type

This list box contains a list of currently defined document types by name. You can edit the name by double clicking any entry. As you change the selection, other entries in the dialog change to reflect the setting for documents of the selected type.

Syntax Coloring

When checked, CLint++ will syntax color documents of this type according to the **Colors**, **Keywords**, and **Characters** properties described below.

Keyword Macros

When checked, CLint++ will observe you typing. If you type a space, tab, Return, or Escape when the text to the immediate right of the insertion point is the name of a **keyword Macro**, CLint++ then inserts the macro expansion text at the current insertion point.

Expand macro on space

When not checked, CLint++ only expands **keyword Macros** when you type Escape.

Preprocessing

When checked, CLint++ permits you to Preprocess documents of this type from Lint | Preprocess into a new Edit Window. You should only check this for document types for which preprocessing makes sense, such as C/C++.

Source Checking

When checked, CLint++ permits you to Source Check documents of this type from Lint | Analyze. You should only check this for document types for which source checking makes sense, such as C/C++.

Function Search

When checked, CLint++ permits you to search for function or variable definitions in documents of this type. You should only check this for document types for which C style declarations make sense, such as C/C++.

New File Template

This edit field contains the template for files of a given type created by File | New. The template should contain the string '**`%d`**' if it occurs in the extension part, for example:

```
Untitled-%d
```

or '**`%02d`**' if it occurs in the main part, for example:

```
noname%02d.c
```

The string is replaced by a unique number which is the number of New documents you have created since CLint++ was started.

New

This button pops up the New File Type dialog, which allows you to create new document types.

Filters

This button pops up the Edit File Filters dialog, which allows you to edit or add to the list of file extensions taken to be the extensions of documents of this type.

Characters

This button pops up the Character Class dialog, which allows you to modify the characters used in displaying documents of this type.

Colors

This button pops up the Edit Display Colors dialog, which allows you to assign and modify the display colors used by documents of this type.

Keywords

This button pops up the Edit Syntax Keywords dialog, which allows you edit the list of keywords used for syntax coloring documents of this type.

DOS

This button is only active when **Enable DOS** is checked. It pops up the DOS Command Setup dialog, which allows you to specify a DOS command line for a particular document type such as a Makefile.

Edit Syntax Characters dialog

This dialog is popped up by the Edit File Properties | Characters button.

It allows you to specify how comments are handled for the current document type, what character is used for preprocessor support (if any), and any additional characters used in identifiers.

Comment Style

None
When active, no comments are recognized, unless one of the keywords has the Comment attribute. For example, the default Batchfile implementation uses this, and has the ***Comment*** attribute set on the keyword **REM**.

C
When active, C comments `/*---*/` are recognized and colored in the Multiline Comment color.

C++
When active C comments `/*---*/` are recognized and colored in the Multiline Comment color, and C++ comments (`//`) are recognized and colored in the Single Comment color.

Pascal
When active, Pascal comments `{---}` are recognized, and colored in the Multiline Comment color.

Basic
When active, Basic comments (`'`) are colored in the Single Comment color.

Assembler
When active, Assembler comment (`;`) are colored in the Single Comment color.

Make
When active, Make comments (`#`) are colored in the Single Comment color.

Preprocessor Support

When checked, you can specify a character which is used for preprocessor support. When C or C++ are active, '`#`' is automatically filled in here. When Make is active '`!`' is filled in here.

More Identifier characters ($,@,%,#)

You can enter any or all of the characters '`$`', '`@`', '`%`', or '`#`' which will then be recognized as being legitimate characters in identifiers. You can paste any characters in however to get unusual characters. When Basic is active, this is prefilled with '`$`'. When Assembler is active, this is prefilled with '`$@`'.

DOS Command Setup dialog

This dialog is popped up by the Edit File Properties | DOS button.

It allows you to specify a DOS command line for the current document type. You can pass the document filename, full path, or directory into the command line. You run the DOS command using DOS Command from the right mouse button in the Edit Window, or from the Lint | DOS Command menu entry.

DOS Command

This edit box allows you to enter a command line. You can use one of these macros in the command line:

`%f`	replaced by the document filename
`%F`	replaced by the full pathname of the document
`%D`	replaced by the directory part of the document filename
`%b`	replaced by the basename of the file - that is the name with no extension.
`%e`	replaced by the extension of the file with no name or period.

Wait for completion

When active, the chosen command will run to completion, which allows CLint++ to collect any output into the message window.

Run in background

When active, the chosen command will run without CLint++'s further involvement. If you use this, be aware that CLint++ can't capture the output from the command.

Run minimized

When checked, the command will execute as an icon.

Prompt for parameters

When checked, you are prompted for additional command arguments with the DOS Command Argument dialog when you try to run a DOS command. This allows you to bypass entering a command line in **DOS Command** above, since you'll be prompted for one anyway.

Collect output

This checkbox is only available when **Wait for completion** is active. When checked, the command output is collected, and presented in the Message Window when the command terminates. Be aware that at most 48k of output text may be collected this way. See DOS Command for more detail.

New File Type dialog

This dialog is popped up by the Edit File Properties | New button.

It allows you to create new document types, and prefills the fields in the dialog with proposed values for you to change.

Description

This is the descriptive text which is used to identify documents of this type. It may not be the same as the descriptive text for any existing document type. Once created, you can edit this descriptive text from the Edit File Properties dialog.

Template

This is the descriptive text which is used to identify documents of this type. It may not be the same as the template for any existing document type. Once created, you can edit this template from the Edit File Properties dialog.

Initial Colors

This drop down list offers you the name of all existing document types. Choose the one whose color scheme most closely matches your needs. You can then specify the colors in detail with the Edit Display Colors dialog after creating the new document type.

Extension

Enter a master file extension here. It must differ from all other extensions used by all other document types. If the new document type is to respond to more than one extension, you'll have to use the Edit File Properties dialog's ***Filters*** button to add more after creating the new document type.

Filter

Enter descriptive filter text as you want it to appear in the File | Open ***List Files of Type*** drop down list.

Syntax coloring

Check this box if the new document type is to support syntax coloring.

Edit File Filters dialog

This dialog is popped up by the Edit File Properties ***Filters*** button.

It allows you to add or change the filter text displayed by the File | Open ***List Files of Type*** drop down list, and to add to or change the list of file extensions for each filter.

Filters:

This list box shows the filter strings currently in use for documents of this type. You can add filters with **New**, or **Edit** existing filters, or **Delete** a filter.

Extensions:

This list box extensions currently associated with the filter selected in **Filters**. You can add extensions with **New**, or **Edit** existing extensions, or **Delete** an extension.

Edit Display Colors dialog

This dialog is popped up by the Edit File Properties | Colors button.

It allows you to specify the foreground and background colors for each of the syntax elements used by syntax coloring.

Syntax Element

This list box shows all the syntax element names in the colors currently selected, with a short black highlight marker. As you change the selection, **Foreground** and **Background** will change to show you which colors are in use by each.

Foreground

This list box shows all the 16 Windows solid color names. As you change the selection, **Syntax Element** will change to show you the effect of your choice.

Background

This list box shows all the 16 Windows solid color names. As you change the selection, **Syntax Element** will change to show you the effect of your choice.

The available syntax elements you can specify are:

Plain Text	Everything not one of the following
Marked Block	A block selected by dragging or Shift-movement key combinations
Error Lines	Lines for which CLint++ detects an error in analysis.
Single Comment	Single comments introduced by keywords marked with the **Comment** attribute in Edit Syntax Keywords, or as specified in the Edit Syntax Characters dialog.
Multiline Comment	Multiline comments as specified in the Edit Syntax Characters dialog.
Keywords	Keywords as defined in the Edit Syntax Keywords dialog.
Punctuation	All normal punctuation "`:;{}[]()*&^%$#@!,.=+-~<>`" unless otherwise specified in the Edit Syntax Characters dialog.
Identifiers	Sequences of letters or digits not started by a digit which are not keywords.
String Literals	Strings quoted with double quotes (`"`).
Integer Constants	Numbers not one of the following.
Floating Constants	Numbers including a decimal point or exponent.
Octal Constants	Numbers with a leading zero.
Hex Constants	Numbers with a leading **`0x`** or **`0X`**
Character Literals	Strings quoted with single quotes (`'`).

Preprocessor Lines	Lines started by the preprocessor character specified in the Edit Syntax Characters dialog.
Illegal Text	Characters (like `@` for C or C++) which are illegal for a given document type.
Line Numbers	Line numbers when visible.
Original Line Numbers	Original line numbers when visible.

Edit Syntax Keywords dialog

This dialog is popped up by the Edit File Properties | Keywords button.

It allows you to add to or edit the keywords used for syntax coloring.

Ignore Case

When checked, the keywords are not case sensitive.

Keywords

This list box shows all the keywords in alphabetic order. You can add **New** keywords, and **Edit** or **Delete** existing ones.

Edit Keyword dialog

This dialog is popped up by the Edit Syntax Keywords | New or Edit buttons.

It allows you to specify or change the keywords used for syntax coloring, to specify if the keyword is a comment marker (like Batchfile or Basic **REM**), and possibly to define or modify a macro to be executed when you type the keyword.

Keyword:

This edit field allows you to enter or edit the keyword.

Keyword macro

When checked, the keyword has an associated **Definition** which is expanded when you type the keyword.

Type

You may choose Keyword or **Comment** from this list. If **Comment** is selected, then the comment keyword and the rest of the line will be colored in the **Single Comment** color as specified in the Edit Display Colors dialog.

Definition

Enter here a macro definition. The definition may consist of any typeable characters, and any of a number of special commands announced by a single '\' backslash character. The commands are:

\	a single backslash character	
\n	Enter	
\t	Tab	
\b	Backspace	
\d	Delete	
\U	Up Arrow	
\D	Down Arrow	
\L	Left Arrow	
\R	Right Arrow	
\E	End	
\H	Home	
\u	Page Up	
\P	Page Down	
\CU	Scroll up:	Ctrl-Up Arrow
\CD	Scroll down:	Ctrl-Down Arrow
\CL	Word Left:	Ctrl-Left Arrow
\CR	Word Right:	Ctrl-Right Arrow
\CE	Goto End:	Ctrl-End
\CH	Goto Start:	Ctrl-Home
\Cu	Goto Wintop:	Ctrl-Page Up
\CP	Goto Winbot:	Ctrl-Page Down
\SU	Select Up	Shift-Up Arrow
\SD	Select Down:	Shift-Down Arrow
\SL	Select Left:	Shift-Left Arrow
\SR	Select Right:	Shift-Right Arrow
\SE	Select to Line End:	Shift-End
\SH	Select to Line Start	Shift-Home
\Su	Select Page Up:	Shift-Page Up
\SP	Select Page Down:	Shift-Page Down
\CSU	Select Line Up:	Ctrl-Shift-Up Arrow
\CSD	Select Line Down:	Ctrl-Shift-Down Arrow

\CSL	Select Word Left:	Ctrl-Shift-Left Arrow
\CSR	Select Word Right:	Ctrl-Shift-Right Arrow
\CSE	Select to End:	Ctrl-Shift-End
\CSH	Select to Start:	Ctrl-Shift-Home
\CSu	Select to Wintop:	Ctrl-Shift-Page Up
\CSP	Select to Winbot:	Ctrl-Shift-Page Down

Key Assignments dialog

This dialog is popped up by the Options | Key Assignments menu entry.

It supports assigning any Alt/Ctrl/Shift based key combination to any edit or checking action, and all menu actions.

Function

This list box shows all of CLint++'s key assignable operations. As you scroll through this list, you can see what if any keys are currently assigned to that function. To assign a key to a function, scroll to the function you want to assign, select the key you want from the **Key** list, and click **Assign**. If the key is already assigned to a different function, you'll be asked if you want to change the assignment.

Key

This list box shows all the key presses CLint++ will allow you to assign to functions. As you scroll through this list, you can see what functions if any they are assigned to.

Current Keys

This list box shows which keys if any are assigned to the currently selected function.

Currently mapped to

This shows the name of the function if any which the currently selected key is mapped to.

Assign

Clicking on this button assigns the current key to the current function if not already assigned. If the key is assigned to a different function, you'll be asked if you want to delete the previous assignment.

Unassign

Clicking on this button deletes the currently selected key assignment in **Current Keys**.

Default

Clicking on this button restores all key assignments as they were when CLint++ was shipped. This is useful when you have lost some or all of the default key assignments.

Lint Options dialog

This dialog box may be entered from Options | Lint Options, or from Lint | Edit Lint File. In the first case, you are controlling CLint++'s global default setup in CLINT.CFG which is used for all checking unless you specifically override an option locally in a Lint File. In the second case, you are controlling Lint File specific options.

CLint++ stores the global default options in the file CLINT.CFG which is stored in the same directory as CLINTW.EXE. This file is also used by the command line version of CLint++ to control its operation in the same way as the Windows version. The file is commented ASCII text, and you can drag it into CLint++ for inspection or modification from File Manager or Explorer.

Program specific options are stored in Lint Files which have a CLN extension. You can view or modify these files as ASCII text by right clicking Lint Window | Edit as Text.

Local override

This check box is only present when invoked from Lint | Edit Lint File. When checked, it makes CLint++ copy the complete CLINT.CFG options into the Lint Files. Subsequent uses of the lint file for checking will get all configuration options from the Lint File only and ignore CLINT.CFG. The default is to inherit options from CLINT.CFG. This is particularly useful if a file uses a different compiler from the CLINT.CFG default.

Compiler

CLint++ offers direct support for many popular compiler products. Simply choose your compiler from this list, and CLint++ will automatically fill in the relevant fields in the dialog. If you have installed the compiler elsewhere than the default CLint++ expects, you will need to edit the Header directory entry to tell it where your compiler's INCLUDE directory is.

Header directories:

CLint++ pre-fills this list box with the expected name of your compiler's INCLUDE directory when entered from Options | Lint Options. You can add additional header directories to the list with **New**, edit existing entries with **Edit**, or delete entries with **Delete**. The command line program equivalent is **`-I`**.

Checking libraries

CLint++ pre-fills this list box with the checking library for your compiler when entered from Options | Lint Options. You can add additional libraries to the list with **New**, edit existing entries with **Edit**, or delete entries with **Delete**. These libraries contain definitions for the vendor run-time libraries, or you can create checking libraries for your own code. The command line program equivalent is **`-l`**.

Relative Library Paths

This check box is only present when invoked from Lint | Edit Lint File. When checked, CLint++ stores all library path names in relative rather than absolute form. This makes it easy to move an entire project elsewhere on your disk.

Project is a Library

This check box is only present when invoked from Lint | Edit Lint File. When checked, CLint++ knows that your program is incomplete, and so suppresses checks for symbols defined but not used, or used but not defined. The command line program equivalent is **`-u`**.

Preserve *.~TP files

This check box is only present when invoked from Lint | Edit Lint File. When checked, CLint++ does not delete the CLINT.~TP temporary file containing the result of analysis. You can then rename the file with a suitable CLB extension, and use it as a library in subsequent checking for other modules.

Character constant length 1..4:

You can specify the maximum length of character constants CLint++ will allow without complaint here. The default is 2. The command line program equivalent is `-KcN` where N is 1-4.

C only

This specifies that all source files in the project are C only, and C++ is not supported. The command line program equivalent is `-KC1`.

C comments nest

When checked, CLint++ allows nested comments of the form '/* ... /* ... */ ... */' with an unlimited nesting level. Be aware that this is not allowed by the **ANSI** Standard. The default is not checked. The command line program equivalent is `-C`.

Microsoft 'const' and 'volatile'

When checked, CLint++ supports the Microsoft usage for **const** and **volatile**. CLint++ checks this box for those compilers which require this extension. The command line program equivalent is `-Kb1`.

Allow Sizeof or cast in #if/#elif.

When checked, CLint++ allows **sizeof()** and casts in **#if** and **#elif** preprocessor directives. Be aware that this is not allowed by the **ANSI** standard. However, it is required by the Archimedes 68HC11 compiler, and if you choose that compiler, CLint++ checks this box for you. The command line program equivalent is `-G`.

32 bit integers and fields.

When checked, CLint++ knows that ints are 32 bit, and therefore supports bit fields in structures longer than 16 bits. This is checked for you by CLint++ if you choose the Intel iC86 compiler, or Watcom 386 compiler. Many other compilers will also generate 32 bit code, and if you have a 32 bit program, check this box. The command line program equivalent is `-3`.

x86 register names.

When checked, CLint++ supports the use of x86 pseudo-register names _AX etc. Be aware that this is not allowed by the **ANSI** Standard. CLint++ checks this box for those compilers which support this extension. The command line program equivalent is `-H`.

'extern register' storage class.

When checked, CLint++ supports a new storage class '**extern register**'. Be aware that this is not allowed by the **ANSI** Standard. This is required by some compilers for embedded controllers. When you choose the Archimedes 68HC11 compiler, CLint++ checks this box for you. The command line program equivalent is `-Y`.

Support 'asm'

When checked, CLint++ supports **asm** in all its forms. CLint++ checks this box for those compilers which support this extension. The command line program equivalent is `-Ks1`.

Support 'long long'

When checked, CLint++ supports the **long long** data type. CLint++ checks this box for those compilers which support this extension. The command line program equivalent is `-KL1`.

Defines

This pops up a dialog which allows you to specify preprocessor symbols which must be defined prior to checking. CLint++ prefills this according to the compiler you choose.

Style

This pops up a dialog where you can specify many style options.

Type

This pops up a dialog where you can specify many type options.

General

This pops up a dialog where you can specify many general options.

C++

This pops up a dialog where you can specify C++ options and warnings.

Identifier length 0,32..255:

You can specify specific identifier lengths from 32 to 255 characters, or if you specify 0 remove any limit on identifier length. The default is 32. The command line program equivalent is `-KiNNN` where NNN is 0, or 32-255.

Maximum error count 10..255:

You can specify the maximum number of errors CLint++ will allow in any individual file while checking. If this limit is exceeded, CLint++ stops checking. The default is 15. The command line program equivalent is `-eNNN` where NNN is 10-255.

Maximum warning count 10..255:

You can specify the maximum number of warnings CLint++ will allow in any individual file while checking. If this limit is exceeded, CLint++ stops checking. The default is 60. The command line program equivalent is `-wNNN` where NNN is 10-255.

Save

This button is used to copy your settings to the configuration file CLINT.CFG, or the Lint File. If the file is read-only this button can't be selected. CLint++   requires that the numeric values are in the allowed ranges, and warns if you are specifying a header directory which does not exist or does not contain headers files; and warns if any library you specify does not exist.

C++ Options dialog

This dialog is popped up by the ***C++*** button in Options | Lint Options.

It allows you to specify C++ specific options.

warn of reference temporaries

Off by default. When checked, you will be warned of any reference temporaries created by your program. The command line program equivalent is **`-wrtm`**.

base initialization without class is obsolete

On by default. When checked, you will be warned of initialization of a base class without mentioning the base class name. The command line program equivalent is **`-wbnc`**.

warn of hidden virtual functions

On by default. When checked, you will be warned of a member function declaration which hides a virtual function of the same name inherited from a base class. The command line program equivalent is **`-whvf`**.

prefix operator used as postfix

On by default. When checked, you will be warned of the use of prefix ++ or -- as postfix forms. The command line program equivalent is **`-wpre`**.

array size for 'delete' ignored

On by default. When checked, you will be warned of mentioning an explicit array size in **delete[]**. The command line program equivalent is **`-wasd`**.

function may throw a type, but no throw specifier

On by default. When checked, you will be warned of calls of functions which throw a type when n the calling function has no **throw** specifier. The command line program equivalent is **`-wtns`**.

function may throw a type not in throw specifier

On by default. When checked, you will be warned of calls of functions which throw a type other than that declared in the calling function. The command line program equivalent is **`-wtni`**.

Support ANSI/ISO keywords 'and', 'or', etc.

Off by default. When checked, the ANSI/ISO keyword extensions are enabled. The command line program equivalent is **`-Ak`**.

Warn of ANSI/ISO keywords

Off by default. It can only be selected when ***Support ANSI/ISO keywords 'and', 'or', etc.*** is checked. When checked, if any of the ANSI/ISO keywords (**and**, **and_eq**, **bitand**, **bitor**, **compl**, **not**, **or**, **or_eq**, **xor**, **xor_eq**, **not_eq**) occur in the source, you will be warned of them. The command line program equivalent is **-wans**.

Support ANSI/ISO 'bool' type

Off by default. When checked, the ANSI/ISO **bool** type and the constants **true** and **false** are enabled. The command line program equivalent is `-Ab`.

Support ANSI/ISO 'namespace' and 'using'

Off by default. When checked, the ANSI/ISO **namespace** and **using** keywords are enabled. The command line program equivalent is `-An`.

Deprecate the use of 'static'

Off by default. It can only be selected when ***Support ANSI/ISO 'namespace' and 'using'*** is checked. When checked, you are warned when global **static** declarations are used. The command line program equivalent is **-At**.

Deprecate the use of access declarations

Off by default. It can only be selected when ***Support ANSI/ISO 'namespace' and 'using'*** is checked. When checked, you are warned when access declarations are used. The command line program equivalent is **-Ad**.

Support ANSI/ISO #include

Off by default. When checked, the ANSI/ISO header style (where extensions are not required) is supported. The command line program equivalent is **-Ah**.

Support ANSI/ISO if, while etc.

Off by default. When checked, the ANSI/ISO declarations of local variables in if, while, and switch is supported. The command line program equivalent is **-Ai**.

Edit Header dialog

This dialog is popped up by the ***Header directories* | *New*** or ***Header Directories* | *Edit*** buttons in Options **|** Lint Options.

It allows you to navigate to or enter a header directory name.

Relative Path

When checked, the name you select will be converted to a relative path when you select OK, if on the same drive as the file the Options **|** Lint Options parent dialog is working with.

Header

This edit field allows you to directly type a header directory name. It also shows the current directory resulting from navigating around the disk using **Directories** and **Drive**.

Directories

This list box contains a list of directories in the current directory. Clicking on any entry causes that to become the current directory, and updates the **Header** edit field accordingly.

Drive

This combo box contains a list of on your machine. Selecting any entry causes that to become the current drive, and updates the **Header** edit field and **Directories** list box accordingly.

Defines dialog

This dialog is popped up by the **Defines** button in Options | Lint Options.

It allows you to specify preprocessor symbols which must be defined prior to checking. CLint++ prefills this according to the compiler you choose.

Defines

CLint++ pre-fills this list box with some symbols according to the compiler you specified in Options | Lint Options. You can add additional definitions to the list with **New**, edit existing definitions with **Edit**, or delete definitions with **Delete**. The command line program equivalent is `-D`. The syntax is:

symbol	defines the symbol with the value 1
or	
symbol =	defines the symbol with empty replacement text
or	
symbol=replacement	defines the symbol with the given replacement text

You may concatenate definitions on the same line using semicolons (;) as separators.

Undefines

This specifies the names of preexisting symbols to remove. It is not an error to specify a name which is not defined. You can add a symbol to the list with **New**, edit existing symbols with **Edit**, or delete symbols with **Delete**. The command line program equivalent is `-U`. The syntax is:

symbol	remove the definition of symbol

Style Warnings dialog

This dialog is popped up by the Style button in Options | Lint Options.

'{' advised after 'for'

When checked, you will be warned of any **for** loop whose body is not braced, unless it is the empty statement (;). The default is not checked. The command line program equivalent is **`-wfbr`**.

'{' advised after 'else'

When checked, you will be warned of any **else** whose body is not braced, unless it is the empty statement (;). The default is not checked. The command line program equivalent is **`-webr`**.

'{' advised after 'while'

When checked, you will be warned of any **while** loop whose body is not braced, unless it is the empty statement (;). The default is not checked. The command line program equivalent is **`-wwbr`**.

'{' advised after 'if'

When checked, you will be warned of any **if** whose body is not braced, unless it is the empty statement (;). The default is not checked. The command line program equivalent is **`-wibr`**.

' ' advised after operator

When checked, you will be warned of any C operator not followed by a space or newline. The default is not checked. The command line program equivalent is **`-wspc`**.

Empty body of loop or condition

When checked, you will be warned of any **if**, **else**, **for**, **while**, or **do** whose body is empty, unless it is the empty statement (;). The default is checked. The command line program equivalent is **`-welb`**.

Legal empty body is {} not ;

When checked, you will be warned of any **if**, **else**, **for**, **while**, or **do** whose body is empty, unless it is the empty statement {}. The default is not checked. The command line program equivalent is **`-webb`**.

Use of 'register'

When checked, you will be warned of any use of the **register** keyword. The default is checked. The command line program equivalent is **`-wreg`**.

Suggest symbols which should be static

When checked, you will be informed that any symbols in your program only used in only one file could be declared **static**. The default is not checked. The command line program equivalent is **`-wsgs`**.

Note static names matching globals

When checked, you will be warned of informed of any **static** symbols in your program which have the same name as a global symbol. The default is not checked. The command line program equivalent is **`-wscg`**.

Report symbols defined but never used

When checked, you will be informed of any symbols you defined (allocated storage to) which are not used. The default is checked. The command line program equivalent is **`-wnus`**.

Report redefined library symbols

When checked, you will be informed of any symbols you defined which have the same name as a runtime library symbol. The default is checked. The command line program equivalent is **`-wlib`**
.

Report possibly unintended assignment

When checked, CLint++ checks all assignments occurring in a conditional context. If the assignment expression is not explicitly compared with a value, you will get warned. The default is checked. The command line program equivalent is **`-wpua`**.

Report nested comments

When checked, you will be informed of any C comments which are nested like '/* ... /* ... */ ... */'. This check is disabled if you allowed nested comments in Options | Lint Options. The default is checked. The command line program equivalent is **`-wcom`**.

Report '//' comments in C

When checked, you will be informed of any C++ comments in C files. The default is unchecked. The command line program equivalent is **`-wcpc`**.

Report ANSI trigraphs

When checked, you will be informed of any ANSI trigraphs outside comments in your program. The default is checked. The command line program equivalent is **`-wtri`**.

Function body larger than N lines

When checked, you will be informed of any function bodies longer than the limit you enter. The default is not checked. The command line program equivalent is **`-wlfbNNN`**.

Type Warnings dialog

This dialog is popped up by the Warnings button in Options | Lint Options.

'enum' was typedefed (C)

When checked, CLint++ will warn of any typedef of an enumerated type. The default is not checked. The command line program equivalent is **`-wtye`**. (C only).

'struct' was typedefed in lowercase (C)

When checked, CLint++ will warn of any typedef of structure in lower case. The default is not checked. The command line program equivalent is **`-wtys`**. (C only).

'struct' typedefs should be upper case (C)

This is only enabled if ***'struct' was typedefed in lowercase*** is checked. When checked, CLint++ will warn of any typedef of structure not fully capitalized. The default is not checked. The command line program equivalent is **`-wtysU`**. (C only).

Primitive type was typedefed (C)

When checked, CLint++ will warn of any typedef of primitive type like **int** or **char**. The default is not checked. The command line program equivalent is **`-wtyb`**. (C only).

Pointer type was typedefed (C)

When checked, CLint++ will warn of any typedef of any pointer types. The default is not checked. The command line program equivalent is **`-wtyp`**. (C only).

Complex type was typedefed (C)

When checked, CLint++ will warn of any complex typedefs. The default is not checked. The command line program equivalent is **`-wtyg`**. (C only).

Mixing pointers to different char types

When checked, CLint++ will warn of assignments or comparisons between differing char types. The default is checked. The command line program equivalent is **`-wmxp`**.

Constant is long (requires 'L' suffix)

When checked, CLint++ will warn of any constants larger than 16 bits which don't have an L suffix. The default is checked. The command line program equivalent is **`-wcil`**.

Report enumeration misuse (C)

When checked, CLint++ will warn of comparisons or assignments of enumerated constants with other types. The default is checked. The command line program equivalent is **`-wenu`**.

warn of untyped objects implicitly 'int'

When checked, CLint++ will warn of the declaration of objects without a specific type For C++, this is now deprecated. The command line program equivalent is **`-wenu`**.

allow any integral type in bitfields (C)

When checked, CLint++ will allow any integral type for a bitfield. The command line program equivalent is **`-waib`**.

allow enumerated types in bitfields (C)

When checked, CLint++ will allow enumerated types for bitfields. The command line program equivalent is **`-weib`**.

warn of return value unused

When checked, CLint++ will warn of calls of functions which return a value which is not used. The default is unchecked. The command line program equivalent is **`-wrvi`**.

Don't warn about

You can enter exceptions to the rules you've enabled in this box with **New**, or **Edit** or **Delete** such names. Any name occurring in this box will not provoke any complaint no matter how used. The command line program equivalent is **`-Nname`**.

General Warnings dialog

This dialog is popped up by the   Warnings button in Options | Lint Options.

Automatic array/struct initialized

When checked, CLint++ warns of automatic struct or array objects which are initialized. The default is checked. The command line program equivalent is **`-wiag`**.

Multiply included files

When checked, CLint++ warns of header files included directly or indirectly more than once. The default is checked. The command line program equivalent is **`-winc`**.

Unreachable code

When checked, CLint++ warns of code which control can't reach. The default is checked. The command line program equivalent is **`-wunr`**.

Don't warn of unreachable 'break'

This is only enabled if ***Unreachable code*** is checked. When checked, CLint++ suppresses warnings of unreachable **break** statements. The default is checked. The command line program equivalent is **`-wnub`**.

Structure passed by value

When checked, CLint++ warns of passing or returning structure values. This warning is suppressed for C++ inline functions. The default is checked. The command line program equivalent is **`-wstv`**.

Constant condition/comparison

When checked, CLint++ warns of conditional expressions whose value is determinable at compile time. The default is checked. The command line program equivalent is **`-wcon`**.

Degenerate unsigned comparison

When checked, CLint++ warns of conditional expressions which are degenerate because one or both of the operands are unsigned. The default is checked. The command line program equivalent is **`-wduc`**.

Suppress warnings in headers

When checked, CLint++ disables warnings in your header files. The default is not checked. The command line program equivalent is **`-whdr`**.

Suppress warnings in system headers

When checked, CLint++ disables warnings in system (compiler supplied) header files. The default is checked. The command line program equivalent is **`-wshd`**.

Check printf-like function calls

When checked, CLint++ checks all **printf()**, **fprintf()**, **sprintf()**, **scanf()**, **fscanf()**, and **sscanf()** function calls for correct use of the format specifiers. You can specify checking of additional functions you write with this functionality in the ***'printf' like functions*** list box. The default is checked. The command line program equivalent is **`-wprn`**.

Non ANSI printf size specifiers

This is only enabled if ***Check printf-like function calls*** is checked. When checked, CLint++ warns of the non-ANSI printf size specifiers **N** and **F**. The default is not checked. The command line program equivalent is **`-wnac`**.

Report loss of significance

When checked, CLint++ warns of assignments from **long** to shorter types. The default is checked. The command line program equivalent is **`-wsig`**.

Report strict loss of significance

This is only enabled if ***Report loss of significance*** is checked. When checked, CLint++ warns of any assignment from a type to a shorter type. The default is not checked. The command line program equivalent is **`-wsig+`**.

Function should be prototyped

When checked, CLint++ warns of any call of a function or pointer to function which is not prototyped. The default is checked. In **C++**, this is always active. The command line program equivalent is **`-wpro`**.

Return and return of a value

When checked, CLint++ considers it an error to return directly and also return a value from a function, or to fail to return a value when a return value is declared, or to return a value when the function is declared **void**. The default is checked. The command line program equivalent is **`-wret`**.

Long (>16) field declared

This is only enabled if Options | Lint Options ***Support 32 bit integers and fields*** is not checked. When checked, CLint++ warns of fields declared longer than 16 bits. The default is checked. The command line program equivalent is **`-wlnf`**.

Non ANSI initializer bracing

When checked, CLint++ warns of structure and array initialization which is not braced according to the ANSI guidelines. The default is checked. The command line program equivalent is **`-wbbr`**.

'printf' like functions

You can enter the names of functions which require format checking like **printf()** in this box with **New**, or **Edit** existing function names, or **Delete** function names. The command line program equivalent is **`-FNname`** where *N* is 1 or more.

'exit' like functions

You can enter the names of functions which don't return like **exit()** in this box with **New**, or **Edit** existing function names, or **Delete** function names. CLint++ already knows that **exit()**, **abort()**, **_exit()**, and **longjmp()** don't return. The command line program equivalent is **`-F0name`**.

Version Control dialog

This dialog is popped up by the Options | Version Control menu entry.

This dialog allows you to configure CLint++ to work with any version-control tool with a DOS command line interface. You must have such a tool in order to use CLint++'s integrated version management capabilities. CLint++ supports the popular products PVCS from Intersolv, and RCS from Mortice Kern Systems directly, and other comparable products with some small setup effort. Also, there is a freeware port by **R&D Associates** of ***GNU RCS*** available for use directly with CLint. (See CompuServe IBMPRO Library #3).

CLint++ assumes a standard configuration where the version control archives are stored in a subdirectory immediately below the directory containing the managed files. Some sites store the archives elsewhere - perhaps on a separate network drive. See **Use path file** below to support this.

You configure CLint++ to interface with your version control tool by entering your Administrator established version-control User Name (typically the same as your network login name if you use a network), and then if you use PVCS or RCS, just click on the appropriate button - CLint++ fills in the rest of the dialog for you. Then choose **OK**, and you're done.

If you use one of the other fine version control products, you will need to set CLint++ up to execute on your behalf the commands you would type at the DOS command line. There is one directory name and five commands to be entered. CLint++ requires that all the commands named can be found in your ***PATH***.

Before you can enter anything, you must enable Version Control with by clicking the **Use Version Control** checkbox.

Version Control User Name

This is the name by which your version control tool knows you, and is usually set by your Version Control Administrator.

Archive Subdirectory:

In here you enter the subdirectory name your tool expects. For PVCS the default is **VCS,** for RCS this is **RCS**.

Get and Lock for editing:

Enter the command to check a writable file out of an archive and lock the archive excluding the filename - CLint++ appends that name to the command string you enter. You must ensure that you use any 'force' switch the tool offers to ensure that it won't prompt for input.

Unlock without change:

Enter the command you would use to unlock a locked archive, and restore the workfile to its original state. This should leave the workfile read-only. This may also require a 'force' option.

Place under Version Control:

Enter the name of a batchfile to create a new archive and check in a file for the first time. Model your batchfile on **PINIT.BAT** or **RINIT.BAT** which we supply for PVCS and RCS.

Retrieve archive info:

Enter the name of a batchfile to extract the archive information from the archive into a file. Model your batchfile on **PINFO.BAT** or **RINFO.BAT** which we supply for PVCS and RCS. CLint++ expects to get an information file containing a line with the strings REV or HEAD to specify the current revision, and a line containing LOCK which identifies the person who currently has a lock on the file. CLint++ only pays attention to the first such lines found. These pattern searches are not case sensitive.

Use Version Control:

Check this box to enable CLint++ support for version control. This box is checked automatically if you choose either of the default buttons.

Use path file

When checked, CLint++ reads the file named in **Path control file** in the same directory as the document file, looking for a line starting with the pattern specified in **Path pattern**. If found, the version control archive is taken to be in the directory specified following the pattern.

Path control file:

This edit field is only enabled when **Use path file** is checked. Enter the name of a file CLint++ will read to find the directory containing version control archive files, for example ***VCSPATH.CFG***. When searching for an archive, CLint++ first looks under the current directory, and if no archive is found, looks in the file given here for the directory which should contain the archive. When creating new archives, the directory given here is only used if there is no archive directory beneath the file. Directory names may be relative, and should not include the trailing **Archive Subdirectory** component.

Path pattern:

This edit field is only enabled when **Use path file** is checked. Enter a pattern string containing no white space which CLint++ will search for in column 1 only of the file specified in **Path control file**. The pattern may have as its last character an **'='** with no preceding white space. For example, given the pattern string ***VCSPATH=***, CLint++ will match the strings:

```
VCSPATH = directory
VCSPATH= directory
VCSPATH =directory
```

In addition, CLint++ will permit the directory name to be prefixed by a Makefile style macro call ***$ (STRING)***. If present, such a macro string is deleted from the directory name before use. This is useful if the definition is actually in a Makefile.

Use PVCS Extension Rule .??V

The Intersolv PVCS tool changes archive names to have an extension where the third character is V and prior characters not present in the workfile name become underscores. This box is checked automatically if you choose **Default PVCS**. You should check it if you use the **R&D Associates** port of the ***GNU RCS*** package, which uses this rule also.

On Loading, Get and Lock files.

When checked, whenever you open a controlled file, CLint++ will establish the state of the file, and ask if you want to lock the file for editing. If you answer Yes, the file is locked for editing. This box is checked automatically if you choose either of the default buttons. You may find it convenient to not have this box checked, and instead right click the mouse in an Edit Window and lock the file from the Version Status menu entry Version Control Status dialog.

On Saving, Put changes and Unlock changed files.

When checked, whenever you save a changed file which is controlled, CLint++ examines the version status. If the file is not locked, you are asked if you want to lock the file and make the changes. Answering yes locks the file and then saves. If the file is locked, it is saved (written over the workfile), and CLint++ opens the Check In dialog. You may find it convenient to not have this box checked, and instead right click the mouse in an Edit Window and save the changes to the archive from the Version Status menu entry Version Control Status dialog.

On Saving, place New files under version control.

When checked, every time you save a new file (one which did not exist before), CLint++ will ask if you want to place the file under version control. If you answer Yes, CLint++ opens the Version Control Save dialog.

On Saving, place existing files under version control.

When checked, every time you save an existing file not under version control, CLint++ will ask if you want to place it under control. Answering Yes opens the Version Control Save dialog.

Minimize DOS boxes running version commands.

To avoid dependencies on the specifics of version control tools, CLint++ uses the lowest common denominator of the DOS command line to interface with the tool. When checked, CLint++ runs these DOS boxes minimized. Some tools may require interaction, and in these cases you should not check this, or CLint++ and the DOS box will appear to hang. This box is checked automatically if you choose either of the default buttons, as neither PVCS or RCS will prompt from these DOS boxes. Apparent hangs can be resolved by double clicking the minimized icon to restore the DOS box, and checking if it has prompted you for some reply.

Default PVCS

Choosing this button automatically fills in all edit fields and checks boxes as described above for direct support of **PVCS** versions 5.02 and higher.

Default RCS

Choosing this button automatically fills in all edit fields and checks boxes as described above for direct support of **MKS RCS** versions 5.2 and higher.

OK

Accept the changes you've made. You are required to fill in all six edit fields. If any of them is empty, or does not refer to a command in your PATH, you will be told of the problem. OK will only function when all fields have valid entries. However, CLint++ can't check that any vendor specific flags you enter are correct. For tools other than those directly supported, some experimentation and assistance from your System Administrator may be required.

Help

Opens the Windows Help on this page.

Cancel

Dismisses the dialog box without saving any changes.

Version Control Status dialog

This dialog allows you to view the version control status of an open file. You can place a file under version control, lock a file, and unlock a file - if you own the lock.

Revision list:

This list box shows the version status listing obtained from your version manager. You can scroll the list to view any part of this status.

Place under version control.

This button is only present if the file is not currently controlled. Choose it, and the file will be placed under version control. The Edit Window will be reloaded to show any changes the version manager makes.

Check in and unlock

This button is only present if the file is controlled. It is only active if the file is currently locked, and you own the lock. Choose it, and the Version Control Save dialog pops up to allow you to specify the comments. The Edit Window will be reloaded to show any changes the version manager makes.

Lock file for editing

This button is only active if the file is under version control and not currently locked. Choose it, and the file is locked, and becomes writable.

Remove current edit lock.

This button is only active if the file is under version control, locked, and you own the lock. Choose it, and the existing lock is removed.

Version Control Save dialog

When you save a file by any means such as File | Save, and **On Saving, place existing files under version control** is checked in the Version Control dialog, CLint++ checks to see if the file is under version control and locked, and that you own the lock. If so, it open this dialog offering to check the file in with its changes.

Enter Version Control Update Comments

This edit box allows you to enter up to 512 characters of commentary stating the reason for the change.

Update archive and remove lock

This button updates the version control archive for the file, removes the lock, and reloads the Edit Window to show any changes made to the file by the version manager.

Remove the current edit lock.

This button discards the current edit lock on the archive, thus restoring the disk file to its previous state. Your changes are lost, and only exist in the Edit Window memory.

File Menu

The File menu provides commands for creating new files, opening existing files, saving files, printing files, and exiting the application. In addition, up to 9 of the most recent files you've opened are visible beneath the Exit entry. Selecting any of them immediately places you in a window containing that file - opening a new window if required.

New	Create a new, untitled, document.
Open	Open an existing file.
Close	Close the current document.
Save	Save the current document if its contents have changed.
Save As	Save the current document under a new name.
Save All	Save all modified documents.
Insert	Insert a file into the current document.
Revert	Restore the document to an earlier state.
Save Selection As	Save the selected text in a new file.
Page Setup	Set printing characteristics.
Print	Print the current document.
Compare	Compare files for differences.
Exit	Exit CLint++ application.

File New command

The File | New command opens the New File Type dialog box. You choose the type of document you want to open.

Document Types

This list box holds the descriptive names of all the document types. Double click on one, or select one and choose **OK** to open a window of that type.

OK

Opens a new window of the selected type.

File Open command

File | Open displays the Open dialog box so you can select a file to load into a new window. You can also create a new file by naming a one that doesn't currently exist.

File Name:

This edit field contains the filters for the current **List Files of Type** selection. You can type a filename or wildcard directly here. Below it is a list box containing all files matching the pattern. Double click on one, or select one and choose **OK** to open it.

List Files of Type

This drop down list offers a selection of filters corresponding to the document types currently in existence. Select a filter to display files of that type.

Directories:

You can navigate around the file tree by double clicking on the visible directory names. As you do, the file list is updated to show you the files of the current type in that directory.

Drives:

This drop down list presents all the drives on your system. If you select a different drive letter, the other fields will change to show your selection.

OK

Choose this to open the selected file.

File Close command

File | Close closes the currently active window. If the contents have been modified, you are asked if you want to save the changes.

File Save command

File | Save command saves the document in the active window to disk. If the document is unnamed, the Save File As dialog box is displayed so you can name the file, and choose where it is to be saved. You are notified if you attempt to write to a read-only file.

File Save As command

File | Save As allows you to save a document under an new name, or in a new location on disk. The command displays the Save As dialog box. You can enter the new file name, including the drive and directory. All windows showing this file are updated with the new name. If you choose an existing file name, you are asked if you want to overwrite the existing file. If the new file name has a different extension than the old name, the document type is changed (and therefore also the syntax coloring scheme) to reflect this.

File Name:

This edit field contains the proposed name of the file to save. You can type a filename or wildcard directly here. Below it is a list box containing all files matching the pattern. Double click on one, or select one and choose **OK** to save to it.

Save File as Type

This drop down list offers a selection of filters corresponding to the document types currently in existence. Select a filter to display files of that type.

Directories:

You can navigate around the file tree by double clicking on the visible directory names. As you do, the file list is updated to show you the files of the current type in that directory.

Drives:

This drop down list presents all the drives on your system. If you select a different drive letter, the other fields will change to show your selection.

OK

Choose this to save the selected file.

File Save All command

File | Save All saves all currently modified windows to their original files. If any window has not been named, CLint++ opens a File | Save As dialog to prompt you where to save the file.

File Insert command

File | Insert pops up an Open dialog for you to specify a file. On choosing OK in the dialog, the contents of the file are inserted into the active window at the insertion point.

File Revert command

File | Revert restores the active window from the original disk file discarding all your changes. It is useful when a complex series of edits has gone wrong, and it would be too much work to undo.

File Save Selection As

File | Save Selection As opens the Save As dialog to save the currently selected text in a new file.

File Print command

File | Print prints the contents of the active window. Use File | Page Setup to set page layout options.

While printing is in progress, the Print dialog is active, and shows you the printing status. If you choose Cancel from this dialog, printing is aborted.

File Compare command

File | Compare opens the Compare Files dialog where you specify which files to compare.

File 1

Select one of the currently open documents from the list, or click on the box and then choose **Browse** to locate a file on the disk.

File 2

Select one of the currently open documents from the list, or click on the box and then choose **Browse** to locate a file on the disk.

Tile

Vertical	This button is selected by default. It causes the chosen file windows to be displayed side by side while being compared.
Horizontal	If you choose this button, the windows are displayed one above the other while being compared.

OK

This button is only enabled if the **File 1** and **File 2** selections are different. Choose it to start the comparison. If the files are different, the Compare dialog pops up with the windows tiled as you specified.

Browse

Choose this button to locate a file not currently loaded in an Edit Window to be compared. The chosen file is written by default to whichever of **File 1** or **File 2** was most recently selected. The default is **File 1**.

Compare dialog

The Compare dialog is popped up by choosing the File | Compare **OK** button, if the files being compared are different. The two windows show highlighted lines for the first lines found which differ.

Next Match

This button is only available if a difference is found between the files. The lines which differ are highlighted in each window. When chosen, the difference engine searches for the next match.

Next Difference

This button is only available when a match is found between the files. The lines which match are highlighted in each window. When chosen, the difference engine searches for the next difference.

Cancel

Closes the dialog and restores all windows as they were before File | Compare was chosen.

File Page Setup command

File | Page Setup displays the Page Setup dialog box which allows you to select and configure the printer to be used to print documents in the application. The controls operate as follows:

Margins

Left	Set the left print margin. The default is **0.5** inches.
Right	Set the right print margin. The default is **0.5** inches.
Top	Set the top print margin. The default is **0.5** inches.
Bottom	Set the bottom print margin. The default is **0.5** inches.

Wrap Lines

If this is selected, lines which are too long to fit within the space between the margins will be continued on the next line with a '`---->`' continuation marker.

Number Lines

If this is selected, lines are numbered in a 4 digit field. Files with more than 9999 lines will format line numbers incorrectly after 9999 lines.

Header

You may enter any header text not longer than 40 characters here. The default is '`%f - %d`'. The special sequences are:

'`%f`' which is replaced by the name of the filename being printed,

'`%d`' which is replaced by the date,

'`%p`' which is replaced by the current page number.

'`%l`' which is replaced by the number of the last page.

Footer

You may enter any footer text not longer than 40 characters. The default is '`Page %p of %l`'. The special sequences are as described for the **Header**.

Font

Choosing Font opens the Font Selection dialog. You may select only those printer fonts available which are monospaced.

File Exit command

File | Exit exits CLint++. If you've modified documents without saving, you'll be prompted to save before exiting. If you check the Options | Edit Options ***Restore Session*** checkbox, the position and state of every open window will be restored when you next use CLint++.

Edit Menu

The Edit menu provides commands to undo or redo edits, access the clipboard, and to delete text.

Undo	Undo the previous operation.
Redo	Redo the previous undone operation.
Cut	Delete selected text and move it to the clipboard.
Copy	Copy selected text to the clipboard.
Paste	Move text from the clipboard to the current document.
Delete	Delete selected text.
Select All	Select the entire document contents.
Record Macro	Start recording a macro.
Play Back Macro	Play back a recorded macro.

Record Macro command

Edit | Record Macro starts recording a keyboard macro. Only those keystrokes in the window which don't involve dialog boxes are recorded. At most 255 keystrokes may be recorded in a macro. Macro definitions are lost when you exit CLint++. Recording is terminated either by Escape, or by selecting this menu entry again, when it's legend reads Stop Recording, or by typing 256 characters.

Play Back Macro command

Edit | Play Back Macro if available replays the macro recorded by Edit | Record Macro. You can use Options | Keyboard Assignments to assign this to any available key press.

Edit Undo Command

Edit | Undo if selectable undoes the most recent edit action in the current file. CLint++ remembers up to 1000 edit operations if you have enough memory. By default, only those actions which occurred since the most recent file save are remembered, but you can choose to remember all operations regardless of file saves by setting the Options | Editor Options ***Undo past earlier save*** checkbox.

Edit Redo Command

Edit | Redo if selectable undoes the most recent undo action in the current file. Redoing an action makes it available for further undoing by Edit | Undo. CLint++ remembers up to 1000 edit operations if you have enough memory. By default, only those actions which occurred since the most recent file save are remembered, but you can choose to remember all operations regardless of file saves by setting the Options | Editor Options ***Undo past earlier save*** checkbox.

Edit Cut Command

Edit | Cut if selectable copies the selected text to the clipboard, and then deletes the selected text from the file. To paste the copied text into another document, choose Edit | Paste. Use the Edit | Undo command to restore the deleted text.

Edit Copy Command

Edit | Copy if selectable leaves the selected text intact and places an exact copy of it in the clipboard. To paste the copied text into another document, choose Edit | Paste. This operation can't be undone.

Edit Paste Command

Edit | Paste can only be selected if there is text in the clipboard. It inserts the text currently selected in the clipboard into the current window at the cursor position, or replaces the currently selected text with the pasted text. Use the Edit | Undo command to restore the text.

Edit Delete Command

Edit | Delete deletes the currently selected text from the current document. The text is not placed in the clipboard. Use the Edit | Undo command to restore the text.

Edit Select All Command

Edit | Select All selects the entire file. You can then copy the file to the clipboard, or save the file to a new file with File | Save Selection As.

Search Menu

The Search menu provides commands to find and replace text.

Find	Find a pattern of text.
Find Next	Find the next occurrence of text pattern.
Find Previous	Find the previous occurrence of text pattern.
Replace	Replace one pattern of text with another.
Find Matching Bracket	Find the matching occurrence of any bracket.
Find Functions	Find all functions in loaded files.
Find Files Containing	Find files containing a given pattern.
List Found Files	Show a list of files found by Find Files Containing.
Goto Line	Go to a specified line in the document.
Goto Bookmark	Goto a previously remembered bookmark.
Set Bookmark	Note a bookmark at the current document position.

Search Find dialog

Search | Find searches the current document for a text pattern. The command displays the Find dialog which controls the search process. Options in the dialog determine whether the case of characters is significant. As each match is found, it is highlighted in the document.

Pattern

Enter a pattern to search for, or select a pattern from the drop down list of the 9 most recent patterns you used. If the current Edit Window has a selection, this box is prefilled with it.

Match Upper/Lower case.

When checked, CLint++ matches the case of the pattern exactly, otherwise case is ignored.

Regular expression

When checked, CLint++ uses Regular Expression searching to locate the pattern, which may contain special characters to control the search.

Fwd

Starts the search forward from the current cursor position.

Back

Starts the search backward from the current search position.

Regular Expressions

Regular Expressions are a powerful method for specifying searches in text. All characters other than those detailed below are treated as simple match characters, and match exactly.

The standard special patterns are summarized here:

`[letter1-letter2]`	matches one occurrence of all letters in the range letter1 to letter2
`[^letter1-letter2]`	matches one occurrence of all letters not on the range letter1 to letter2
`[letter1-letter2]*`	matches zero or more occurrences of all letters in the range
`[letter1-letter2]+`	matches one or more occurrences of all letters in the range
`[letter1-letter2]?`	matches zero or one occurrence of all letters in the range
`.`	(period) matches any single character - including space and tab
`^`	matches the start of a line
`$`	matches the end of a line
`:*`	matches zero or more non-blank characters
`:+`	matches one or more non-blank characters
`:?`	matches zero or one non-blank characters
`\t`	matches a single tab
`\( expr \)`	matches the expression - used to control `|`

A regular expression is zero or more **branches**, separated by '`|`'. It matches anything that matches one of the branches.

A branch is zero or more **pieces**, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an **atom** possibly followed by '`*`', '`+`' or '`?`'. An atom followed by '`*`' matches a sequence of 0 or more matches of the atom. An atom followed by '`+`' matches a sequence of 1 or more matches of the atom. An atom followed by '`?`' matches a match of the atom, or the null string.

An **atom** is a regular expression in parentheses - `\(` followed by `\)` - (matching a match for the regular expression), a **range** (see below), '`.`' (period - matching any single character), '`^`' (caret - matching the null string at the beginning of the line), '`$`' (matching the null string at the end of the line), a '`\`' followed by a '`t`' matching a single tab character, a '`\`' followed by a single character other than '`t`','`(`', or '`)`' (matching that character), '`:*`' matching zero or more non-blank characters, '`:+`' matching one or more non-blank characters, '`:?`' matching zero or one non-blank character, or a single character with no other significance (matching that character).

A **range** is a sequence of characters enclosed in '`[]`'. It normally matches any single character from the sequence. If the sequence begins with '`^`', it matches any single character ***not*** from the rest of the sequence. If two characters in the sequence are separated by '`-`', this is shorthand for the full list of ASCII characters between them (e.g. '`[0-9]`' matches any decimal digit). To include a literal '`]`' in the sequence, make it the first character (following a possible '`^`'). To include a literal '`-`', make it the first or last character.

To find all occurrences of a misspelled '***function***' you might use the pattern:

```
fu[a-z]*on
```

which matches any string starting with '**fu**', followed by any number of lower case letters, and ending in '**on**'.

Regular expression searches although more powerful, are slower than direct searches.

Search Next command

Search | Next command repeats the last Find, searching forward, but otherwise using the same settings as the original find.

Search Previous command

Search | Previous command repeats the last Find, searching backward, but otherwise using the same settings as the original find.

Search Replace dialog

Search | Replace searches the current document for a text pattern, and replaces occurrences of the pattern with new text. The command displays the Replace dialog which controls the search/replace process. Options in the dialog determine whether the case of characters is significant. The dialog is used to specify the pattern to search for, and the text to replace occurrences with.

Pattern

Enter a pattern to search for, or select a pattern from the drop down list of the 9 most recent patterns you used. If the current Edit Window has a selection, this box is prefilled with it.

Replace with

Enter a replacement pattern, or select a pattern from the drop down list of the 9 most recent replacements you used.

Match Upper/Lower case.

When checked, CLint++ matches the case of the pattern exactly, otherwise case is ignored.

Regular expression

When checked, CLint++ uses Regular Expression searching to locate the pattern, which may contain special characters to control the search.

Confirm changes

When not checked, CLint++ performs all the replacements without prompting.

Start

Starts the replacement. If you didn't check **Confirm changes**, the Confirm Replace dialog pops up.

Confirm Replace dialog

This dialog pops up when you choose **Start** from Search | Replace, and you did not check ***Confirm changes***. The Edit Window shows the match highlighted.

Replace all

Check this box if you don't want to be prompted further for replacements.

Yes

Choose Yes to replace the selected text, and search for the next instance.

No

Choose No to skip and search for the next instance.

Cancel

Cancels the replacement operation.

Search Find Matching Bracket command

Search | Find Matching Bracket finds the succeeding matching bracket if the insertion point is to the left of one of:

`{`, **`[`**, or **`(`**

and the preceding matching bracket if the insertion point is to the left of one of:

`}`, **`]`**, or **`)`**

Search Find Function command

Search | Find Function collects all the function definitions from all open documents which are searchable, and adds then to the current functions list, and the pops up the Find Function dialog to allow you to choose a function definition. The file containing the chosen definition is opened, and the cursor positioned on the first line of the function body.

In order to use **Find Function**, you must enable the Edit File Properties ***Function Search*** checkbox for the document types you want to search. The default C/C++ editor document type has this checkbox enabled by default.

The search engine recognizes C or C++ or C-like functions wherever defined. It distinguishes between declarations (including in classes), and definitions (whether in classes or not), and shows only the definitions in the list. Functions which are **static** have the marker **[static]** after the function name. Functions which are **virtual** defined in a class definition have the marker **[virtual]** after the function name.

The search engine only reads files which are newly loaded, or have been changed since the last Find Function command. Although it is very fast, this avoids delays when repeatedly using Find Function with many loaded files.

Function List

This combo box allows you to select from the listed functions. You can type the first few letters of the function name, and the list box part will change to show the first function matching those letters you typed. You can directly select entries in the list box part also.

Goto

Choosing Goto opens the file containing the currently highlighted selection (if not already open), and positions the cursor on the first line of the function body.

Clear

When chosen, the function list is emptied, and the dialog dismissed. Choose this when you have too many functions from earlier searches cluttering the list.

Search Find Files Containing dialog

Search | Find Files Containing dialog searches files for occurrences of a pattern you specify.

Pattern

Enter a pattern to search for, or select a pattern from the drop down list of the 9 most recent patterns you used. If the current Edit Window has a selection, this box is prefilled with it.

Files

Enter one or more wildcard patterns, or choose from the drop down list of the 9 most recent such file patterns you used.

Match Upper/Lower case.

When checked, CLint++ matches the case of the pattern exactly, otherwise case is ignored.

Search Subdirectories

When checked, CLint++ will also search subdirectories of the starting directory you specify.

Regular expression

When checked, CLint++ uses Regular Expression searching to locate the pattern, which may contain special characters to control the search. This search will be slower.

Directory:

This edit field allows you to directly type a directory name. It also shows the current directory resulting from navigating around the disk using **Directories** and **Drive**. It is the directory within which searching will occur when you choose OK.

Directories

This list box contains a list of directories in the current directory. Clicking on any entry causes that to become the current directory, and updates the **Directory** edit field accordingly.

Drive:

This combo box contains a list of on your machine. Selecting any entry causes that to become the current drive, and updates the **Directory** edit field and **Directories** list box accordingly.

Find

Starts the search.

Matching Files dialog

The Search | List Found Files command pops up this dialog. It shows the pattern used in the most recent file search, the file wildcard pattern used, and a list of files containing that pattern you've not yet examined.

Files

This list box holds a list of all files which contain the given pattern that you've not yet examined. Double click on one. or select one and choose **Goto** to open a new or existing Edit Window on the file.

Goto Line dialog

The Search | Goto Line command pops up this dialog.

Line Number:

This combo box lets you type a line number in your file, or select one of the 9 most recently used line numbers.

Goto

Moves you to the line.

Book Marks dialog

The Search | Goto Book Marks command pops up this dialog.

Bookmark List

This list box contains the filenames and line numbers of the currently remembered bookmarks. You can double click on one, or select it and chose **Goto** to be placed at the remembered line in the file.

Delete

Choosing delete removes the selected bookmark from the list.

Set Book Mark command

The Search | Set Book Mark command adds the current file and line number to the Search | Goto Book Marks dialog box list.

Lint Menu

New Lint File	Create a new project.
Open Lint File	Open an existing project.
Edit Lint File	Edit the current lint document.
Lint Source	Lint the current program or file.
Generate Metrics	Generate prototypes for the current program or file.
Preprocess	Preprocess the current file.
Generate Globals	Generate prototypes for the current program or file.
Generate Library	Generate prototypes for the current program or file.
Generate Prototypes	Generate prototypes for the current program or file.
Dependencies	Generate dependencies for the current program or file.
Message	Open the analysis message window.
Version Status	Display the version control status of the current program or file.
DOS Command	Run a DOS command.
Lint Options	C or C++ source lint options

Generate Library command

Lint | Generate Library is only available if there are header files defined in the Lint File Window. You can drag/drop header files in the same directory as the Lint File into this window - they are taken to be header files defining the publicly visible interface of the library. The command generates a file of the ***projname*.CLB** where *projname*.CLN is the name of the Lint File. This library is generated as though by the DOS CLint++ command line:

```
            clint header_file_list -Lprojname.CLB
```

By default only those symbols prototyped or declared **extern** in the header files are written to the library file as definitions - symbols from included headers are not written to the output.

For **C++**, all non-**inline** member functions, and **static** members are also written to the library file as definitions. The assumption is that a library defines all the member functions in these headers.

If any errors or warning occurred, and the Message Window is not open, it is opened to display them.

Generate Globals command

Lint | Generate Globals performs the standard CLint source analysis, and writes the declarations of all globally visible symbols to the file ***projname*.~GL** where *projname*.CLN is the name of the Lint File. This file is generated as though by the DOS CLint++ command line:

```
          clint header_file_list -Xprojname.~GL
```

and then opens the file ***projname*.~GL** in a new Edit Window, and then deletes the output file.

If any errors or warning occurred, and the Message Window is not open, it is opened to display them.

Generate Prototypes command

Lint | Generate Prototypes performs the standard CLint source analysis, and writes the prototypes of all non-**static** functions to the file ***projname*.~PR** where *projname*.CLN is the name of the Lint File. This prototype file is generated as though by the DOS CLint++ command line:

```
            clint file_list -p=projname.~PR
```

and then opens the file ***projname*.~PR** in a new Edit Window, and then deletes the output file.

If any errors or warning occurred, and the Message Window is not open, it is opened to display them.

New Lint File command

Lint | New Lint File opens the Save As dialog for you to specify a new lint file. Once created, it opens the Lint File ready for you to add files and specify options.

When first created, CLint++ inspects the directory containing the new Lint File for header files ***.h??** and adds the to the project as ***Headers***, and for files with the extensions:

```
*.a?? *.def *.dlg *.inc *.log *.m?? *.rpt *.rc *.s?? *.t??
```

and adds them to the project as ***Other***. To complete the project, drag/drop the C or C++ files you require into the window, and drag/drop any libraries (*.CLB files see Lint | Generate Library) into the window.

Lint File

A lint file has a CLN extension and contains a series of directives meaningful to both CLint++ for Windows and the DOS command line program. The file is commented ASCII text, and may be placed under version control, or viewed and edited as text. The directives specify the include paths, checking libraries, preprocessor symbols, function names, configuration options, header files, additional files, and one or more source files. You can most easily change these options by right-clicking in the Lint File Window and choosing Edit Lint File.

You can drag files from File Manager and drop them on a Lint File Window. Doing so adds them to the list of source files.

You can drag any ***.CLB** file into a Lint File Window. This has exactly the same effect as adding the *.CLB file to the project using Lint **|** Edit Lint File, choosing the ***New*** button in the ***Checking Libraries*** group, and browsing for the file - but much easier.

You can drag any ***.H??** file in the same directory as the Lint File. This specifies files to be used when creating ***.CLB** checking libraries with Lint **|** Generate Library, see Lint **|** New Lint File for details.

When you ask CLint++ to check a source file, it looks in the directory containing the source file for a Lint File. The first such file seen is taken to contain the options to use to control the checking process. When used this way, the source file list in the lint file is ignored.

PROJECT.CFG

Additional include paths or libraries may be specified by placing a **PROJECT.CFG** file in the same directory as the lint file. This file may only contain comments, or **`-I`**, **`-D`**, **`-l`**, or warning switches. This is useful when a lint file may be used in more than one directory, and you need to specify paths to reach include files or checking libraries. When your program is used this way, be sure to specify checking library names without path components. CLint++ expects to find such checking libraries in one or more directories specified by the **`-l`** switch.

PROJECT.CLN

The DOS command line version of CLint++ treats the file **PROJECT.CLN** specially: if present in the directory in which it is invoked, no command line arguments need be supplied. Instead, it reads the file and gets all lint options and the file list, and checks those files as though you had typed their names on the command line. For this reason, you may find it convenient to name all your program files **PROJECT.CLN**. When CLint++ is used this way, it accepts command line options which add to or override options present in the lint file. To generate prototypes for example, just type:

```
CLINT -p
```

and prototypes for all your functions will be written to the screen. See **`-p`** for more information on this option, and CLint++ for DOS for full details of all the command line and lint file switches.

Open Lint File

Lint | Open Lint File opens the Open dialog for you to specify an existing Lint File which it opens.

Edit Lint File

Lint | Edit Lint File or right clicking the Lint File Window | Edit Lint File pops up the Lint Options dialog from which you can examine or modify any of the options specific to the program. The dialog box used is the same as that used for the default lint options, except there are a few additional program specific controls.

When preparing a library file of your own, one of the best ways is to create a Lint File using Lint | New Lint File, and than drag all the source files of your project into it, and then use Lint | Generate Library.

You can drag any checking library you create (CLB files) into a Lint File Window. This has exactly the same effect as adding the *.CLB file to the project using Lint | Edit Lint File, choosing the ***New*** button in the ***Checking Libraries*** group, and browsing for the file - but much easier.

See also Lint File.

Lint Source

Lint | Lint Source analyzes the active window for programming errors. If the active window is a file, that file is analyzed in isolation using (if one exists) the lint file in the directory containing the file for options. If the active window is a lint file, all files in the program are analyzed together, and additional programming errors only visible when all files are considered are reported.

If any errors or warning occurred, and the Message Window is not open, it is opened to display them.

Preprocess

Lint | Preprocess preprocesses the active window into a new Edit Window, and then deletes the output file. All comments are preserved, and the preprocessed output can be compiled or checked directly. To preserve whether C or C++ source was preprocessed, CLint++ generates ***projname*.~cp** files for C source, and ***projname*.~pp** files for C++ source, where *projname*.CLN is the name of the Lint File. The file is generated as though by the DOS CLint++ command:

```
        clint file -Eprojname.~PP
```

If any errors or warning occurred, and the Message Window is not open, it is opened to display them.

Generate Metrics

Lint | Generate Metrics analyzes the active window for function metrics into a new Edit Window, and then deletes the output file. If the active window is a file, that file is analyzed in isolation using (if one exists) the Lint File in the directory containing the file for options. If the active window is a Lint File, all files in the program are analyzed together, and all function metrics are reported. The file is generated as though by the DOS CLint++ command:

```
clint files -Rprojname.~pm
```

where *projname* is the name of the source file.

The output is a comprehensive set of metrics for the source code which looks like this:

```
 Line Function                                      Lines Loop Goto Cont
Effort
cpextdef.cpp
  121 do_pragma(CSTR)                                  68    1    0    0
13.69
  197 TokenStream::deftoken(Operator, char *, WORD)    18    0    0    0
1.87
 2004 checkClass(StorageClass &, StorageClass)         16    0    0    0
0.28
 3246 CppParser::extdef()                             407    1   21    5
95.63
```

Lines

is the number of source lines in the function body

Loop

is the number of for/while/do loops in the function

Goto

is the number of gotos in the function

Cont

is the number of continues in the function

Effort

is Halstead's effort metric modified for C/C++

If any errors or warning occurred, and the Message Window is not open, it is opened to display them.

Dependencies

Lint | Lint Source analyzes the active window for dependencies into a new Edit Window, and then deletes the output file. If the active window is a file, that file is analyzed in isolation using (if one exists) the list file in the directory containing the file for options. If the active window is a lint file, all files in the program are analyzed together, and all dependencies are reported. The file is generated as though by the DOS CLint++ command:

```
clint files -Mprojname.~mk
```

where *projname* is the name of the source file.

The output is a very comprehensive skeleton of a Makefile which you can use by saving the window with an MAK extension, or paste into makefiles of your own. The output looks like this:

```
##AUTOMACRO##
LNTFLGS =   -R5
MAK = cpextdef.mak

# Object files
OBJ = cpextdef.obj

# Source files
SRC = cpextdef.cpp

# Local header files
HDR = cpparse.h lexical.h

##END_AUTOMACRO##

##AUTODEPEND##

cpextdef.obj: cpextdef.cpp cpparse.h lexical.h

# Indirect file
cpextdef.tmp: $(MAK)
echo +cpextdef  & > cpextdef.tmp

lint: cpextdef.cln $(SRC) $(HDR)
      clint cpextdef.cln $(LNTFLGS) -o cpextdef.rpt
      echo > lint

lintlib:    cpextdef.cln $(HDR)
      clint $(HDR) -Lcpextdef.clb
      echo > lintlib

clean:
      del *.obj
      del *.tmp

##END_AUTODEPEND##
```

The section **##AUTOMACRO##** specifies Make macros used in the main section **##AUTODEPEND##** which defines the targets:

lint

specifies the commands need to run the DOS CLint++ and write its output to a report file.

lintlib

specifies commands needed to generate a *.CLB checking library.

clean

specifies commands to remove object and temporary files from a directory.

object files

specifies commands to make the object files assuming a standard default rule for running your compiler.

indirect file

specifies commands to write a minimal indirect file for running you linker.

If any errors or warning occurred, and the Message Window is not open, it is opened to display them.

Message Window

Lint | Message opens the Message window, or it is opened automatically if any errors or warnings occur as a result of source processing activities..

The message window is used to display errors and warnings arising from source processing activities. You can double click on any error or message line, and CLint++ will place you on the line of source text at the point of the error, opening a new Edit Window if needed. You can get specific help on any error or warning by selecting it and pressing F1.

There are several message styles which can occur:

Checking <file>:	This message announces that CLint++ has started analyzing the file.
Preprocessing <file>:	This message announces that CLint++ has started preprocessing the file.
Message file(NNN): text	This shows the text resulting from handling a #message directive line **NNN** of **file**.
Error file(NNN): text	This shows the text of an error message caused by a major problem at line **NNN** of **file**.
Warning file(NNN): text	This shows the text of a warning message caused by a minor problem at line **NNN** of the **file**.

When you right-click on the message window, a menu pops up showing:

Goto Error Line	Same as double clicking the line
Help on this Error	Same as pressing F1, gets specific help on the error
Next Error	Same as pressing F3, move the selection to the next error, or beep if no more errors
Previous Error	Same as pressing Ctrl-F3, move the selection to the previous error, or beep if no previous error
Save messages	The contents of the message window are written to a new Edit Window as a file called ***report.~lg*** in the directory containing the file from which the first message originated. The file does not actually exist, you have to save the window with Save As to create it.
Help	this Help page

You can control the detailed reporting of warnings by using the preprocessor #pragma directive.

When output from compilers or other DOS command line tools is captured, CLint++ parses the line looking for errors and warnings. You can double click on any such lines to go directly to the source line. Help is not available for lines obtained by capturing the output of a DOS Command, as CLint++ has no way of knowing the details of the command line tool you ran. Use **Next Error** or **Previous Error** to rapidly find those lines.

DOS Command

Lint | DOS Command runs the command specified for the current document type by the DOS Command Setup dialog. Each document type may have a separate DOS command specified to permit compilers, Make, and other tools to be run on the file.

If you selected ***Prompt for parameters*** in the DOS Command Setup dialog, the DOS Command Argument dialog pops up to allow you to edit the command. You can use one of these macros in the command line:

`%f`	replaced by the document filename
`%F`	replaced by the full pathname of the document
`%D`	replaced by the directory part of the document filename
`%b`	replaced by the basename of the file - that is the name with no extension.
`%e`	replaced by the extension of the file with no name or period.

If you selected ***Collect output*** in the DOS Command Setup dialog, CLint++ will attempt to parse the collected output to determine if the lines are error or warning lines from a translator. These lines begin with either **Error** or **Warning**, which is followed by a filename and line number. If it finds lines like this, then you can double-click in the line in the Message Window to go directly to the line containing the error. CLint++ can't determine the column of the error detected by external tools, and can't provide context help for such lines.

DOS Command Arguments

This dialog pops up when you select Lint | DOS Command, and you selected ***Prompt for parameters*** in the DOS Command Setup dialog.

It shows you the directory in which the command will execute, and the proposed command. You can accept the proposed command as is, or you can edit the command, or Cancel to dismiss the dialog.

If you selected ***Collect output*** in the DOS Command Setup dialog, CLint++ will capture all output written by the command to the DOS screen, and present that captured output in the Message Window when the command terminates.

Window Menu

The Window menu provides commands to control the position and layout of application's windows.

New Window	Open a new window on the current document.
Cascade	Resize and position all windows in an overlapping pattern.
Tile Horizontally	Resize and position all windows in an non overlapping pattern.
Tile Vertically	Resize and position all windows in an non overlapping pattern.
Arrange Icons	Align all iconized windows along a grid.
Close All	Close all windows.
Next	Switch to the next window.

New Window command

Window | New Window creates a window looking into the active Edit Window document. As these new windows are created, the titles reflect the order in which they were created. You may make any changes in any of the multiple windows into the underlying file. There is only one Undo history for the file, so operations performed in one window may be undone in another.

Window Cascade command

Window | Cascade arranges all file windows from the top-left position of the application's main window so that the title bar of each is visible.

Window Tile Horizontally command

Window | Tile arranges all windows one above the other in a non-overlapping pattern.

Window Tile Vertically command

Window | Tile arranges all windows side-by-side in a non-overlapping pattern.

Window Arrange Icons command

Window | Arrange Icons arranges all iconized windows into rows along the bottom of the application's main window.

Window Close All command

Window | Close All closes all windows open in the application. You will be asked if you wish to save changes for each changed document being closed.

Window Next command

Window | Next switches to the next window.

Help Menu

The Help menu provides access to the help system and the About dialog.

Help on CLint	Help topic contents.
Using Help	How to use the Windows Help system.
About	Information on CLint++ and Windows.

Network Menu

The Network menu provides commands for viewing received mail, viewing you mail folders, sending mail, handling Work Groups, handling the User list, handling Projects, and receiving report Notifications.

In Basket	View the contents of your In Basket.
Mail Folders	View your mail folders.
New Mail	Create new mail.
Work Groups	Add or change Work Groups.
Users	Add or change User details.
Projects	Add or change Project details.
Notifications	Collect any new notifications.

In Basket dialog

The Network | In Basket command pops up this dialog. If you need to enter your Password, that dialog pops up first.

Messages

This list shows the date the mail was created, the mail type (one of Mail, CC, BCC), the logname of the sender, and the subject. Select the message you want and double click to view, or press the View button.

View button

The View button opens the View Mail dialog with the selected message displayed.

View Mail dialog

The In Basket View button pops up this dialog.

From

This shows who the mail is from.

Date

This shows when the mail was sent.

To

This shows the list of recipients - it always shows you, and shows any other recipients also.

Cc

This shows any recipients who were Cc'd this mail.

Subject

This shows the mail subject.

Reply

If you choose this button, the Reply Options dialog is opened where you specify how to reply with the Send Mail dialog.

Save

This button pops up the Save Folder dialog, so you can specify which folder to save your mail in. If you don't save you mail here, or delete it, it will be automatically saved for you in the Autosaved Mail folder which is created if required.

Delete

This deletes the mail message permanently.

Attached

This button is only visible if there are attachments. It shows the Attached Files dialog which allows you to read an attached file into CLint++.

Attached Files dialog

The View Mail Attached button pops up this dialog. It shows a list of attached files.

Open

This button opens the selected attachment in CLint++ for editing or viewing.

Save Folder dialog

The View Mail Save button pops up this dialog.

Folders

Shows the list of available mail folders. Double click on one to save in that folder.

Save

Select a folder name from **Folders**, then choose this to save the mail.

New

This button opens the New Folder dialog where you enter the name of a folder to create.

New Folder dialog

The Save Folder New button pops up this dialog. Enter the name of the new folder and choose OK, or choose Cancel.

Reply Options dialog

The View Mail Reply button pops up this dialog. The settings you choose are remembered.

Reply to all recipients.

When checked, the reply will be sent to all **To** recipients as well as the original sender.

Reply to CC recipients.

When checked, the reply will be sent to all **CC** recipients as well as the original sender. It is impolite to use this without also using **Reply to all recipients**.

Include original text.

When checked, the reply message window in the Send Mail dialog will be pre-filled with the message text. This is helpful when you want to quote part or all of the original message.

OK

Enters the Send Mail dialog with the options you chose.

New Mail dialog

The Network | New Mail command pops up this dialog. If you need to enter your Password, that dialog pops up first. You must enter as a minimum a recipient, and a subject. Message text is optional.

To

This button pops up the Address dialog where you can edit or add recipients to the list.

Cc

This button pops up the Address dialog where you can edit or add recipients to the list of courtesy copies.

Bcc

This button pops up the Address dialog where you can edit or add recipients to the list of blind courtesy copies. The recipients won't be able to see if any Bcc copies were sent

Subject

Enter the subject of your mail. A subject must be supplied.

Attach

This pops up the Attach Files dialog where you can edit or add attachments to your message. Recipients will get copies of the files delivered to their local mail directories.

Send

This button sends the mail.

Receipt

Check this box if you want a receipt. You will get receipt mail from each recipient as they read the message.

Address dialog

The Send Mail dialog To, Cc, and Bcc buttons pop up this dialog.

Address list

This list shows all possible recipients. Projects are shown with a '**$**' prefix, and if chosen refer to all users assigned to that project. Double clicking an entry adds it to the **To** list.

To

This list shows the current recipients. Double clicking an entry deletes it.

<<

This button deletes the selected recipient in the **To** list.

>>

This button adds the selected user in the **Address** list to the **To** list.

Attach Files dialog

The Send Mail dialog Attach button pops up this dialog. It shows a list of currently attached files.

Add

This button pops up the Open File dialog for you to select a file to attach.

Remove

This button removes the selected attachment from the list.

Password dialog

The Password dialog pops up if you choose any action which requires your password. It is titled with the action for which the password is required.

Password

Enter your password in this edit field, and press OK.

Change Password

If you check this box, you must enter your new password in the **New Password** and **Password Again** boxes, and then choose OK.

New Password

This edit field is only active if you check **Change Password**. You must enter your new password here, and again in the **Password Again** box.

Password Again

This edit field is only active if you check **Change Password**. You must enter your new password in **New Password**, and again here.

OK

Dismisses the dialog box if your password is valid, or has been changed. Otherwise, you are prompted to reenter your password.

Cancel

This dismisses the dialog, and abandons the attempted action which required your password.

Mail Folders dialog

The Network | Folders command pops up this dialog. If you need to enter your Password, that dialog pops up first.

Open

Select a folder name from the list, and choose this button, or double click the folder name in the list to view the contents of your folder. The In Basket dialog is used to display your mail messages.

Delete

This button deletes the selected folder. The folder and all its contents are permanently deleted.

Work Group dialog

The Network | Work Groups command pops up this dialog. If you need to enter your Password, that dialog pops up first.

Group List

This list shows all the currently defined Work Groups. There may be no more than 16 groups. They are equivalent to summary job descriptions, and CLint++ predefines the groups Administrator, Engineer, Manager, and Tester for you. Double click any entry to edit it

Edit

This pops up the Edit Work Group dialog to modify details of an existing group.

New

This pops up the Edit Work Group dialog where you enter details about the new group.

Edit Work Group dialog

The Work Group dialog Edit button pops up this dialog.

Work Group

Enter or modify the name of the Work Group here.

Group ID:

Enter or modify the Work Group ID here. There may be no more than 8 characters in the ID.

Permissions

This button pops up the Permissions dialog where you set the basic permissions for a Work Group.

Permissions dialog

The Edit Work Group dialog Permissions button pops up this dialog, as does the Edit User dialog Permissions button. For a Work Group, all check boxes are enabled, allowing administrators or Managers to set the possible permissions for whole classes of users. For individual users, only those checkboxes which were set in the Work Group of which they are a member are enabled. Administrators and Managers can set specific permissions for the individual as long as they are a subset of the Work Group permissions previously established.

Permissions for

This shows which Work Group or User the permissions are for.

General

These permissions should be reserved for Administrators or Managers only.

Can create Projects

The Projects dialog New button is only enabled for this user when checked.

Can edit Projects

The Projects dialog Edit button is only enabled for this user when checked.

Can create Users

The Users dialog New button is only enabled for this user when checked.

Can edit Users

The Users dialog Edit button is only enabled for this user when checked.

Can create Groups

The Groups dialog New button is only enabled for this user when checked.

Can edit Groups

The Groups dialog Edit button is only enabled for this user when checked.

Can specify Field names

The various Fields buttons are only enabled for this user when checked.

Originate Reports

These permissions govern what a user can change when submitting a report.

Set Originator

This permission is only useful for secretarial staff entering reports created by others. It allows the originator to change the originator name, which defaults to the user name.

Set Assignee

This permission is useful for Managers entering reports for immediate assignment. The default assignment is <none>.

Set Origination date

This permission is useful for secretarial staff entering reports created by others at some earlier time. The default is the current date.

Disposition

This permission is useful for anyone who enters informational reports - that is, reports which are closed, but entered to assure completeness of the information.

Classification

This permission is allows you to specify the classification when originated. The default is Normal.

Can Always

These permissions should only be used for Managers and Administrators. They allow arbitrary changes to important fields.

Change Originator

This permission is useful for Managers correcting reports created by others. It allows them to change the originator name.

Change Origination date

This permission is useful for Managers correcting reports created by others..

Change Assignee

This permission is required by Managers, allowing them to assign any report to others.

Change Disposition

This permission is required by Managers, allowing them to close any report.

Change Classification

This permission is required by Managers, allowing them to change the classification of any report.

Change Notes

This permission is required by Managers, allowing them to add or amend commentary material to any report.

Add Attachments

This permission is required by Managers, allowing them to attach supplementary files to any report.

Can If Assignee

These permissions should be used for those users who can be assigned reports, and need to record the work that they did. These permissions are only effective when the assignee field of the report is the user name.

Change Originator

This permission is useful only for Managers correcting reports created by others. It allows them to change the originator name.

Change Origination date

This permission is useful only for Managers correcting reports created by others..

Change Assignee

This permission is required by any user who can reassign reports assigned to them. This is useful when an Engineer may assign a report to a QA group for testing.

Change Disposition

This permission is required by any user able to close or otherwise change the disposition of reports assigned to them.

Change Classification

This permission is required by any user who can classify reports assigned to them.

Change Notes

This permission is required by any user who works on a report to allow them to make working notes concerning the work they did.

Add Attachments

This permission is required by any user who must attach files describing work done - for example, test reports.

Notifications dialog

This dialog is popped up by the Edit User dialog Notifications button. Users can set specific notifications for themselves. This allows users to disable notifications which are not relevant to their work.

The possible notifications are:

A record you originated changes

When checked, you are notified when some other user changed a record you originated.

A record you originated is closed

When checked, you are notified when some other user closes a record you originated.

A record is originated by you

Useful if you want a record of reports you originated.

A record is originated

When checked, you are notified when a record is created. This is useful for Managers who must examine and assign such records to other users.

A record is changed

When checked, you are notified when a record is changed. This is useful for Managers who want to track progress.

A record is closed.

When checked, you are notified when a record is closed.

A record assigned to you changes

When checked, you are notified when some other user changes a record assigned to you. Useful if you are working on a problem this report relates to, and additional information is added to the report.

A record assigned to you is closed.

When checked, you are notified when some other user closes a record assigned to you.

A record is newly assigned to you.

When checked, you are notified when a record is assigned to you. Useful when this informs you of new work you need to do.

A record assigned to you is reassigned.

When checked, you are notified when a record assigned to you is assigned to someone else by someone other than yourself. Useful when you need to know if your work is being reassigned.

User List dialog

The Network | Users command pops up this dialog. If you need to enter your Password, that dialog pops up first.

User List

This list shows the User ID, whether disabled, full user name, and current Work Group assignment for each user. If you have Edit permission for users, or you are the given user, double clicking an entry pops up the Edit User dialog.

Edit

If you have Edit permission for users, or you are the selected user, choosing this pops up the Edit User dialog.

New

If you have Create permission for users choosing this pops up the Edit User dialog to allow the details for a new user to be entered.

Edit User dialog

The User List dialog Edit button pops up this dialog.

Work Group

This combo box allows you to select the Work Group assignment for this user. It is not enabled unless you have Edit permission for users.

User ID

This edit box allows you to enter the user's ID. ID's may not have more than 8 characters. It is not enabled unless you have Edit permission for users, and you are creating a new user.

Password

This edit box allows you to specify the password for the user. It is not enabled unless you have Edit permission for users, or you are the named user..

Full Name

This edit box allows you to specify the full name of a user.   It is not enabled unless you have Edit permission for users, or you are the named user..

1 through 4

These fields are only shown if the field names have been defined using the **Fields** button. Use them to enter additional user data.

Permissions

This button pops up the Permissions dialog if you have Edit permission for users.

Fields

This button pops up the User Custom Entry Fields dialog if you have Edit Fields permission. You use it to configure the four optional fields which can be used to describe users.

Notifications.

This button pops up the Notifications dialog if you have Edit permission for users, or you are the named user.

User Custom Entry Fields dialog

The Edit User dialog Fields button pops up this dialog. You use it to define the names of the four optional user data entry fields. Fields with blank names are not displayed in the Edit User dialog. Enter the strings you require and choose OK.

Project List dialog

The Network | Project command pops up this dialog. If you need to enter your Password, that dialog pops up first. It allows you to create and edit projects, create and edit sub-projects, originate reports, open the Query Window, open the Report Window, or Intray Window for a project, export or import data to a project, or configure a project. Several of these actions require Edit permission on projects.

Project List

This list shows the project ID, project full name, and the number of users assigned to a project; or sub projects. Only users assigned to a project will receive notifications of reports created or changed in a project. Double clicking an entry pops up the Edit Project dialog if a project line, and the Sub Project Name dialog if a sub-project line. For given users without Edit Project permission, only those projects they are assigned to are visible to them.

Edit

This button pops up the Edit Project dialog allowing you to assign users, and specify the project name, if you have Edit Project permission.

New

This button pops up the Edit Project dialog allowing you to specify details of a new project, if you have Create Project permission.

Subproject

This button pops up the Sub Project Name dialog allowing you to specify the ID and full name of a sub project. There may be no more than 4 sub projects under any project. It is only available if you have Create Project permission, and there are fewer than 4 sub projects.

In Tray Window

This opens the Intray Window for the selected sub project. The window will only have entries if you have received notifications of reports from that sub project.

Query Window

This opens the Query Window for the selected sub project.

Report Window

This opens the Reports Window for the selected sub project.

Submit

This opens the Submit dialog for the selected sub project, allowing you to create a report for that sub project. You must have Originate permission to enable this button.

Import

This opens the Import dialog for the selected sub project, allowing you to import comma separated record files (CSV). You must have Edit Project permission to enable this button.

Export

This opens the Export dialog for the selected sub project, allowing you to export comma separated record files (CSV).

Fields.

This opens the Configure Form dialog for the selected sub project. You must have Edit Fields permission to enable this button.

Import dialog

The Project List dialog Import button pops up this dialog. It allows you to specify which records and in what order to import, and the file to import from. Import files must be in comma separated record form, with minor variations described below. The default values are appropriate for direct import from Excel CSV format. The imported records are appended to the database, so this is most useful when initializing a new database. No attempt is made to identify and remove duplicate entries if you import the same records more than once. You can remove the database files for a project, and re-import it if importing failed or botched fields, but be aware that deleting a database also deletes the field definitions which you will have to reconfigure.

Field Setup

Choose a previously stored setup from this combo box.

Import File

Enter the name of the import file, or use the **Browse** button to find the file.

Source Fields

This list contains the names of the fields in the order they will be parsed from the input file. You add fields by double clicking the field you need from the **Destination Fields** list.

>>

This button deletes the selected field name from the **Source Field** list.

<<

This button adds the field selected in the **Destination Field** list to the **Source Field** list.

Up

This button moves the selected **Source Field** up one line.

Skip

This button inserts the entry **<skip>** in the **Source Field** list. The corresponding input field will be ignored.

Down

This button moves the selected **Source Field** down one line.

Destination Fields

This list contains the names of all the fields in the database.

Save Field Setup

This button pops up the Save Setup dialog to allow the current field settings to be saved.

Date Format

This combo box offers the choices ***Control Panel***, ***M/D/Y***, ***D/M/Y***, and ***Y/M/D***. Choose the date format used by your import file.

Field Delimiter

This combo box offers the choices ***Comma*** and ***Tab***. Choose the delimiter used by your import file.

Quote Character

This combo box offers the choices ***Double Quote*** and ***Single Quote***. Choose the quote character used by your input file.

Import

This button is only enabled after an import file and one or more fields are defined. Choose it to import your file. Any format errors found while importing pop up the Import Error dialog to allow you to specify a recovery action.

Browse

This button pops up the File Open dialog to allow you to choose a file to import. The dialog will only see CSV files.

Fields

This button pops up the Configure Form dialog to allow you to specify the types and names of the fields.

Import Error

The Import dialog Import button pops up this dialog when errors are detected in the import file.

File

This names the import file.

Line

This gives the import file line number where the error was found.

Field

This edit box gives the current text of the import field. You may change it to a value which is more appropriate, or correct spelling errors, and retry importing the field.

Error

This box states the nature of the error.

Options

This offers six actions you can take to rectify the problem.

Add field as is.

The error is ignored, and the value used as is.

Retry changed.

This option is automatically selected for you if you change the Field text. It retries the import of the field.

Add choice

This option is offered if the field is a Choice List. If chosen, the text of the input field will be appended to the list of choices in the Choice List.

Today's date

This option is offered if the field is a Date or Date-Time. If chosen, the current date or date and time is written to the field.

Skip Record

The entire record is skipped. This is useful to skip input records which are simply title headings.

Leave blank

This cause the empty string to be saved in the field.

Fail Import

This causes the import process to stop. Since the import was incomplete, you may want to delete the database file, correct the problem, and retry the import.

Cancel

This causes the import process to stop. Since the import was incomplete, you may want to delete the database file, correct the problem, and retry the import.

Export dialog

The Project List dialog Export button pops up this dialog. It allows you to specify which records and in what order to export, and the file to export to. Export files will be in comma separated record form, with minor variations described below. The default values are appropriate for direct export to Excel CSV format.

Field Setup

Choose a previously stored setup from this combo box.

Export File

Enter the name of the export file, or use the **Browse** button to find the file.

Source Fields

This list contains the names of all the fields in the database.

>>

This button deletes the selected field name from the **Destination Field** list.

<<

This button adds the field selected in the **Source Field** list to the **Destination Field** list.

Up

This button moves the selected **Destination Field** up one line.

Skip

This button inserts the entry **<skip>** in the **Destination Field** list. The corresponding input field will be ignored.

Down

This button moves the selected **Destination Field** down one line.

Destination Fields

This list contains the names of the fields in the order they will be written to the export file. You add fields by double clicking the field you need from the **Source Fields** list.

Save Field Setup

This button pops up the Save Setup dialog to allow the current field setting to be saved.

Date Format

This combo box offers the choices ***Control Panel***, ***M/D/Y***, ***D/M/Y***, and ***Y/M/D***. Choose the date format used by your import file.

Field Delimiter

This combo box offers the choices ***Comma*** and ***Tab***. Choose the delimiter used by your import file.

Quote Character

This combo box offers the choices ***Double Quote*** and ***Single Quote***. Choose the quote character used by your input file.

Export

This button is only enabled after an export file and one or more fields are defined. Choose it to export your file.

Browse

This button pops up the File Save dialog to allow you to name the export file. The dialog will only see CSV files.

All Fields

This button copies all the fields in the **Source Field** list to the **Destination Field** list.

Edit Project dialog

The Project List dialog Edit and New buttons pop up this dialog. It allows you to specify the full name of a project, and assign users to a project. If creating a new project, you also specify the project ID.

Project ID

The project ID must not exceed 8 characters. This box is only enabled if creating a new project.

Full Name

Enter or change the full name of a project.

Users

This lists all users. Double click on any entry to add them to the Allocated Users list.

>>

This button copies the selected user in the User list to the Allocated Users list.

<<

This button deletes the selected user from the Allocated Users list.

Allocated Users

This lists all users allocated to the project. Only allocated users will receive notification of reports originated or changed in any of this project's sub projects. Double clicking an entry deletes an allocated user from the list.

Sub Project Name dialog

The Edit Project dialog Subproject button pops up this dialog, as does double clicking on a sub project line in the project list.

Short Name

Enter a short name not exceeding 8 characters for the sub project. This box is only enabled if creating a new sub project.

Full Name

Enter or change the full name of the sub project.

Help on CLint++ command

This opens the Windows Help application on the Contents page of this help file.

Using Help command

This opens the Windows Help application on the Windows supplied **How to Use Help** section.

About command

This pops up the About dialog which details your license, and details about your Windows configuration.

Copying Text

To copy text to a document from the clipboard, position the cursor at the desired insertion point, and choose Edit | Copy from the menu.

Deleting Text

To delete text from a document, select the text, and choose Edit | Cut or Edit | Delete from the menu. Cut will put the deleted text in the clipboard, and Delete will simply delete it.

Exiting

To exit the application, choose File **|** Exit from the menu.

Lint File Window

The lint file windows in CLint++ allow you to specify a collection of program files to be checked as a unit. You can add, delete, or drag-drop files into a lint file window. You can specify program specific lint options.

When you right click in a Lint File window, a menu pops up showing:

Edit	Specify project specific options
Edit as Text	Re-open the project file in an Edit Window
Add Files	Add files to the project. The hot key is Insert.
Delete this File	Delete the specified file from the project. The hot key is Delete.
Open this File	Open the specified file in an Edit Window.
Version Status	Display the version control status of the selected file.
Properties	Change the language assumed for the file
Lint Source	Analyze the current program.
Generate Metrics	Generate metrics for the current program.
Generate Prototypes	Generate prototypes for the current program.
Generate Globals	Generate globals for the current program.
Generate Library	Generate a library for the current program.
Dependencies	Generate dependencies for the current program.
Lint File Version Status	Show the Lint File's version control status.
Help	this Help page

Files are displayed like:

```
file.c                      [C]     [R]     [Version Control]
file2.cpp                   [C++]   [R]     [Version Control]
Header: header.h            [C++]
Other: dialog.dlg                           [Version Control]
Library: library.clb
```

Source files and headers show whether they are thought to be C or C++, which you can change from the Properties dialog, and whether they are currently read-only, and currently under Version Control. A read-only file is checked in and can't be modified, otherwise it has been checked out for editing. In the above example, ***header.h*** is not yet under version control, and ***dialog.dlg*** is currently checked out for editing.

Double-clicking the line of any file other than a library opens an Edit Window on the file.

Properties dialog

This dialog is selected from the LintFile Window | Properties pop-up menu. You can set the language for a file directly (most useful for headers) or choose automatic selection. By default CLint++ considers files of the form ***.C** to be **C** language, and files of the form ***.C??** to be **C++**. It chooses the language for headers based on whether there are more C or C++ files in the sources.

You can specify files which are not C or C++ as **None**, for example batch files. Such files won't be parsed.

To support the use of version control with files of types CLint++ can't edit (Word DOC files, Excel XLS files, Write WRI files, Project MPP files), you can specify that the file has the **No Edit** attribute. This is done automatically for you for the file types specified above,   and suppresses the normal double-click opening of the file. It also prevents the file from being parsed.

Report Window

The report windows in CLint++ display graphical reports of the Disposition field of the database with time. Reports are binned by week or month, and displayed according to the options specified in the Report Format dialog.

When you right click in a Report window, a menu pops up showing:

Format	Pops up the Report Format dialog
Print	Prints the contents of the window.
Help	this Help page

Report Format dialog

The Report Window | Format command pops up this dialog. It allows you to use or modify a previously saved report format, or create a new one. It displays up to 16 checkboxes for the first 16 choices in the first flex field of the database record. When these boxes are checked, the corresponding reports are displayed.

Week Rate

A line graph of the number of reports created of each enabled type each week from the earliest report to date.

Month Rate

A line graph of the number of reports created of each enabled type each month from the earliest report to date.

Week Cumulative

A stacked bargraph of the total number of reports of each enabled type each week from the earliest report to date.

Month Cumulative

A stacked bargraph of the total number of reports of each enabled type each month from the earliest report to date.

Week Line

A line graph of the total number of reports of each enabled type each week from the earliest report to date.

Month Line

A line graph of the total number of reports of each enabled type each month from the earliest report to date.

Use

Save the current settings, and redisplay the report using them. If the report has no title, you must supply one.

Edit Window

The edit windows in CLint++ allow reading, editing, printing and saving ASCII text files. Syntax coloring is supported as standard for C, C++, DOS Batch files, and Makefiles. You can configure the coloring, the keywords and syntax, and many other things required for arbitrary programming languages - see Configuring CLint for details.

When you right-click in a window, a menu pops up showing:

View	The currently active modes have tick marks.
as Text	format the window contents as text.
as Binary	format the window contents as binary (show all characters).
as Hex	format the window contents as Hex with 16 bytes per line. Editing may take place as normal in this view, but we advise over type mode to avoid changing the length of a binary file.
Line Numbers	Show line numbers.
Original Line numbers	Show original (unedited) line numbers. When lines are deleted or joined (even if later undone), the original line number for that line is lost.
Find	Find a pattern of text.
Replace	Replace one pattern of text with another.
Find Files Containing	Find files containing a given pattern.
List Found Files	Show a list of files found by Find Files Containing.
Goto Line	Go to a specified line in the file.
Lint Source	Lint the file.
Generate Metrics	Generate metrics for the file.
Preprocess	Preprocess the file.
Generate Prototypes	Generate prototypes for the file.
Version Status	Display the version control status of the file.
Run DOS Command	Run a DOS command.
Help	this Help page

The following standard special key strokes are effective in editor windows, although most of them can be reprogrammed from the Options | Key Assignments dialog. Of course, all the standard key operations are available also, and they too can be customized.

Ctrl-Y	Delete the current line.
Tab	If there is no selection, insert a Tab. If there is a selection entirely in a single line, replace the selection with Tab. If one or more complete lines are selected, indent the selected lines one tab stop.
Shift-Tab	If one or more complete lines are selected, unindent the selection one tab stop.
F2	If any characters are selected, make them all uppercase; otherwise make the character to the right of the caret uppercase, and advance the caret one position.
Ctrl-F2	If any characters are selected, make them all lowercase; otherwise make the character to the right of the caret lowercase, and advance the caret one position.
Escape	If there is a selection, clear it; otherwise ignored.
Alt-B	Set a bookmark at the current line.
Ctrl-B	Run the Bookmarks dialog to go to a previously set bookmark.
Ctrl-G	Run the Go To Line dialog to go to a specific line.
F4	If the cursor is on a bracket of any type or brace, jump to the matching bracket or brace.
F3	If a search has been previously conducted with Search | Find or Search | Replace, repeat the forwards.
Ctrl-F3	If a search has been previously conducted with Search | Find or Search |

	Replace, repeat the backwards.
Ctrl-S	Run the Search | Find dialog.
Ctrl-R	If a macro has been defined, replay it.
Alt-Backspace	Undo the most recent edit operation, including selections and cursor movement.
Shift-Backspace	Redo the most recent action which was undone.
Ctrl-P	Print the current file.

Query Window

The Query windows allow you to make queries of sub projects for reports. You can use previously defined queries, define new ones, sort the data, specify how fields are displayed, search for occurrences of strings in the displayed fields, submit new reports, and print the contents of the window.

Queries are stored in files with QRY extensions under a directory below CLint++'s directory with the same name as the Project ID. You can drag-drop QRY files onto CLint++ from File Manager.

The headings visible across the top of the window can be dragged to whatever position you select for display. They are remembered when you close the window, and are printed as shown on the window.

Double clicking any line, or typing Enter pops up the Edit Record dialog to allow you to edit the report.

When you right-click in a window, a menu pops up showing:

Find	Find a pattern of text.
Headings	Choose or define display fields in the window.
Queries	Choose or define a query to run on the sub project. Running a new query empties the window of its previous contents.
Sort	Choose or define a sort to run on the window contents.
Submit	Submit a new report to this sub project.
Copy to Intray	Copy the currently highlighted report to the sub project's Intray window.
Intray Window	Open or make active the sub project's Intray Window.
Report Window	Open or make active the sub project's Report Window.
Print	Print the contents of the window.
Help	this Help page

Copy to Intray command

This command copies the highlighted Query Window report to the Intray Window. It then opens the Intray Window if not already open, and makes it active. If you've not already entered your password, you are prompted to do so.

Intray Window

The Intray windows allow you to receive notifications of reports created or changed in sub projects. You can sort the data, specify how fields are displayed, search for occurrences of strings in the displayed fields, delete entries, submit new reports, and print the contents of the window.

Intrays are stored in files with ITR extensions under a directory below CLint++'s directory with the same name as the Project ID. You can drag-drop ITR files onto CLint++ from File Manager.

The headings visible across the top of the window can be dragged to whatever position you select for display. They are remembered when you close the window, and are printed as shown on the window.

Double clicking any line, or typing Enter pops up the Edit Record dialog to allow you to edit the report.

When you right-click in a window, a menu pops up showing:

Edit	Edit the currently selected record.
Find	Find a pattern of text.
Headings	Choose or define display fields in the window.
Sort	Choose or define a sort to run on the window contents.
Delete	Delete a report line.
Submit	Submit a new report to this sub project.
Query Window	Open or make active the sub project's Query Window.
Report Window	Open or make active the sub project's Report Window.
Print	Print the contents of the window.
Help	this Help page

Headings Setup dialog

The Query Window | Headings command or Intray Window | Headings command pops up this dialog. You use it to specify the names and order of the headings in either of these windows.

Display Setup

This shows the current setup description text. You can select other available stored setups for this sub project.

Fields

This is a list of all fields in the sub project database records. Double clicking on any entry copies the field name to the **Display Fields** list.

>>

This button copies the currently selected **Field** entry to the **Display Fields** list.

<<

This button deletes the currently selected **Display Field** entry.

Display Fields

This is a list of the field headings in the order they will be displayed in the window. You can change the order with the **Up** and **Down** buttons.

Up

This button moves the currently selected **Display Field** entry up one place.

Down

This button moves the currently selected **Display Field** entry down one place.

Query dialog

The Query Window | Query command pops up this dialog. You use it to choose a previously defined query for this sub project, or to define a new one, and then run the query on the database. All records matching the query conditions are copied into the Query Window formatted using the current headings. This destroys the previous Query Window contents.

Query

Select an existing query from this box.

Fields

This is a list of all fields in the sub project database records. Double clicking on any entry pops up the Compare dialog for you to specify a condition involving the field.

Save Query

This button pops up the Save Setup dialog to let you save the current query conditions for use at a later time.

Add Condition

This button pops up the Compare dialog for you to specify a condition involving the selected Field.

Conditions

This list contains the current set of conditions used for searching. Double clicking an entry pops up the Compare dialog for you to modify a condition. Conditions may be separated by **And**, **Or**, or **Not** operators. When searching, the conditions and operators are executed in the order they occur in this list.

And

This button inserts the **And** operator in the **Conditions** list.

Or

This button inserts the **Or** operator in the **Conditions** list.

Not

This button inserts the **Not** operator in the **Conditions** list.

Delete

This button deletes the selected **Condition** line.

Edit

This button pops up the Compare dialog to edit the selected **Condition** line.

Search

This button performs the search on the database. With a large number of entries, it may take several seconds.

Compare dialog

The Query dialog Add Condition button pops up this dialog. You use it to specify one of three comparison operators (equal, not equal, contains), for a search condition.

Field Type

This shows the type of the field.

Field name

This shows the name of the field.

Test for

Equal

Checks for an exact match between the pattern string and the field.

Not Equal

Checks for no match between the pattern string and the field.

Contains

Checks if the field contains the pattern string anywhere within it.

Match upper/lower case.

When checked, matches are case sensitive.

With

Enter a pattern string to compare with. When the field is of type User, this box is prefilled with the string $USER which always means you.

Edit Record dialog

The Query Window | Submit command or Intray Window | Submit commands pop up this dialog, as does double-clicking a line in either of these windows.

When entered from Submit, you are creating a new report, otherwise editing an existing one.

You must have appropriate permissions to make several of the changes described here.

Project

This shows the sub project to which the report belongs.

Record

This shows the record number, or for Submit the word New.

Last Modified

This shows the date when the record was last modified.

Description

This edit box allows you to enter as detailed a report as you wish up to a maximum of 42K of text.

Originator

This box shows the originator of the report, which will be you if you are creating a new report. You must have Change Originator permission to change it.

Date

This box shows the origination date, which is today if you are creating a new report. You must have Change Origination date permission to change it.

Assignee

This box shows who is currently assigned the report, or **<none>**. You must have Change Assignee permission to change it.

Date

This box shows the date the report was last assigned, or is empty. It is changed only by changing the Assignee box.

Assigned by

This box shows who last assigned this report to someone, or shows **<none>**. It is changed only by changing the Assignee box.

Disposition

This is the first of the 16 flex fields you can configure to suit your needs. By default it is of type Choice List, and has the possible values ***Closed***, ***Open***, and ***Deferred***. We advise that it remain a Choice List. The Report Window uses this field to categorize reports.

Severity

This is the second of the 16 flex fields you can configure to suit your needs. By default it is of type Choice List, and has the possible values ***Low***, ***Normal***, ***Urgent***, and ***Critical***. We advise that one of the choices always remain ***Normal***.

Record #

This is the third of the 16 flex fields you can configure to suit your needs. By default it is of type String, and records the record number, but may be used for any purpose if record numbers are not required.

Comment

This is the fourth of the 16 flex fields you can configure to suit your needs. By default it is of type String. It can be used for any purpose

Save

This button is only enabled if you have permission to edit or create the report. When chosen, the report is saved.

Print

This button is only available if the report is being edited. It prints the report headings, full description, all notes, and the log using as many pages as required. The report format can't be configured.

Attach

This button is only available if you have Attach permission. It pops up the Attach Files dialog to allow you to permanently attach files to the report, or retrieve previously attached files. Once attached, the files can't be deleted or modified. This is suited to test reports.

Notes

This button is only available if you have Notes permission. It pops up the Edit Notes dialog to allow you to modify or create notes. Once created, a Note can't be deleted, although it may be edited as often as required.

Log

This button is only available if viewing an existing report. It pops up the View Log dialog to allow you to see what changes were made, who made them, and when. Log entries can't be modified or deleted.

Fields

This button is only available if you have Fields permission. It pops up the Configure Form dialog where the types and names of the 16 flex fields may be specified.

Configure Form dialog

The Edit Record | Fields command pops up this dialog. It allows you to change the field names and types of each of the 16 flex fields in a report.

There are 16 check boxes in the dialog. By default the first four are enabled as described in the Edit Record dialog.

Checking any unchecked box checks it and pops up the Configure Field dialog with the field type set to String.

Unchecking a checked box disables the field, and removes it from the dialog.

When you disable a field, the only information lost is the type. To edit a field, uncheck and then check the box.

Configure Field dialog

This dialog is entered by checking any box in the Configure Form dialog. You use it to specify the type and descriptive text for a flex field. When entered, the type is always String regardless of the previous type. You will have to re-select the proper type.

Field Type

This combo box offers the types ***String***, ***Date***, ***Date-Time***, ***Choice List*** and ***Check Box***. The first three types are string fields, but Date/Time fields are validated on entry. The Choice List type provides a choice from a list you define, and a check box provides a yes/no or true/false value. The dialog box changes as you select these types to allow you to specify the data they require.

Field name

Enter the text of the field name. As you type, the exact spacing and appearance of the field name appears in the Example box. You can cause individual letters in the name to be underlined by prefixing them with '**&**'.

Edit length

Only available if the type is ***String***. Enter the maximum number of characters you are allowing for the field. The maximum value possible is 63, which is the default.

Choices

Only available if the type is ***Choice List***. This is a list of the choices you define. Edit existing choices by double clicking them, or enter new choices with the **New** button.

New

This button pops up the Choice Name dialog to allow you to enter a new choice.

Edit

This button pops up the Choice Name dialog to edit the selected **Choice**.

Delete

This button deletes the selected **Choice**.

Up

This button moves the selected **Choice** up one line.

Down

This button moves the selected **Choice** down one line.

Example

This box shows the field as it will appear. It is functional, but no changes made to it are preserved.

Save

Save the changes you made.

Choice Name dialog

The Configure Field dialog New and Edit buttons pop up this dialog. Enter or change the choice string.

Attach Files dialog

The Configure Field dialog Attach button pops up this dialog. It allows you to attach new files or retrieve existing files from the report.

Attached Files

This list shows all the currently attached files.

Add

This pops up the File Open dialog where you choose the file to attach. Provided the file you choose is not already attached, you are reminded that the file is irrevocably attached to the report. If you answer yes to the reminder, the file is attached, and added to the list.

Copy

This button is only available if there are entries in the list. It copies the selected file to a directory with the same name as the Project ID below the directory containing CLINTW.EXE.

View Log dialog

The Configure Field dialog Log button pops up this dialog. It displays all changes to the report, who made them, and when. Log entries can't be changed or deleted, and are printed in full when you print a report.

Sort Setup dialog

The Query Window | Sort command or Intray Window | Sort command pops up this dialog. You use it to specify the names and order of the fields in the sort.

Sort Setup

This shows the current setup description text. You can select other available stored setups.

Fields

This is a list of all fields in the sub project database records. Double clicking on any entry copies the field name to the **Sort Fields** list.

>>

This button copies the currently selected **Field** entry to the **Sort Fields** list.

<<

This button deletes the currently selected **Sort Field** entry.

Sort Fields

This is a list of the field headings in the order they will be used to sort the window content. You can change the order with the **Up** and **Down** buttons. Double clicking any entry deletes it.

Save Sort Setup

This button pops up the Save Setup dialog to allow you to name a sort.

Up

This button moves the currently selected **Display Field** entry up one place.

Down

This button moves the currently selected **Display Field** entry down one place.

Descending

When checked, the currently selected **Sort Field** will be sorted in descending order. The default is ascending.

Sort

Runs the sort.

Save Setup dialog

This dialog is called from any dialog which needs to name and save a setup.

Description

Choose an existing description, or type a new one.

Save

Saves the setup in the specified description.

Cancel

Dismisses the dialog without saving.

Drag/Drop from the File Manager

A file can be opened into an editor in the application by dragging the file from the File Manager, and dropping it on CLint++'s main window. CLint++ examines the extensions of the files, and uses the appropriate editor type you have configured. Any extensions it does not recognize are opened in a default Text editor window.

If the current active window is a Lint File Window, you will be asked if the files dropped should be added to the lint file, or opened as files.

Printing

There are two commands on the File menu which support printing of documents from the application. File | Page Setup is used to specify your page configuration. File | Print causes the current document to be printed.

can't use 'void' value

You have attempted to use directly or indirectly a void value. Void expressions don't have values.

# operator should be followed by an argument name

The **#** stringify operator must be followed by one of your macro's argument names.

#elif after #else at line NNN

You have an **#elif** following a **#else** which is at line NNN. **#elif** may only occur following **#if**, **#ifdef**, **#ifndef** or **#elif**.

#elif not within a conditional

You have a **#elif** not within a conditional. **#elif** may only occur following **#if**, **#ifdef**, **#ifndef** or **#elif**.

comma operator not allowed in #if

You have used the ',' operator in a #if expression, like this:

```
            #if 1,2 || 3
```

This is not allowed.

#else after #else at line NNN

You have an **#else** following a **#else** which is at line NNN. **#else** may only occur following a **#if**, **#ifdef**, **#ifndef** or **#elif**.

#else not within a conditional

You have a **#else** not within a conditional. **#else** may only occur following a **#if**, **#ifdef**, **#ifndef** or **#elif**.

#endif not in conditional

You have a **#endif** not within a conditional. **#endif** may only occur following a **#if**, **#ifdef**, **#ifndef**, **#elif** or **#else**.

#error <string>

A **#error** directive was encountered. This is used to halt compilation and print a message. **CLint++** treats it the same way.

#include: filename must be quoted

You must enclose filenames to be included with either double quotes or angle brackets like this:

```
            #include "file.h"
```

or

```
            #include <file.h>
```

The only exception is if you have define a macro containing the include name like this:

```
            #define header    "myhdr.h"
            #include header
```

#line filename must be quoted

You must enclose the filename in double quotes. The syntax for **#line** is:

```
#line 10 "file.c"
```

#line number expected

A number is required for the **#line** directive. The syntax for **#line** is:

```
#line 10 "file.c"
```

#line numbers must be >= 1

Only positive integers greater than zero will be accepted. The syntax for **#line** is:

```
#line 10 "file.c"
```

#line syntax

Meaningless **#line** statement. The syntax for **#line** is:

```
#line 10 "file.c"
```

#undef: '<string>' is not a macro

You have tried to **#undef** a name which is reserved or a built-in macro, such as **__TIME__**, **__DATE__**, **__LINE__**, or **__FILE__**.

#ifdef with no argument
#ifndef with no argument
#undef with no argument

A single identifier must follow these directives.

#ifdef argument starts with digit
#ifndef argument starts with digit
#undef argument starts with digit

Identifiers may only start with **a-z**, **A-Z** or underscore (**_**).

<filename> is multiply included

The given file has been **#include**d directly or indirectly more than once. This is usually caused by header files including other header files. Including a file more than once will slow your compiler down, and possibly lead to multiple declaration problems. You should protect files against multiple inclusion by surrounding each **#include** directive that occurs in a header file with a test like this:

```
#ifndef __MYHEADER_H
#include "myheader.h"
#endif
```

Then in 'myheader.h' you do:

```
#ifndef __MYHEADER_H
#define __MYHEADER_H

...... body of include file

#endif
```

This will prevent such problems completely. It is the method used by most compiler vendors too. You can control these reports specifically with the ***Multiply included files*** checkbox in the Options | Lint Options | General dialog (or **`-winc`** for the command line program).

'<identifier>' undefined

CLint++ has seen a reference to the given identifier preceding any declaration of it. You must declare all identifiers before use.

'<identifier>' has a different declaration in file(line)

When all files have been parsed, **CLint++** checks that all symbols have consistent declarations in all lint files.

'<identifier>' has a duplicate definition in file(line)

When all files have been parsed, **CLint++** checks that each symbol is defined exactly once as required by the **ANSI** standard.

'<identifier>' hides a library symbol in file

When all files have been parsed, **CLint++** checks to see if any of your symbols has the same name as a symbol defined in any of the libraries you specified. Although it is legal to redefine a library symbol, **CLint++** considers it to be poor practice. You can suppress this warning from the Options | Lint Options dialog box. You can control these reports specifically with the ***Report redefined library symbols*** checkbox in the Options | Lint Options | Style dialog (or **`-wlib`** for the command line program).

'<identifier>' is assigned to but never used

When all files have been parsed, **CLint++** checks for symbols defined as global and assigned to but never referenced. Such symbols waste space. When checking part of a program (e.g. a library), you can suppress this message in the Lint Options dialog box. You can control these reports specifically with the ***Report symbols defined but never used*** checkbox in the Options | Lint Options | Style dialog (or **`-wnus`** for the command line program).

'<identifier>' assigned to but not used

It is wasteful and, possibly, a bug to have variables who's value is never used. This may indicate that in using **#if** conditionals that some logic is not correct. You can control these reports specifically with the ***Report symbols defined but never used*** checkbox in the Options | Lint Options | Style dialog (or **`-wnus`** for the command line program).

'<identifier>' declared but not used

This simply wastes space. You should remove the declaration. You can control these reports specifically with the ***Report symbols defined but never used*** checkbox in the Options | Lint Options | Style dialog (or **`-wnus`** for the command line program).

label '<identifier>' previously defined at file(NNN)

You have used a label which has the same name as some other identifier. This is not an **ANSI** error but using the same name for a variable and a label must be considered bad practice.

'<identifier>' previously initialized by file(NNN)

You have initialized the given object more than once. Possibly the name was miss-typed on the second occasion.

'<identifier>' should be static

When all files have been parsed, **CLint++** checks for symbols defined as global but used in only the module in which they are defined. Such symbols should be declared **static** to avoid accidental reference to them in later revisions. When checking part of a program (e.g. a library), you can suppress this message in the Options | Lint Options dialog box. You can control these reports specifically with the ***Suggest symbols which should be static*** checkbox in the Options | Lint Options | Style Warnings dialog (or **`-wsgs`** for the command line program).

'<identifier>' is never used

When all files have been parsed, **CLint++** checks for symbols you defined but did not use. Such symbols are wasting space in your program, and should be removed. When checking part of a program (e.g. a library), you can suppress this message in the Options | Lint Options dialog box. You can control these reports specifically with the ***Report symbols defined but never used*** checkbox in the Options | Lint Options | Style Warnings dialog (or **`-wnus`** for the command line program).

'<identifier>' is used but never defined

When all files have been parsed, **CLint++** checks for symbols you used but did not define, and could not be found in any **CLint++** library you specified. Perhaps you forgot to specify a library, or to define the symbol. This is exactly equivalent to the linker's message about undefined symbols. When checking part of a program (e.g. a library), you can control these reports specifically with the ***Project is a library*** checkbox in the Lint | Edit Lint File dialog also available by right-clicking in the Lint File Window and choosing Edit, (or **-u** for the command line program).

'<identifier>' not an argument

You are using the old-style C function declaration, and have declared an argument not present in the parameter list.

The following example would report 'd' as not being an argument.

```
void f(a, b, c)
int a, b, d;
{
.....
}
```

If you are using a pre-**ANSI** compiler, you have our condolences. If not, switch to using prototypes. **CLint++** can help by generating prototypes for you.

'<identifier>' possibly used before definition

You may be using a variable before having assigned a value to it. If so, this is a serious error. However, this can arise when using **for** loops like this:

```
char *p, *q;
for (p = "1234"; *p; p = q)
q = p + 1;
```

which complains that **q** is possibly used before definition, although we can see that is assigned to before its first use. You can suppress this type of warning by initializing the variable.

'<identifier>' is not a label

You have used an identifier as the target of a **goto**, but it is not a label.

'<identifier>' is not a member of '<identifier>'

You have used an identifier as though it is a member name (i.e. after '.' or '->') following a known structure name, but the identifier is not a member of that structure.

'<library>' in old style CLint++ file format - ignored

The library format changed from 1.XX/2.XX to the current version of **CLint++**. This is because the extended checking in the new version needs more information than was recorded in the older versions. When **CLint++** sees an old-style library it issues this warning. You should rebuild the library using this version.

'~' is not a binary operator
'!' is not a binary operator

These are the only two operators than can't be used as binary operators.

'.' must be followed by a member name
'->' must be followed by a member name

Whatever follows these operators is not a member name.

'{' required to initialize aggregate

The **ANSI** standard requires that initialization of aggregates must have the initializer list enclosed in braces.

'/*' in comment started in line NNN

This may not be a problem, but the **ANSI** standard does not support nested comments. If you comment out a piece of code as follows you will get the above message.

```
            /* A=1 /* Temporary change */
```

CLint++ also supports the C++ style of comments, so you can prevent this warning when it is intentional like this:

```
            // A = 1 /* Temporary change */
```

Additionally, for compatibility with Borland and other compiler vendors, you can control these reports specifically with the Options | Lint Options | ***Support Nested C comments*** (or `-C` for the command line program).

-ve case with unsigned switch expr

You should not use negative case values with unsigned **switch** expressions (even though they might work). Possibly the type of the expression should be re-examined.

too many cases (> 8000)

Clint can't handle more than 8000 case labels in a single switch. We would love to see your code.

address of automatic returned

A deadly trap with functions which return pointers is returning the address of an **auto** variable. Since the pointer refers to storage on the stack which is no longer allocated, it really points to junk. You must always fix this.

arguments given to macro '<identifier>'

You have tried to call a macro with arguments, but it does not take any. Only the parentheses are required.

ambiguous operators should have parentheses

The operators concerned are '**&&**', '**||**'. '**<<**', '**>>**', '**&**', '**|**' and '**^**'. The precedence of these operators is often confused, and you should use parentheses to make your intentions explicit.

argument '<identifier>' unused

You have declared a function argument which is never used. If the argument is unused intentionally, you can silence **CLint++** by preceding the function definition with the line:

```
#pragma argsused
```

or

```
/*ARGSUSED*/
```

This is also effective with most modern compilers. You can control these reports specifically with the ***Report symbols defined but never used*** checkbox in the Options | Lint Options | Style dialog (or **`-wnus`** for the command line program).

'asm' or '_asm' not followed by operands

The syntax of **asm** is one of:

```
asm assembly
```

or

```
asm {
assembly
assembly
}
```

or

```
asm("assembly");
```

CLint++ saw something other than one of these. The last form may be used as though it was a function call in an expression, in which case it is assumed to return an **int** value. Note that **CLint++** makes no attempt to parse the assembly language text!

can't modify 'const' object

A constant object may not be modified. The value of a **const** object is usually established at compile time, unless it is a non const object passed to a function as a **const** (to prevent that function from modifying it).

at most one 'default' allowed

Extra defaults have no meaning. Check your **switch** braces.

illegal function storage class

Functions may not be: **auto**, **register** or **typedef**.

badly punctuated parameter list in #define

When you define a macro with parameters, the list of parameters must begin with `'('` followed by a comma separated list of parameters, followed by `')'`. The parameter list seen was not like this.

body is <N> lines long

This function is longer than the current warning level for function size. The default is 100 lines. You can control these reports specifically with the Options | Lint Options | Style | ***Function body larger than NNN lines*** (or **`-wlfbNNN`** for the command line program).

both return and return of a value

This is a warning of potentially disastrous results, as returned values will be indeterminate! In C++, it is illegal. You can control these reports specifically with the ***Return and return of a value*** checkbox in the Options | Lint Options | General dialog (or **`-wret`** for the command line program).

'break' not in loop or switch

You may only use **break** inside a loop or **switch**. Elsewhere it is meaningless. This could possibly be caused by a bracing error.

call of <identifier> with no prototype

You have called a function for which no prototype declaration has been seen. You may either have failed to include a library header file, or not provided prototype declarations in your local header files. In this last case, **CLint++** can help you generate prototypes. You can control these reports specifically with the Options **|** Lint Options **|** General **|** ***Function must be prototyped*** (or `-wpro` for the command line program).

call of non-function

You have tried to call something which is not a function. For example:

```
void f()
{
int i;

i();
}
```

This will probably be a typing error, or you may have misplaced parentheses inside a complex expression.

can't cast to function or array

You may not perform these types of cast as they are meaningless.

can't cast to union or structure

You may not perform these types of cast as they are meaningless.

can't find file "<filename>"

The include file specified could not be found. Check your Options | Lint Options dialog box include paths, or spelling (or `-I` for the command line program)..

can't find library "<filename>"

The library file specified could not be found. Check your Options | Lint Options dialog box include paths, or spelling (or `-l` for the command line program)..

can't have array of function

You can't have arrays of functions - only arrays of pointers to functions.

can't take address of register

In C, register variables cannot have addresses because, once declared as **register**, a variable is assumed to be in a machine register (even if a one is not available for it, and memory storage is used). C++ allows you to take the address of **register** variables.

unreachable case <N>

This occurs when you use a shorter integer **switch** expression than the **case** values.

The following example provokes this warning.

```
char c = getch();

switch (c) {
case 'a': .........

case '128': ........
}
```

You can control these reports specifically with the Options | Lint Options | General | ***Unreachable code*** (or **`-wunr`** for the command line program).

'case' not in switch

A **case** has been found which is not inside a **switch** statement. This is possibly a bracing error.

code has no effect

The expression does nothing! Maybe something like this:

```
a == 1;
```

where you may have meant:

```
a = 1;
```

conflicting pointer modifiers

You have used more than one of the type modifiers **near**, **far**, **huge**, **_ss**, **_cs**, or **_handle**. At most one of these modifiers may occur adjacently.

conflicting storage classes

You may not have more than one storage class specifier. For example, the following lines will not compile as they are self- contradictory:

```
static auto int i;
static extern char j;
```

constant assigned to 'enum <enumname>'
constant compared with 'enum <enumname>'
constant used with 'enum <enumname>'

You have assigned, compared, or otherwise used a constant and an enumerated type. If this is intentional, replace the constant with an enumerated constant, or cast the constant to the enumerated type. You can control these reports specifically with the Options | Lint Options | Type | ***Report enumeration misuse*** (or **`-wenu`** for the command line program).

constant 'switch' expression

This message warns that in this case the **switch** expression is constant and therefore, at most, one case will ever be executed. A likely cause of this problem could be the result of an expanded macro or a typing error.

Incidentally, if you ever think a macro is causing unexpected results, **CLint++** can preprocess the source for you into an editor buffer so you can see exactly what happened. You can control these reports specifically with the Options | Lint Options | General | ***Constant condition in if/while*** (or **`-wcon`** for the command line program).

constant 'if' condition
constant 'while' condition
constant 'for' condition
constant 'do' condition

Constant conditions are assumed to be errors unless you have used the following types of statement, which are considered to be common idiom:

```
while (1) {
.....
}

do {
.....
} while (1);
```

This message may be accompanied by complaints about constant unsigned comparisons, which is the real cause of the problem. You can control these reports specifically with the Options | Lint Options | General | ***Constant condition in if/while*** (or **`-wcon`** for the command line program).

constant is long

You should add the suffix **L** to a constant number. Alternatively, if the constant is in the range 32768 through 65535, and being passed or assigned to an unsigned value, you should use the **U** suffix. You can tell **CLint++** to support 32 bit integers from the Options | Lint Options dialog box, in which case these warnings won't occur. You can control these reports specifically with the Options | Lint Options | Type | ***Constant is long (requires 'L' suffix)*** (or **`-wcil`** for the command line program).

constant required

In several places, such as array declarations and case values, constant expressions are required. For example:

```
int i[5];

int i[foo];
```

In the second case 'foo' should be a macro with a constant value, enumeration constant, or **const** object. Also:

```
case foo: .....
```

will not work, and won't be allowed by your compiler anyway. There are other places where this may occur, such as during initialization.

constant unsigned comparison

This is probably a bug, due to assuming that the **unsigned** value is actually **signed**. For example:

```
unsigned int i;

if (i < 0) {
.....
}
```

where the condition is always false, or the following, where the condition is always true:

```
unsigned int i;

if (i >= 0) {
.....
}
```

You can control these reports specifically with the Options | Lint Options | General | ***Constant condition in if/while*** (or **`-wcon`** for the command line program).

'continue' not in loop

A **continue** has been found which was not inside a loop. Check for a possible bracing error.

conversion may lose significant digits

You have tried to assign a value to a variable which is greater than that variable's possible range.

```
char c = 3;
int i = 256;

c = i;
```

In this case 'c' would end up with a value of zero!

By default, only assignments of longs to ints, shorts or chars will be detected by **CLint++**. When enabled, **CLint++** it will also detect assignments of **long double** to **double** or **float**, and **double** to **float**. If you want to explicitly make the conversion, use a cast. You can control these reports specifically with the Options | Lint Options | General | ***Report loss of significance*** or ***Report strict loss of significance*** (or **`-wsig`** or **`-wsig+`** for the command line program).

declaration syntax

CLint++ encountered a declaration which was impossible to make sense of, such as:

```
int 0;
```

'defined' must be followed by ident or (ident)

defined must be followed by an identifier or parenthesized identifier.

'default' not in switch

A **default** was encountered which was not inside a switch statement. Check your code for possible bracing errors.

degenerate unsigned comparison

This is probably a bug, due to assuming that the unsigned value is actually signed or may have a negative value. For example:

```
unsigned int j;

while (j > 0) {
....
}
```

this code actually means:

```
unsigned int j;

while (j != 0) {
....
}
```

which is much clearer and easier for other programmers to understand, and yourself when you refer to it later. You can control these reports specifically with the Options | Lint Options | General | ***Constant condition in if/while*** (or **`-wcon`** for the command line program).

different enum types in assignment
different enum types in conditional
different enum types in comparison

You have assigned one enumerated type to a different enumerated type. You can control these reports specifically with the Options | Lint Options | Type | ***Report enumeration misuse*** (or **`-wenu`** for the command line program).

division by zero

An expression was found which would divide a constant by zero at compile time. Possible causes could be a typing error or the result of a macro expansion.

does not have address

You have tried to take the address of something which does not have one, such as a number. Check for macro expansions or typing mistakes.

duplicate case (NNNN)

The indicated case label has the same numeric value (NNNN) as an earlier case. All case labels must be unique. This may arise from macro expansion, or from undefined symbols.

'else' not after 'if'

An **else** was found which did not immediately follow the end of an **if** statement, like this:

```
if (foo)
xxxx();
yyyy();
else
zzzz();
```

In the above example you probably forgot the braces required between the **if** and the **else**, although the indentation makes your intention clear. We recommend you always use braces after **if**, **while**, **do** and so on to avoid this problem.

empty body of 'do' loop
empty body of 'while' loop
empty body of 'for' loop

Check for bracing errors, or for accidental (or otherwise) commenting out of the code. You can control these reports specifically with the Options | Lint Options | Style | ***Empty body of loop or condition*** (or **`-welb`** for the command line program).

empty else clause of 'if'
empty then clause of 'if'

Check for bracing errors, or for accidental (or otherwise) commenting out of the code. In any case, the resulting code may be wasteful and inefficient. You can control these reports specifically with the Options | Lint Options | Style | ***Empty body of loop or condition*** (or **`-welb`** for the command line program).

end of file in macro call

A '\' (backslash) may have been found on the last line of a file while a macro call was in progress, or you may have a mismatched parenthesis inside a macro call, in which case it will always be a missing ')' which causes this problem.

enumerator '<identifier>' not in 'enum <identifier>'

You have an enumerated constant from a different enumeration than the one to which you are assigning. Here is a specific instance:

```
enum x {a, b};
enum y {c, d};

enum x i = c;
```

You can control these reports specifically with the Options | Lint Options | Type | ***Report enumeration misuse*** (or **`-wenu`** for the command line program).

evaluation order of '<operator>' undefined

In the following example, it is undefined whether 'a[2]' or 'a[3]' will be assigned to, as it is up to compiler writers whether they increment 'i' before or after calculating the address of 'a[i]'.

```
int a[5], i=2;

a[i]=++i;
```

Always break statement like this into two or more statements. The operator may be any which has a side effect.

expression syntax

CLint++ could not make sense of the expression.

expression too complex

There are too may parentheses or arguments in the expression. **CLint++** can't handle more than 40 (!) levels of parentheses.

bitfield too long for type

CLint++ assumes that an **int** takes up 2 bytes (16 bits), therefore no field may be larger than 16. (Incidentally the **ANSI** standard warns that the use of bit fields can lead to all manner of portability problems, as every aspect of bit fields is left to the compiler writer!). You can control these reports specifically with the ***Long (>16) field declared*** checkbox in the Options | Lint Options | General dialog (or **`-wlnf`** for the command line program).

function requires compound statement

Possibly a prototype has a missing ';' and is therefore assumed to be a function declaration, or you could have missed out the first '{'.

identifier expected

A declaration was expecting an identifier to declare. This is most likely to be a typing error such as the following:

```
int ,j;
```

illegal void type for '<identifier>'

You may not declare any objects as type **void**, like this:

```
void i;
```

perhaps you were intending to type one of the following:

```
void *i;
void i();
```

illegal character '<char>' [<hex value>]

This character cannot be part of any C identifier or token. Look for a missing '"' before this character, as you probably meant it to be part of a string.

illegal conditional, ':' missing

This is probably a typing mistake like the following:

```
        (x)? y, z;
```

where you intended to put a colon in place of the comma.

illegal indirection

You have tried to use an object that is not a pointer, either by subscripting or direct use of the '*' operator, like this:

```
int i;
j = *i;
```

illegal dynamic initialization

In C, you have attempted to initialize a variable outside the body of any function with an expression. This is not possible as code would be needed, but which function could contain the code? In C++ this is possible, and there are very few restrictions on initialization.

illegal structure operation

In C, the only operations allowed on structures are accessing a member with '.' or '->' and assigning structures of the same type to each other. In C++, you can use **operator** functions and constructors to support almost any operation on structures or classes.

external '<identifier>' declaration differs from file(NNN)

A symbol declared **extern** in a function block differs from the same name outside the block. For example,

```
extern int i;

void f()
{
extern char i;
}
```

illegal type of operand

The bit operators '<<', '>>', '&', '|', and '|' can't be applied to operands of type **float**, **double**, or **long double**.

invalid pointer operation

The only things you can do with pointers are add or subtract integers to them, subtract or compare two pointers of the same type or indirect them using '[]' or '*'. Perhaps you may have added an 'int *' and a 'char *' or any other type of mismatch.

incompatible structure assignment

You have tried to assign a structure of one type to a different one. In a structure assignment both *must* have the same type. In C++ this rule is relaxed if an appropriate **operator=** or copy constructors are defined.

initialization of auto aggregate

An 'aggregate' is any structure, union or array, that is: a collection of more primitive objects. All compilers use code to initialize **auto** objects. You can save code and speed up your program by declaring the object **static** if you don't intend to modify it. You can control these reports specifically with the Options | Lint Options | General | ***Automatic array/struct initialized*** (or **`-wiag`** for the command line program).

initializing undefined structure

You have tried to initialize a structure before it has been declared.

invalid macro name '<identifier>' in #define

The name you have tried to use for the macro is not legal. Macro names must begin with a letter, digit, or underscore.

label '<identifier>' not used

The label is not used in your function. You should remove it.

lvalue required

An 'lvalue' is an expression which can be modified by an assignment. You have tried to assign or modify something which is not an 'lvalue'. For example:

```
0 = a;
```

macro calls nested too deep

You may not nest macro calls more than 800 (!) deep.

missing ')' after 'if' condition
missing ')' after 'while' condition
missing ')' after 'do' condition

Generally these will be typing errors, we have tried to be as helpful as possible in diagnosing them. You will normally find it obvious what the problem is.

missing '(' after 'if'
missing '(' after 'while'
missing '(' after 'switch'

Generally this is a typing error. You will normally find it obvious what the problem is.

')' not seen after '...'

The ellipsis operator (**...**) may only occur as the last argument of a prototype. It indicates that an unspecified number of arguments of unknown type my occur. You used it other than as the last argument of a prototype.

missing ':'

Generally this is a typing error. You will normally find it obvious what the problem is.

missing label name after 'goto'

Generally this is a typing error. You will normally find it obvious what the problem is.

missing '}'

Generally this is a typing error. You will normally find it obvious what the problem is.

missing ';'

Generally this is a typing error. You will normally find it obvious what the problem is.

missing ')' in function call

Generally this is a typing error. You will normally find it obvious what the problem is.

missing '<character>'

Generally this is a typing error. You will normally find it obvious what the problem is.

missing operand

Generally this is a typing error. You will normally find it obvious what the problem is.

missing ')'

Generally this is a typing error. You will normally find it obvious what the problem is.

missing operator

Generally this is a typing error. You will normally find it obvious what the problem is.

missing ']'

Generally this is a typing error. You will normally find it obvious what the problem is.

missing ';' in 'for'
missing ')' in 'for'

Generally these will be typing errors. You will normally find it obvious what the problem is.

missing macro name in #define

The syntax of **#define** is:

```
            #define macro_name replacement
```

or

```
            #define macro_name(args) replacement
```

missing argument type

CLint++ was reading a function definition, and found a function argument which had no type. Supply the missing type.

mismatched filename quotes in #include

You have used both angle bracket and double quotes, like this:

```
#include "filename>
```

or

```
#include <filename"
```

misuse of 'sizeof'

sizeof requires a following type declarator or object name, or you may have used it as a binary operator.

mixing pointers to different 'char' types

You are using a pointer to 'char *' and a pointer to 'unsigned char *' in the same expression. Although this will almost always work, you may need to consider using casts to make your intentions clear. You can control these reports specifically with the Options | Lint Options | Type | ***Mixing pointers to signed/unsigned char*** (or **`-wmxp`** for the command line program).

conflicting calling conventions

You have mixed **fortran**, **pascal**, or **fastcall** in the function declaration. At most one of these may be used.

newline in string
newline in character constant

The **ANSI** standard requires that strings and character constants are terminated prior to the end of a line. You probably have a missing quote. If you were trying to get an end of line in single quotes, use '\n'. To continue strings, use either '\' prior to the end of a line, or terminate the current string and begin again on the next line.

no arguments to macro '<identifier>'

You have called a macro with only parentheses, but it requires arguments.

initializer only partly braced

The initializer expression is not braced in accordance with **ANSI** recommendations. You can control these reports specifically with the Options | Lint Options | General | ***Non-ANSI initializer bracing*** (or **`-wbbr`** for the command line program).

non-portable pointer conversion

This means that you have used a non pointer value expression in a context where a pointer is required. For example:

```
            char *p=0x123456;
```

In these cases cast the constant to the proper type. Although not casting may appear to produce working code it is not guaranteed and is exceedingly unlikely to be portable.

empty subscript not first

You are initializing an array where a dimension (other than the first) is missing. For example:

```
char c[][] = {"ABC", "DEF", "GHI", "JKL"};
```

Remember, only the first subscript may be omitted, all subsequent subscripts *must* be supplied.

only 1 argument to macro '<identifier>'

You have called the macro with one argument, but it requires more.

only N arguments to macro '<identifier>'

You have called the macro with N arguments, but it requires more.

parameter name starts with a digit in #define

Parameters in **#define**s must be identifiers and therefore may not begin with a digit.

pointer operands of '-' must have the same type

You may only subtract pointers of the same type.

pointer assignment allows const violation

CLint++ takes great care to check that const objects are not assigned to. It has noticed your program assigning to a pointer which does not have the **const** modifier a pointer to a const object possibly like this:

```
const int i = 5;
int *ip = &i;
```

If this assignment was allowed, you could write to 'i' by dereferencing 'ip'.

possible unintended assignment

CLint++ considers assignments occurring in conditions to be poor practice. You can control these reports specifically with the Options | Lint Options | Style | ***Report possible unintended assignments*** (or **`-wpua`** for the command line program).

printf: unknown conversion %<char>

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Consult your compiler's documentation for the **printf()** function. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

printf: non-ANSI size specifier '<specifier>'

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. You have used the non-ANSI 'N' or 'F' specifiers. You can control these reports specifically with the Options | Lint Options | General | ***Non-ANSI printf size specifiers*** (or **`-wnac`** for the command line program).

printf: more conversions than arguments

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Supply the missing arguments, or remove the surplus conversions. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: found <N> unused arguments

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. There are more arguments than conversions. Remove the surplus arguments, or add more conversions. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: format string argument missing

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. You must supply a format string to these functions. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

printf: 'short' argument to %<specifier> - use %h<specifier>

The **ANSI** standard requires that when using a short argument, the **%h** form of the conversion must be specified. Many programs which omit this work well, but are not portable. Add the 'h' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: 'long' argument to %<specifier> - use %l<specifier>

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Add the 'l' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: 'long long' argument to %<specifier> - use %L<specifier>

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Add the 'L' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: 'long double' argument to %<specifier> - use %L<specifier>

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Add the 'L' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: '*' requires 'int' argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Supply an int argument, or change the precision specifier to a constant. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

printf: %<specifier> requires 'double' argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Supply a **double** argument, or change the specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

printf: %<specifier> requires integral argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Supply an integral argument, or change the conversion. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: %s requires 'char *' argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Supply a **char *** argument. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

printf: %p requires pointer argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Supply a pointer argument. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: %n requires 'int *' argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Supply a pointer argument to an integral type. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: %L<specifier> requires 'long double' argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either remove the 'L' specifier, or change the argument type. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: %l<specifier> requires 'long' argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either remove the 'l' specifier, or change the argument type. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: %L<specifier> requires 'long long' argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either remove the 'L' specifier, or change the argument type. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: %h<specifier> requires 'short' argument

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either remove the 'h' specifier, or change the argument type. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: %<specifier> is an illegal conversion

printf() and its companion functions (**sprintf()** and **fprintf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. The conversion is illegal, and must be changed to a legal one. Consult your compiler's reference on **printf()**. You only get this warning if you enable Options **|** Lint Options **|** General **|** ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

redeclaration of '<identifier>' previously seen at line(file)

Once declared you may not redeclare any object.

redefinition of '<identifier>' as tag previously seen in file(NNN)

Once declared you may not redeclare any object.

redefinition of '<macro>' not identical

While the **ANSI** standard allows you to define macros as many times as required, each definition must be functionally equivalent (i.e. differing in white space only) to all the others. **CLint++** has seen a redefinition which is not equivalent.

'register' used

Modern compilers allocate registers very efficiently. You should not use **register** unless you are very familiar with the compiler's code generation strategy. You can control these reports specifically with the ***Use of 'register'*** checkbox in the Options | Lint Options | Style dialog (or **`-wreg`** for the command line program).

runtime division by zero

When you run this program, this expression will attempt to divide by zero. Did you really mean that?

statement after 'if' should have '{'
statement after 'else' should have '{'
statement after 'do' should have '{'
statement after 'while' should have '{'
statement after 'for' should have '{'

Mainly a matter of style. However, always using the brace pairs (even when not necessary) can avoid errors. You can control these reports specifically with the Options | Lint Options | Style | ***'{' advised after 'for'*** or ***'{' advised after 'else'*** or ***'{' advised after 'while'*** or ***'{' advised after 'if'*** (or **`-wfbr`**, **`-webr`**, **`-wwbr`**, **`-wibr`** for the command line program).

static '<identifier>' assigned to but not used

This wastes space in your program. You can control these reports specifically with the Options **|** Lint Options **|** Style **|** ***Report symbols defined but not used*** (or **`-wnus`** for the command line program).

static '<identifier>' declared but not used

This wastes space in your program. As the object is static, it is not visible outside the file containing the definition, and therefore can't ever be referenced. You can control these reports specifically with the Options | Lint Options | Style | ***Report symbols defined but not used*** (or **`-wnus`** for the command line program).

struct/union passed by value

Mainly a matter of efficiency, as this type of code is slow and could unnecessarily use a lot of stack space. Also, some earlier compilers failed to handle recursive functions which return structure values. In any case, it is never necessary to use this as you can always achieve the same effect using pointers. In C++, use references. You can control these reports specifically with the Options | Lint Options | General | ***Structure passed by value*** (or **`-wstv`** for the command line program).

struct/union value returned

Mainly a matter of efficiency, as this type of code is slow and could unnecessarily use a lot of stack space. Also, some earlier compilers failed to handle recursive functions which return structure values. In any case, it is never necessary to use this as you can always achieve the same effect using pointers. In C++, use references. You can control these reports specifically with the Options | Lint Options | General | ***Structure passed by value*** (or **`-wstv`** for the command line program).

suspicious pointer conversion

You have compared or otherwise used pointers to two different types. You may need to add a cast, or change the program design.

syntax error

It was too hard for **CLint++** to provide a specific diagnostic.

syntax error in 'asm'

The syntax of asm is one of:

```
asm assembly

asm {
      assembly
      assembly
}

asm("assembly");
```

text after #include filename

'#include' must be only followed by a single quoted filename. Comment out or remove the text.

text after #else
text after #endif

#else and **#endif** may not be followed by anything other than a comment. Comment out or remove the text.

text after #ifdef argument
text after #ifndef argument

#ifdef and **#ifndef** must be followed by a single identifier. Comment out or remove the text.

too few arguments in function call

There are fewer arguments in the call of the pointer to function than declared in the prototype. Supply the missing arguments.

too few arguments in call of '<function>'

There are fewer arguments in the call of the function than declared in the function's prototype. Supply the missing arguments.

too many '}'s

Almost always a typing error. Check your braces.

too many arguments in call of '<function>'

There are more arguments in the call of the function than declared in the function's prototype. Remove the surplus arguments.

too many arguments in function call

There are more arguments in the call of the pointer to function than declared in the prototype. Remove the surplus arguments.

too many (N) arguments to macro '<identifier>'

You have supplied more arguments in the macro call than it requires. Remove the surplus arguments, or change the definition.

too many initializers for '<identifier>'

You declared an array or struct but supplied more initializers than it has elements or members.

too many structure initializers

In a multiply braced initialization you supplied more initializers in an inner-braced pair than there are structure members.

trigraph '??='
trigraph '??/'
trigraph '??''
trigraph '??('
trigraph '??)'
trigraph '??!'
trigraph '??<'
trigraph '??>'
trigraph '??-'

If you get this you may be unaware that the above sequences are actually reserved by the **ANSI** standard. We only mention this as some compilers may act on them. To find out more about trigraphs read the second edition of "*The White Book*". (K&R). You can control these reports specifically with the Options | Lint Options | Style | ***Report ANSI trigraphs*** (or **`-wtri`** for the command line program).

typedefed a pointer type as '<identifier>'

Issues of style and clarity. You can control these reports specifically with the Options | Lint Options | Type | ***Pointer type was typedefed*** (or `-wtyp` for the command line program).

typedefed <type> as '<identifier>', use <identifier>

Issues of style and clarity. You can control these reports specifically with the Options | Lint Options | Type | ***struct typedefed in lowercase*** or ***struct typedefs must be uppercase*** (or **`-wtys`** or **`-wtysU`** for the command line program).

typedefed enum as '<identifier>'

Issues of style and clarity. You can control these reports specifically with the Options | Lint Options | Type | ***'enum' was typedefed*** (or **`-wtye`** for the command line program).

questionable typedef as '<identifier>'

Issues of style and clarity. You can control these reports specifically with the Options | Lint Options | Type | ***Complex type was typedefed*** (or **`-wtyg`** for the command line program).

type mismatch in argument '<identifier>'

You have called a function with an argument type different from that declared in the function's prototype.

no type allowed in old-style argument

Old style function definitions look like this:

```
void f(a, b)
int a, b;
{
```

but **CLint++** has seen a declaration looking like this:

```
void f(a, int b)
```

which is as far as it has read. Because the first argument had no type, it assumes that it is reading an old style definition, but then it saw the type. Old style definitions are not recommended. Perhaps you omitted the type on the first argument?

undefined label '<identifier>'

CLint++ got to the end of the function and found that you hadn't declared this label, which is required because it used by a **goto**.

unexpected ']'
unexpected ')'

Typing error? Did you mean that?

unexpected end of file

You have a missing '}' at the end of the file.

unknown directive #<string>

This is not a known '#directive'.

misuse of 'based'

The based keyword is a Microsoft extension. It has been used improperly.

size of type unknown or zero

You have mentioned an object who's size is not known (but is required at this point) because you have not yet declared it. You may need to rearrange your declarations to ensure the declaration precedes this use.

unreachable code

This statement will never be executed. You can silence **CLint++** by preceding the line with:

```
#pragma notreached
```

or

```
/*NOTREACHED*/
```

which is not - unfortunately - effective with most compilers. You can also disable this warning with the ***Unreachable code*** checkbox in the Options | Lint Options | General dialog (or **`-wunr`** for the command line program).

unterminated parameter list in #define

You are defining a macro which takes arguments, but the closing parenthesis ')' was not seen before the end of the line. Check the definition carefully - if the macro spans several lines, you may need to escape the end of line with a backslash.

unterminated comment started in line NNN

You have a '/*' which is not closed before the end of the file. This is usually a mistake, and causes large amounts of code to be hidden by the comment.

unterminated #conditional started in NNN

You have a conditional '#directive' started at line NNN which remains unterminated by a **#endif** at the end of the file. Either remove the original directive, or add a **#endif**.

value assigned to 'enum <identifier>' not an enumeration

You have assigned a value (which is not an enumerated constant) to an enumeration. Like this:

```
        enum x i = 5;
```

You can suppress this warning with the ***Report enumeration misuse*** checkbox in the Options | Lint Options | Style dialog (or **`-wenu`** for the command line program).

value should be returned

This function declares that it will return a value, but at the end of the function there is no **return** statement to give this value. You can control these reports specifically with the ***Return and return of a value*** checkbox in the Options | Lint Options | General dialog (or **`-wret`** for the command line program).

void functions can't return a value

You have attempted to return a value from a function declared as **void**, that is a function declared as not returning a result.

'while' must follow 'do' body

The syntax of **do** is:

```
do {
statement
} while (condition);
```

No **while** was found after the '}'.

zero or negative array dimensions not allowed

You declared an array with an explicit zero or negative subscript, like this:

```
            char c[0];
```

which would allocate zero memory if it were legal. What you may have wanted is a flexible array without subscript, which would count the initializers to establish it's size. In which case you should use something like:

```
            char c[] = {0, 1, 2, 3, 4, 5, 6};
```

which allocates 7 bytes to the array 'c'.

scanf: 'short *' argument to %<specifier> - use %h<specifier>

The **ANSI** standard requires that when using a **short** argument, the **%h** form of the conversion must be specified. Many programs which omit this work well, but are not portable. Add the 'h' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

scanf: 'long *' argument to %<specifier> - use %l<specifier>

The **ANSI** standard requires that when using a **long** argument, the **%l** form of the conversion must be specified. Many programs which omit this work well, but are not portable. Add the 'l' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

scanf: 'long long *' argument to %<specifier> - use %L<specifier>

The standard requires that when using a **long long** argument, the **%L** form of the conversion must be specified. Many programs which omit this work well, but are not portable. Add the 'L' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

scanf: 'long double *' argument to %<specifier> - use %L<specifier>

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Add the 'L' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

scanf: %l<specifier> requires 'double *' argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either change the argument type to be **double**, or remove the 'l' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

scanf: %<specifier> requires integral * argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Change the argument type to be a pointer to an integral type. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

scanf: %s requires 'char *' argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Change the argument type to be **char ***. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

scanf: %L<specifier> requires 'long double *' argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either change the argument type to be **long double**, or remove the 'L' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

scanf: %l<specifier> requires 'long *' argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either change the argument type to be **long**, or remove the 'l' size specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

scanf: %L<specifier> requires 'long long *' argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either change the argument type to be **long long**, or remove the 'L' size specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

scanf: %h<specifier> requires 'short *' argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either change the argument type to be **short**, or remove the 'h' specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: %N requires 'near *' argument
scanf: %N requires 'near *' argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either change the argument type to be **near**, or remove the 'N' size specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or `-wprn` for the command line program).

printf: %F requires 'far *' argument
scanf: %F requires 'far *' argument

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Either change the argument type to be **far**, or remove the 'F' size specifier. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: 'near *' argument requires %N
scanf: 'near *' argument requires %N

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Add the 'N' size specifier to the format string. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

printf: 'far *' argument requires %F
scanf: 'far *' argument requires %F

scanf() and its companion functions (**fscanf()** and **sscanf()**) can be one of the most prolific sources of problems, mainly due to using incorrect '**%**' conversions. Add the 'F' size specifier to the format string. You only get this warning if you enable Options | Lint Options | General | ***Check printf-like function calls*** (or **`-wprn`** for the command line program).

struct has constant members in assignment

You have tried to assign one structure to another, but the destination structure has one or more members which are **const**, or recursively include a structure with one or more **const** members. **ANSI** forbids this.

static '<identifier>' has the same name as a global at file(NNN)

You have declared a **static** object one file with the same name as a globally visible object declared in another. This may cause confusion. You only get this warning if you enable the ***Note static names matching globals*** checkbox in the Options | Lint Options | Style dialog (or **`-wscg`** for the command line program).

printf: non-ANSI format specifier %c
scanf: non-ANSI format specifier %c

You have used **%N** or **%F** specifiers in **printf()** or **scanf()**. These are not **ANSI** specifiers. You only get this warning if you enable General | ***Non-ANSI printf size specifiers*** (or **`-wnac`** for the command line program).

array needs indirection

You have used an array without subscripting it, perhaps like this:

```
struct {
int i;
} x[5];

x.i = 1;
```

internal error at NNNN:NNNN

CLint++ committed a GP fault at the CS:IP address shown. The error is recoverable, and will not affect your use of CLint++ or Windows. Please report this error to us if it occurs to you, as we need that information improve CLint++. Please let us know the version of CLint++ it occurred with.

number syntax

Either a non-hex digit followed '0x', or '8' or '9' occurred in an octal constant.

only member functions can be virtual

You declared a non-member function 'virtual', or a static member function 'virtual'.

character constant out of range

You declared a hex or octal character constant larger than 255, or a wide character constant larger than 65535.

empty expression

You typed something like:

```
                if ()
```

The condition of **if** may not be empty, likewise **while** and **switch**.

integer overflow in #if

A constant expression overflowed during evaluation. No expression may exceed the largest long constant (0x7fffffff) at any point in its evaluation.

empty character constant

You typed a character constant which is empty.

multi-character character constant

You typed a character constant with more characters than the width (1 for ordinary constants, 2 for wide character constants).

character constant too long

You typed a character constant with more characters than allowed in the Lint Options dialog option ***Character constant length***. The command line program equivalent is **`-KcN`** where N is 1-4.

no floating constants allowed in #if

Floating constants are not permitted in **#if** or **#elif**.

no string constants allowed in #if

String constants are not permitted in **#if** or **#elif**.

\\x with no following digits

A hex constant in a string was not followed by hexadecimal digits.

'<operator>' not allowed in #if
'sizeof' not allowed in #if

The ANSI standard states that you can't use various operators, **sizeof**, or casts in #if or #elif. However, some compilers use **sizeof** in #if, and you can enable support for this with the ***Allow sizeof or cast in #if/#elif*** checkbox in the Options **|** Lint Options dialog.

extra after expression in #if

There was extra non-comment text after the constant expression in **#if** or **#elif**.

space advised after '<character>'

There is no white space after the given character. When enabled by the Options | Lint Options | Style | ***' ' advised after operator*** (or **`-wspc`** for the command line program, Clint considers as a matter of good style that white space should follow the '**,**', '**:**', '**;**', '**=**', '**==**', or '**?**' characters.

space advised after '<keyword>'

There is no white space after the given character. When enabled by the Options | Lint Options | Style | ***' ' advised after operator*** (or `-wspc` for the command line program, Clint considers as a matter of good style that white space should follow the keywords **return**, **if**, **while**, **else**, **switch**, **for**, and **do**.

storage class specifier repeated

A storage class specifier was repeated like this:

```
static static int i;
```

storage class specifier used in argument

A storage class other than **register** was used in a function argument.

'virtual' used outside any class

The qualifier **virtual** may not be used outside a class definition. Even when a virtual function is declared in a class, the definition of it outside the class may not use **virtual**. For example, the correct use is:

```
class C
{
virtual int f();
}

int C::f()
{
return 0;
}
```

Virtual is only useful in classes intended to be used as base classes which provide polymorphic methods.

'register' used outside a function

The **register** storage class is a recommendation to place a frequently used **auto** object or argument in a register. No compiler is obliged to act on this recommendation. Since **register** can be considered equivalent to **auto**, **register** may not be used outside a function prototype or body. In **C**, you can't take the address of a **register** object: in **C++** you can.

too many '<type>'

You have repeated a type name like this:

```
            int int i;
```

'<identifier>' not typed, 'int' assumed

An untyped declaration was seen. **int** is assumed for untyped objects. The draft **ANSI C++** standard calls for a warning of this, as it is intended that this feature be disallowed in a later version of the language. We warn of this in **C** also.

types seen before 'class'
types seen before 'struct'
types seen before 'union'
types seen before 'enum'

A type such as **int** was seen before one of these keywords.

storage class not needed unless a declaration

A storage class was seen before a structure declaration which was not followed by an object declaration, like this:

```
            static struct S { int I; };
```

The storage class in this case is redundant.

modifiers not needed unless a declaration

A modifier like **const** was seen before a structure declaration which was not followed by an object declaration, like this:

```
const struct S { int I; };
```

The modifier in this case is redundant.

storage class seen before '...'

A storage class such as **register** was seen before an ellipsis.

types must not occur with 'interrupt'

A type other than **void** was used with **interrupt**. Functions declared **interrupt** may not return values because they are called by hardware, not your code.

illegal type combination

A type combination like **short long** was seen. This is contradictory.

empty struct declaration
empty union declaration

A **struct**, **class**, **union**, or **enum** was seen with no tag or body like this:

```
struct;
```

tag '<identifier>' previously defined in file(NNN)

A tag was redefined as something else.

extra ',' in enum <identifier>

An extra comma was seen when reading an enumeration body like this:

```
            enum E { a, , b};
```

enumeration has no enumerators

An enumeration declaration with no enumerators was seen like this:

```
            enum E {};
```

can't use protection in unions

You can't use the protection keywords **private**, **public**, or **protected** in unions, because you can't derive classes from a union.

':' missing after protection keyword

When **public**, **private**, or **protected** are used in class or structure bodies, they must be followed by a colon like this:

```
struct S {
private:
int I;
};
```

struct has no members
class has no members
union has no members

Structures of all types must have at least one member when defined. We saw something like this:

```
        struct C { };
```

possible undeclared type name '<identifier>'

CLint++ saw an undefined name used in a position where a type name was expected like this:

```
            type *tp;
```

After producing the error, it declares the name as a typedef for **int** and proceeds with the parse. This may cause secondary errors.

extra ',' in argument list

An extra comma was seen in a function prototype like this:

```
int f(int a, , int b);
```

unions can't have static members

Unions can't have **static** members.

unions can't have virtual members

Unions can't have **virtual** members.

unnamed struct not instantiated
unnamed class not instantiated
unnamed union not instantiated

An unnamed struct, class, or union was seen which did not declare any objects. This declares a type which can't be used, for example:

```
struct {
int i;
};
```

This type can't ever be used.

type without declarator

A declaration which does not name an object was seen like this:

```
unsigned long *;
```

unions can't have base classes

An attempt was made to derive a union from some class. Unions can't be base classes, or be derived from other classes.

empty base class list

The colon which announces a base class list was not followed by any base class names.

can't derive a class from itself

A class can't be derived from itself. Allowing this would create a class with an infinite number of members.

base class must be 'struct' or 'class'

Base classes must be structures or classes.

can't derive from a union

You can't derive a class from a union.

'<identifier>' is an incomplete struct or class

All base classes must be fully defined before using them to derive another class.

unnamed class can't be derived

You can't derive an unnamed class from any class, for example:

```
class : baseclass {
int I;
};
```

is an error.

misplaced linkage specifier
misplaced const
misplaced volatile
misplaced _loadds
misplaced _export
misplaced pointer specifier

In a declarator, the placement of these keywords must be precise.

no member functions in C

C does not allow member functions.

old-style arguments forbidden in C++

C++ does not permit old-style function definitions as member functions, for example:

```
struct S {
int f(a)
      int a;
      {}
};
```

is an error.

bitfields may only have integral types

The **ANSI C** standard only allows **int**, **unsigned int**, and **signed int** as bitfield types. **CLint++** allows you to enable support for any integral type, and also for enumerated types for bitfields. **C++** allows both of these always. You enable integral support with the ***allow any integral type in bitfields*** (on by default) and ***allow enumerations in bitfields*** (off by default) checkboxes in the Options **|** Lint Options **|** General dialog (**`-waib`** and **`-weib`** in the command line).

bitfields must be members of a struct or class

Bitfields may only be declared as members of structures or classes like this:

```
struct bits {
int bit: 4;
};
```

bitfield must have non-zero width

Bitfields must have non-zero widths. A special exception is allowed for an unnamed bitfield of a given type, which aligns to the next boundary like this:

```
struct bits {
int bits: 4;
int :0;
int bits2: 4;
};
```

This cause **bits** to occupy the low 4 bits of the first int, and **bits2** to occupy the low 4 bits of the second int. It is only useful when declaring structures for memory mapped I/O.

missing bitfield width

A bitfield declaration with no size was seen like this:

```
struct bits {
int bits: ;
};
```

unknown language string '<string>'

CLint++ only supports **extern "C"** and **extern "C++"** linkage declarations. Other language strings are not supported.

missing '}' to close extern "C" block

CLint++ did not see the closing brace to a language block which should look like this:

```
extern "C" {
declarations . . .
}
```

can't have pointer to reference

C++ does not allow pointers to references because references are not true objects, so:

```
int &* pref;
```

is an error.

constant '<identifier>' needs initializing

When a const object is declared, it should be initialized, because it will not be possible to give it a value later.

only one 'extern \"C\"' function may be overloaded

C++ allows any number of functions to be overloaded provided their types are sufficiently different. This is because by name mangling, each such function has a different name. However, **C** function names are not mangled, so at most one C function may be overloaded with the same name as a family of overloaded **C++** functions.

overloaded functions may not differ only in return type

C++ allows any number of functions to be overloaded provided their types are sufficiently different. This is because by name mangling, each such function has a different name. However, the return type is not included in the mangled name, so functions differing only in return type have the same mangled name, and so can't be overloaded. C++ resolves overloading only using the arguments of the call anyway, so the return type does not participate in overload resolution.

'auto' used outside a function

The storage class **auto** is used to denote automatic storage for functions. It can't be used outside a function.

'<identifier>' is not a type name

CLint++ saw an identifier used in a position where a type name is required, but it is not a type name.

undeclared type name '<identifier>' in throw

CLint++ saw an identifier used in a **throw** argument list where a type name is required, but it is an undefined name.

'throw' arguments may only be types

throw is used to declare the types of exceptions a function may raise during execution of its body, or which functions it calls may raise. For example:

```
            void f() throw(int);
```

declares that f() may throw an exception of type int at runtime. These types are not arguments, and may not be named.

'throw' arguments may not be defaulted

throw is used to declare the types of exceptions a function may raise during execution of its body, or which functions it calls may raise. For example:

```
            void f() throw(int);
```

declares that f() may throw an exception of type int at runtime. These types are not arguments, and may not have default values.

default arguments only supported in C++

Only **C++** allows functions to declare default values for arguments like this:

```
            void defarg(int = 1);
```

non-default arguments may not follow default arguments

When default arguments are used, they must occur as the last arguments in a function. For example:

```
int f(int a = 0, int b);
```

is an error because **b** does not have a default value.

only virtual functions may be declared pure

A pure **virtual** function is used in an abstract class, and its presence makes the class definition abstract. Pure functions are declared like this:

```
virtual void f() = 0;
```

The constant used must be zero, and may only be applied to functions declared **virtual**. The declaration acts as a place holder for real functions which must override the pure function in derived classes.

definition of '<function>' previously seen at file(NNN)

The named function was previously defined in **file** at line **NNN**. You may only define a function once. This may occur with member functions like this:

```
class X {
public:
void f()    { return 0; }
};

void X::f()
{
return 0;
}
```

The second definition is an error because the definition in the class body was already seen.

'~name' is not a destructor

Destructors are functions whose name begins with a **'~'** and must be the same as the name of the class. **CLint++** saw a name like this, but the name did not match the class name. For example:

```
class X {
~Y();
};
```

where **Y** is not the name of the class **X**, so **~Y** is not a legal destructor name.

empty expression in 'if'
empty expression in 'while'
empty expression in 'switch'
empty expression in 'do'

Each of these requires a parenthesized condition which is not empty. We saw something like:

```
        if ()
```

syntax error

It was too hard for CLint++ to work out what was specifically wrong.

'::' needs a following identifier

The **'::'** operator is used to name a specific member of a class, or to access a global name hidden in the current scope. No name was seen following **'::'**. For example:

```
int i;

void f()
{
int i = ::i;
}
```

In this case the hidden global **i** is being assigned to the local **i**.

'!' is not a binary operator
'~' is not a binary operator

One of these operators was used as a binary operator. They may only be unary.

'.' needs struct/union

Members of structures, classes and unions are accessed using the **'.'** operator like this:

```
struct X {
int i;
};

void f()
{
X x;
int i = x.i;
}
```

The name preceding the **'.'** operator is not the name of a structure or union.

'->' needs struct/union pointer

Members of structures, classes and unions are accessed using the **'->'** operator like this:

```
struct X {
int i;
};

void f()
{
X *x;
int i = x->i;
}
```

The name preceding the **'->'** operator is not a pointer to a structure or union.

can't use 'this' here

this is a reserved name of the local pointer to the class for which a member function is called. It only exists for non-static member functions. You attempted to use it in a static member function, or a non-member function.

can't convert '<type1>' to '<type2>'

Most arbitrary conversions between types are not allowed. You probably need a cast to make this conversion. **C++** is more strict about conversions than **C** is. Although in **C++**, **char**, **unsigned char**, and **signed char** are different types, **CLint++** considers differences between pointers to these types to be warnings and not errors.

redeclaration of '<identifer>' differs from file(NNN)

You previously declared this symbol differently in **file** at line **NNN**.

objects with constructors can't be initialized with '{'
objects with virtual functions can't be initialized with '{'
objects with protected members can't be initialized with '{'
objects with base classes can't be initialized with '{'

Both **C** and **C++** allow arrays and structures to be initialized with a braced list of initializers. **C++** does not allow this syntax to be used when the array is a class with constructors, virtual functions, protected members, or base classes.

function definitions can't be 'extern'

You can't define an explicitly **extern** function, although you can prototype such a function. For example:

```
extern int f()
{
return 0;
}
```

is an error, but:

```
extern int f();
```

is not.

initialized objects can't be 'extern'

You can't define an explicitly **extern** object which is initialized, for example:

```
            extern int i = 0;
```

is an error.

only static members may be defined

You can only define static non-function members outside a class, for example:

```
struct X
{
int i;
};
int X::i = 0;
```

is an error.

member definitions may not have a storage class

When static members are defined outside the class in which the are declared, no storage class may be used. For example:

```
class X {
static int i;
};
static int X::i;
```

is an error. This is because static members actually have global scope, so no additional linkage specifier is possible.

'operator ??' may only be declared as a function

The operator keyword may only be used as part of the name of an operator function. You tried to use it as a variable name. For example:

```
class X;

int operator +(X, int);
```

declares an operator function overloading '+' which takes an object of type **X** as an argument, and (presumably) adds an integer, returning an integer. However,

```
int operator +;
```

is an error because no function was declared.

'operator ??' is not a valid operator

You tried to declare an operator function for an unsupported operator. Note that '?', and ':' can't be declared as operator functions.

'operator ??' must have no arguments

The unary operators '!' and '~' when used as member functions take no argument, because the argument is the class for which they are invoked.

'operator ??' must have one argument

Binary operators like '+', '-' and so on when used as member function take only one argument, because the first operand is the class for which they are invoked. Unary operators '!' and '~' used as non-member functions take a single argument which must be the name of some class.

'operator ??' must have two arguments

Binary operators like '+', '-' and so on when used as non-member function must have two arguments of which at least one must be the name of some class.

only functions may be declared 'inline'

The **inline** keyword is a hint to the compiler to expand a function body at the point of use. It can only apply to functions.

'operator <type>' must be declared as a member function

Conversion operator functions overload casts and provide a user defined conversion from one type to another, and may only be declared as member functions. For example:

```
int operator int(double);
```

is an error. The correct use is like:

```
class C {
public:
operator int();
};
```

which declares a conversion from objects of type **C** to **int**.

'<class>' is not a base class of '<class>'

You initialized a non-member or non-base of a class in a constructor. For example:

```
class X {
int i;
X(): J()
{
      i = 0;
}
};
```

tries to initialize **J** which is not a base class or member of **X**.

extra '{'
extra ':'

An unexpected token was seen. Remove it.

reference '<identifier>' needs initializing

When you declare a reference at file level or in a function other than as an argument, it must be initialized to some object to which it will refer. References are only aliases for the value they refer to, and are not objects. For example, in:

```
int f()
{
int j;
int& rj = j;
int& rk;
}
```

the second reference declaration is an error because it doesn't refer to an object.

can't match call of '<function>'

C++ resolves operator overloading by looking at the types of the arguments to the call, and looking for a function declaration with matching types. The matching algorithm is complex, but can be summarized as:

> Look for a function matching the given argument types, or for which the arguments could be converted using standard conversions; of those which match, select that function which has a strictly better match for at least one argument.

For a complete description, see the ARM 13.2. This description is not simple or concise.

call of function with no prototype

C++ requires all functions to be prototyped before use. **ANSI C** only strongly advises this. It is often caused by omitting a library header containing the required definitions.

ambiguity between '<funcname1>' and '<funcname2>'

While resolving the call of an overloaded function, two or more candidates were found which matched the arguments of the call equally well. Only the first two such functions are named in this message. The matching algorithm is complex, but can be summarized as:

> Look for a function matching the given argument types, or for which the arguments could be converted using standard conversions; of those which match, select that function which has a strictly better match for at least one argument.

For a complete description, see the ARM 13.2. This description is not simple or concise.

operator new must return 'void *'
operator new[] must return 'void *'

operator new however declared must return a '**void ***' value.

argument of operator new must be 'size_t'
argument of operator new[] must be 'size_t'

operator new however declared must have a first argument of type 'size_t'.

operator delete can't return a value
operator delete[] can't return a value

operator delete however declared may not return any value.

argument of operator delete must be 'void *'
argument of operator delete[] must be 'void *'

operator delete takes a single argument of type '**void ***'.

overload not allowed here
template not allowed here
inline not allowed here
static not allowed here
typedef not allowed here
extern not allowed here
public not allowed here
private not allowed here
protected not allowed here

These keywords may not occur in argument list of functions, and many other places.

empty parentheses

We saw a () in an expression other than in the context of a call or cast.

arrays can't be initialized with '('

In C++, you can initialize most objects with a parenthesized initializer list, which is meant to resemble a function call. However, arrays of any type can't be initialized this way.

declaration not allowed here

C requires that all declarations in a block precede all executable statements. C++ allows declarations anywhere.

reference member '<identifier>' in class without constructors

When a class has a reference member, it also requires a constructor which will assign a value to the reference member for every instance of the class.

redefinition of '<identifier>' previously defined at file(NNN)

Objects may only be defined once. This one was previously defined in **file** at line **NNN**.

pointer demoted to 'near'

You assigned a **far** or **huge** pointer to a **near** pointer. This is never portable, and if you mean it, use a cast.

can't have array of references

Because references are not real objects, you can't have an array of them.

local class '<class>' can't have static members

Local classes (that is classes declared locally to a function) can't have static members.

switch expressions must be integral

Switch expressions must be integral. They can't be pointers, or floating values.

can't define a 'struct' in arguments
can't define a 'union' in arguments
can't define a 'enum' in arguments

Although you can refer to classes, unions, or enumerations in function arguments, you can't define these in a function argument.

missing base class name

You declared a class as though it was derived from a baseclass, but the name was missing. The correct method is:

```
class X
{
....
};
class Y: public X
{
};
```

The access specifier **public** is advised, as there are very few cases where **private** or **protected** derivation is appropriate.

derived classes need a definition

You declared a class as though it was derived from a base class, but no definition was given for the class. For example:

```
            class X: public Y;
```

is an error.

'identifier' is not a class name

You tried to use the name as a structure tag, but it was already declared as something else.

destructor '~name' not followed by '('

The only things you can do with destructors is define them or call them. In both cases a parenthesis must follow the destructor name.

call of undefined function '<funcname>'

You called a function not previously declared. In C++ this is an error, C implicitly declares the function **extern** with no argument list.

use '.' or '->' to call '<funcname>'

You tried to call a non-static member function without an object. You must call such members with an object like this:

```
class C {
void g();
};
void f()
{
C c;
c.g();
}
```

take the address of or call member function '<funcname>'

You used a non-static member function as though it was an ordinary member. You must either call it, or take its address.

function is too similar to '<funcname>' for overloading

You tried to declare a function which differed from a prior function declaration only in that one argument is a reference. Such functions would be ambiguous when called, and so may not be overloaded. For example:

```
void f(int);
void f(int&);
```

is an error because the second **f** is too similar to the first.

'<member>' is not accessible

The member is not accessible in the scope. For example,

```
class X
{
void f();
};

X x;

void g()
{
x.f();
}
```

provokes this complaint when x.f() is called because g() is not a member of X, and f() is a private member of X.

derivation with no access specifier defaults to 'private'

You derived one class from another without an explicit access specifier. This defaults to **private**, which is almost never what was wanted. Use an explicit access specifier, probably **public**.

base class '<class>' included again

You tried to derive a class from the same base class twice. For example:

```
class A {
.....
};
class B: public A, public A
{
};
```

is an error.

'<class>' is also a base class of '<class>'

This warning notes that one of the base classes in a derivation is also a base class of another base class in a derivation. For example:

```
class A {
....
};
class B: public A {
};
class C: public B, public A {
};
```

will cause this warning when the declaration of **C** is seen, because **A** is also a base class of **B**.

ambiguity between members '<member1>' and '<member2>'

You have members in classes with multiple inheritance which have the same name but were inherited by different routes. For example:

```
class A {
int i;
};
class B: public A {
};
class C: public A {
};
class D: public B, public C {
};

void f()
{
D d;
int k = d.i;
}
```

causes this error. The solution is to declare **B** and **C** as deriving virtually from **A**, which ensures that there will be only one copy of **A** in **D** like this:

```
class A {
int I;
};
class B: public virtual A {
};
class C: public virtual A {
};
class D: public B, public C {
};

void f()
{
D d;
int k = d.i;
}
```

ANSI/ISO keyword 'and' used
ANSI/ISO keyword 'and_eq' used
ANSI/ISO keyword 'bitand' used
ANSI/ISO keyword 'bitor' used
ANSI/ISO keyword 'compl' used
ANSI/ISO keyword 'not' used
ANSI/ISO keyword 'or' used
ANSI/ISO keyword 'or_eq' used
ANSI/ISO keyword 'xor' used
ANSI/ISO keyword 'xor_eq' used
ANSI/ISO keyword 'not_eq' used

The **ANSI/ISO** draft standard defines these keywords as aliases for '&&', '&=', '&', '|', '~', '!', '||', '|=', '^', '^=', and '!=' respectively. **CLint++** only enables support for these keywords if you enable the ***support ANSI/ISO keywords*** checkbox in the Options | Lint Options | C++ dialog (**`wans`** on the command line).

'overload' is obsolete

The **ANSI/ISO** draft standard deprecates the use of the keyword **overload**. Provided you use it only as an identifier, no problems arise. If you use it like this:

```
int f();
overload int f(int);
```

then you get this warning, and you can't then use it as an identifier.

'template' must be followed by '<'

The syntax of **template** is:

```
template <class X, more . . > declaration . . .
```

template arguments must start with 'class'

The syntax of **template** is:

```
template <class X, more . . > declaration . . .
```

template arguments must have an identifier

The syntax of **template** is:

```
          template <class X, more . . > declaration . . .
```

template argument lists must end with '>'

The syntax of **template** is:

```
        template <class X, more . . > declaration . . .
```

templates must declare an object

The syntax of **template** is:

```
template <class X, more . . > declaration . . .
```

illegal template instantiation of '<template>'

You tried to instantiate a template before the template definition was seen. Although you can declare a template to get a forward reference, the template definition must be seen before the first instantiation of the template. This does not mean that the template can't be used before the definition, provided it is only used within the body of another template definition which itself won't be expanded until after the declaration is seen. See the ARM chapter 14.

'for' initializer '<identifier>' not initialized

In C++ you can declare a loop variable in the initializer of a for statement like this:

```
            for (int i = 0; i < 10; i++)
```

However, we saw something like:

```
            for (int i; i < 10; i++)
```

which guarantees that **i** has no initial value.

can't initialize reference '<identifier>' with '<type>'

In C++, references may be initialized by objects of a different type. However, this causes the compiler to generate a temporary of the correct type, convert the value to that temporary, and then refer to the temporary. If the reference is *not* **const**, then you could change the value of the temporary without changing the original value. For example:

```
long l;
int& ri = l;
```

will produce this error. But if the reference is **const**, then no problem can occur:

```
long l;
const int& cri = l;
```

is fine.

temporary used to initialize reference '<identifer>'

In C++, references may be initialized by objects of a different type. However, this causes the compiler to generate a temporary of the correct type, convert the value to that temporary, and then refer to the temporary. For example:

```
long l;
const int& cri;
```

This warning is off by default, but you can enable it with the ***warn of reference temporaries*** checkbox in the Options | Lint Options | C++ dialog (**`-wrtm`** on the command line).

only functions can be friends

Only functions may be declared as friends of a class. You tried to define a non-function as a **friend**.

can't use 'friend' outside a class

friend may only be used in a class declaration to specify that some function or class has access to the protected members of the class. You used **friend** outside any class.

can't declare an 'enum' as a friend

friend may only be used in a class declaration to specify that some function or class has access to the protected members of the class. You tried to declare an enumeration as a **friend**.

can't define a 'struct' in a 'friend' declaration
can't define a 'union' in a 'friend' declaration
can't define a 'enum' in a 'friend' declaration

You can't declare any type in a **friend** declaration.

only classes or functions can be friends

You can't declare anything other than a class or a function as a **friend** of a class.

non-ANSI escape sequence '\C'

The ANSI/ISO draft C++ standard states that all string escapes other than those defined by the standard are incorrect. The C standard states that such escapes are equivalent to not using the '\' character. However, this is such a prolific source of errors, we have decided to treat C and C++ the same.

anonymous union can't have member function '<function>'

You can't declare member functions in an anonymous union. Anonymous unions are a C++ convenience which avoids the need to tag a union like this:

```
union {
int j;
long l;
};
```

where both objects share the same storage, and need no member access operator. They are often used inside structure definitions.

global anonymous unions must be 'static'

Anonymous unions are a C++ convenience which avoids the need to tag a union like this:

```
union {
int j;
long l;
};
```

where both objects share the same storage, and need no member access operator. When used at global level (outside any function or class definition) they must be declared **static**.

pure member functions must be declared with '= 0'

Abstract classes are defined by including one or more pure member functions within the definition. Such functions are declared like this:

```
class C {
virtual void f() = 0;
};
```

We saw a non-zero constant.

initializing reference member '<member>' with non-reference parameter

You have defined a constructor for a class containing one or more reference members. The constructor tries to initialize a reference member with a non-reference. For example:

```
class C {
int& ri;
C(int v): ri(v) {}
};
```

produces this error when the constructor **`C()`** is parsed. You should use something like:

```
C(int& rv): ri(rv) { }
```

reference member '<member>' not initialized

You have defined a constructor for a class containing one or more reference members. The constructor fails to initialize one or more of these references. For example:

```
class C {
int& ri;
C() {}
};
```

produces this error when the constructor **`C()`** is parsed. You should use something like:

```
C(int& r): ri(r) { }
```

base class '<base>' not initialized

You have defined a constructor for a class derived from one or more bases. The constructor fails to initialize one or more of these bases. For example:

```
class B
{
int i;
}
class C: public B {
C() {}
};
```

produces this error when the constructor **`C()`** is parsed. You should use something like:

```
C(int v): B(v) { }
```

member '<member>' declared outside its class

You tried to define a non-static member outside the class. For example:

```
class C {
int j;
}
int C::j;
```

This is only meaningful for static members.

can't define pure member function '<funcname>'

You tried to provide a definition for a pure member function. Such functions act as place holders to be overridden in a derived class and can't be defined.

can't inline member function '<funcname>' after use

Member functions are assumed to be defined when first called. At that time the compiler generates a call instruction to the member. You can define a member function as **inline** prior to its first use - but not after.

can't create an instance of abstract class '<class>'

Abstract classes provide a definition of functionality to be inherited in derived classes which will override the pure member functions in the abstract base. Because pure functions are placeholders, no instance of an abstract class can be created. If you could, then it would be possible to call the pure member. In most implementations this would result in a call to machine address 0.

use '.' or '->' to access member '<member>'

You attempted to use a non-static member of a class in a **static** member function. Static member functions don't have **this**, so you must explicitly use a class or class pointer to call or access the member.

copy constructor '<funcname>' takes reference argument

You tried to declare a copy constructor which takes an instance of the class (as opposed to a reference to it) as its first argument like this:

```
class C {
C(C c);
};
```

This is illegal because calling the constructor would recursively call it to construct the argument **c**, which then recursively calls it again to construct its argument **c**, and so on.

'operator =' must be a member function

operator= is unique amongst operator functions in that it can only be declared as a member of a class, and not at file level.

'operator =' should return a class reference

This warning notes that a declaration of **operator=** does not return a reference to the class of which it is a member. Although not illegal, this is highly questionable, because it should implement the assignment operation for the class like this:

```
class C {
C& operator = (C&);
};
```

This clearly makes sense, because in an expression like:

```
C c1, c2;
c1 = c2;
```

we should expect that **c1** has in some sense had the value of **c2** assigned to it.

can't construct as 'class::~class()' is not accessible

You can control creation of instances of a class by making its destructor **private** or **protected**. No compiler will allow creation of a class if the destructor is not accessible, because it could not then generate a call of the destructor when the object goes out of scope. However, you may have simply omitted to make the destructor **public**.

can't derive from base '<class>' with private destructor

If a class could be derived from a class with a private destructor, no instance of the derived class could be destroyed.

can't initialize a 'new' array

You can't initialize an array constructed with **new**. If array elements need initializing, you must create a default constructor (one which takes no non-default arguments) to do this.

argument '<argname>' already has a default value

You can only declare a default argument for a function once. Subsequent declarations or a definition must not specify a default argument. For example:

```
class C {
int f(int = 0);
};
int C::f(int a = 0)
{
return a;
}
```

is an error.

argument NN already has a default value

You can only declare a default argument for a function once. Subsequent declarations or a definition must not specify a default argument. For example:

```
int f(int = 0);
int f(int = 0);
```

is an error.

argument '<argname>' defaulted after use

It is possible to declare default arguments for a function in pieces like this:

```
            int f(int j, int k, int l);
            int f(int j, int k, int l = 0);
            int f(int j, int k = 1, int l);
```

However, if a call of **f** occurred between the second and third declarations, you would get this error, because you would be trying to change the meaning of code already generated.

argument NN defaulted after use

It is possible to declare default arguments for a function in pieces like this:

```
int f(int, int, int);
int f(int, int, int = 0);
int f(int, int = 1, int);
```

However, if a call of **f** occurred between the second and third declarations, you would get this error, because you would be trying to change the meaning of code already generated.

'<funcname>' hides virtual function '<funcname>' declared at file(NNN)

It is not an error to hide a virtual function in a derived class, but doing so defeats the purpose of the virtual function mechanism. Virtual functions are intended to be overridden, not hidden, which means that the functions in the derived class should have the same signature as in the base class.

You can control these reports specifically with the ***warn of hidden virtual functions*** checkbox in the Options | Lint Options | C++ dialog (or **`whvf`** for the command line program).

'::*' must be preceded by a class name

This operator is used in declaring pointers to members of a specific class. For example:

```
class C {
int I;
};
void f()
{
int C::*ip = &C::i;
}
```

This demonstrates the syntax used to declare pointers to members. It is necessary to specify the name of the class, because this is a pointer to a member of any class **C** (effectively an offset), not a pointer to some specific instance of the member, for which the normal address taking mechanism works well.

pointer to member of '<class>' not a member of '<class>'

An attempt was made to use a pointer or instance of some class to access a pointer-to-member of a different class. For example:

```
class C {
int I;
};
class D {
int I;
};
void f()
{
int C::*ip = &C::i;
D d;
int j = d.*ip;
}
```

produces this error when **d.*ip** is parsed because **ip** has the type pointer-to-member of **C**, but **d** is of type **D**.

pointer to member must follow '.*'
pointer to member must follow '->*'

An attempt was made to use a pointer to member access operator to access an object whose type is not pointer to member.

scanf: 'double *' argument to %f - use %lf (arg NN)

A **double *** argument was used for **%f** which expects a **float *** argument.

scanf: %f requires 'float *' argument (arg NN)

You used **%f** which expects a **float *** argument.

can't have protected member '<member>' in anonymous union

Members of anonymous unions may not be **private** or **protected**.

incorrect library file format in 'file'

The contents of the CLint++ library file are corrupt. It is ignored.

argument list already typed

You have an old style function definition for a function which was previously prototyped. Change the definition to the ANSI style.

'goto' bypasses initialization of local variable

C++ does not permit the use of **goto** in such a way that the initialization of any object is bypassed, because this could cause a destructor to be called for an object which was never constructed.

'case' bypasses initialization of local variable

C++ does not permit the use of goto in such a way that the initialization of any object is bypassed, because this could cause a destructor to be called for an object which was never constructed. This applies to **switch** statements also, because a switch is nothing more than a computed goto.

'default' bypasses initialization of local variable

C++ does not permit the use of goto in such a way that the initialization of any object is bypassed, because this could cause a destructor to be called for an object which was never constructed. This applies to **switch** statements also, because a switch is nothing more than a computed goto.

function definition style is obsolete

The old style function declaration syntax is obsolete in **C++**. CLint++ supports it for non-member functions only for backward compatibility with older code.

syntax error in template declaration

There was a syntax error in the declaration of a template.

can't instantiate template

A template was declared in such a way that it appeared that an instance of the template was also being declared like this:

```
template <class T> test {
T t;
} obj;
```

which appears to be declaring **obj** as an instance of **test<T>**. This is an error.

empty body of 'try' block
empty body of 'catch' block

The syntax of **try** and **catch** is:

```
try {
.....
}
catch (catch_expr)
{
......
}
```

followed by optional additional **catch** blocks to catch errors of differing types. These blocks may not be empty.

need 'catch' block after 'try' block

The syntax of **try** and **catch** is:

```
try {
.....
}
catch (catch_expr)
{
......
}
```

followed by optional additional **catch** blocks to catch errors of differing types. At least one **catch** block must follow a **try** block.

'throw' with no throw specifier

You used **throw** but the calling function is not declared as throwing exceptions. A function which uses throw should declare the types of exception it throws like this:

```
void f() throw(int)
{
throw 1;
}
```

It is not an error to throw a type when no **throw()** clause is present.

You can control these reports specifically with the ***function may throw a type, but no throw specifier*** checkbox in the Options | Lint Options | C++ dialog (or **`wtns`** for the command line program).

'throw' of type '<type>' not in throw specifier

You used **throw** with an exception type not specified in the calling function. A function which uses **throw** should declare the types of exception it throws like this:

```
void f() throw(int)
{
throw 1L;
}
```

In this example, the function throws a **long**, although it is declared as throwing an **int**.

You can control these reports specifically with the ***function may throw a type not in throw specifier*** checkbox in the Options | Lint Options | C++ dialog (or **`wtni`** for the command line program).

'throw' not permitted

You used **throw** in a function which states that it does not throw exceptions. A function which states that it won't use **throw** is declared like this:

```
void f() throw()
{
throw 1L;
}
```

In this example, the function throws a **long**, although it is declared as not throwing an exception, which is an error.

'<funcname>' may throw, but 'throw' not permitted

You called a function which can throw an exception in a function which states that it does not throw exceptions. In this example, **f** is declared as not throwing exceptions, but it calls **g** which could throw an **int**.

```
void g() throw (int);
void f() throw()
{
g();
}
```

The draft ANSI standard states that a compiler is not allowed to fail a compilation because the called function ***may*** throw an exception - and may not.

'<funcname>' may throw a type not in throw specifier

You called a function which may throw an exception type not specified in the calling function. In this example, **f** is declared as possibly throwing a **long**, but it calls **g** which could throw an **int**.

```
void g() throw (int);
void f() throw(long)
{
g();
}
```

You can control these reports specifically with the ***function may throw a type not in throw specifier*** checkbox in the Options | Lint Options | C++ dialog (or **wtni** for the command line program).

'<funcname>' may throw a type, but no throw specifier

You called a function which may throw an exception but the calling function is not declared as throwing exceptions. In this example, **f** is declared without a throw specifier, but it calls **g** which could throw an **int**.

```
void g() throw (int);
void f()
{
g();
}
```

You can control these reports specifically with the ***function may throw a type, but no throw specifier*** checkbox in the Options | Lint Options | C++ dialog (or **`wtns`** for the command line program).

function may throw a type, but no throw specifier

You called a function which may throw an exception but the calling function is not declared as throwing exceptions. In this example, **f** is declared without a throw specifier, but it calls ***g** which could throw an **int**.

```
void (*g)() throw (int);
void f()
{
g();
}
```

You can control these reports specifically with the ***function may throw a type, but no throw specifier*** checkbox in the Options | Lint Options | C++ dialog (or **`wtns`** for the command line program).

function may throw, but 'throw' not permitted

You called a function which may throw exceptions, but your functions states that it does not throw exceptions. In this example, **f** is declared as not throwing exceptions, but it calls ***g** which could throw an **int**.

```
void (*g)() throw (int);
void f() throw()
{
g();
}
```

The draft ANSI standard states that a compiler is not allowed to fail a compilation because the called function ***may*** throw an exception - and may not.

function may throw a type not in throw specifier

You called a function which may throw an exception type not specified in the calling function. In this example, **f** is declared as possibly throwing a **long**, but it calls ***g** which could throw an **int**.

```
void (*g)() throw (int);
void f() throw(long)
{
g();
}
```

You can control these reports specifically with the ***function may throw a type not in throw specifier*** checkbox in the Options | Lint Options | C++ dialog (or **`wtni`** for the command line program).

bitfield too short for '<enumname>', should be NN

When you declare a bitfield member of an enumerated type, the field with must be large enough for all values in the type. For example:

```
enum E { a, b, c };
struct S {
E e: 1;
}
```

is an error because E requires 2 bits to cover the range **a** to **c**.

bitfields must be 'int', 'signed int', or 'unsigned int'

ANSI C only allows **int**, **unsigned int**, or **signed int** as types for bitfields. By default, CLint++ allows you to use any integral type for a bitfield. If you want to enforce the ANSI restriction, clear the ***allow any integral type in bitfields*** checkbox in the Options | Lint Options | Type dialog (or **`-waib`** for the command line program).

'NN' outside range of enumeration '<enumname>'

When you cast a constant to an enumerated type, CLint++ checks that the constant is in the range of the enumeration.

'<funcname>' called with non-volatile object

Member functions may be declared **volatile**: if so they may only be called with a class which is a volatile object. Inside the member function, **this** is a volatile pointer. For example:

```
class C {
void f() volatile;
};

void f()
{
C c;
c.f();
}
```

This call of **f** is an error because **c** is not a volatile object.

'<funcname>' called with const object

Member functions may be declared **const**: if so, they promise to not modify the calling object. Inside the member function, **this** is a const pointer. For example:

```
class C {
void f() const;
void g();
};

void f()
{
C c;
c.f();
const C c1;
c1.g();
}
```

This call of **g** is an error because **c1** is a const object, but **g** is able to modify the calling object.

array size in 'delete' ignored

When you delete an array allocated with **new[]**, you must release it with **delete[]**. In older versions of C++ you were required to pass the number of elements in the brackets to the call of **delete[]**. Modern versions of C++ store the number of elements in hidden storage visible to **delete**, so this is no longer required. CLint++ supports the older syntax, but ignores the value in brackets and warns of it.

You can control these reports specifically with the ***array size for 'delete' ignored*** checkbox in the Options | Lint Options | C++ dialog (or **`wasd`** for the command line program).

too few arguments in template '<templname>'

You tried to invoke a template with fewer arguments than it was defined as taking.

too many arguments in template '<templname>'

You tried to invoke a template with more arguments than it was defined as taking.

template function '<funcname>' requires body

When declaring a template function, you must supply a definition.

type error in argument '<argname>' in template '<templname>'

When a non-type argument is used in a template, the type of the actual argument (which must be a constant expression) must match the type of the formal argument.

invalid template '<templname>'

When instantiating a template, the arguments must be all actual arguments (real types or values) or formal types (obtained from formal types of the enclosing template definition).

can't declare a 'class' in a 'sizeof' or cast expression
can't declare a 'union' in a 'sizeof' or cast expression
can't declare a 'enum' in a 'sizeof' or cast expression
can't declare a 'struct' in a 'sizeof' or cast expression

Although you can mention any type or expression in **sizeof**, you can't define new types.

template argument '<argname>' not a template in '<templname>'

You can define templates which take other templates as formal arguments. You tried to instantiate such a template without supplying an actual template name as the argument.

'template class' must be followed by an instantiated class name

You can explicitly instantiate a template using the syntax:

```
template <class T> class templ {
T j;
};

template class templ<int>;
```

However, the name following **template class** was not a template instantiation.

missuse of template name '<templname>'

The only things you can do with template names are instantiate them by providing an actual argument list enclosed in <> brackets, or use them as formal parameters in other template definitions. A template name was used other than one of these ways.

second argument of operator delete must be 'size_t'
second argument of operator delete[] must be 'size_t'

Both forms of **operator delete** when declared as member functions may be defined as taking a second argument: if present it must be of type **size_t**.

operator delete takes no more than two arguments
operator delete[] takes no more than two arguments

Both forms of **operator delete** when declared as member functions may be defined as taking a second argument. No more than two arguments may be declared.

operator delete can't be overloaded
operator delete[] can't be overloaded

operator delete can't be overloaded, so no more than one declaration for it may be provided.

::operator delete takes only one argument
::operator delete[] takes only one argument

The global forms of **operator delete** only take a single **void *** argument.

operator new may not be virtual or abstract
operator new[] may not be virtual or abstract

Neither form of **operator new** may be **virtual** or pure.

redundant '&' used with array or function

When the name of an array or function is mentioned, it is converted to its address as though by the use of '&'. So using '&' on an array or function name is redundant.

'<identifier>' previously declared 'extern' at file(NNN)

We saw a **static** declaration of an object previously declared **extern**.

#message <message>

CLint++ supports a non-ANSI directive #message. This line was caused by the use of this directive. You can use this to place a message in the error report.

template arguments are implicit in constructor '<constname>'

When a template class is declared, the constructor name should not be qualified by the template arguments. For example:

```
template <class T> vector {
vector<T>();      // wrong!
vector();         // right
};
```

This is only a warning because unfortunately some compiler header files do this.

template arguments are implicit in destructor '<destname>'

When a template class is declared, the constructor name should not be qualified by the template arguments. For example:

```
template <class T> vector {
~vector<T>();      // wrong!
~vector();         // right
};
```

This is only a warning because unfortunately some compiler header files do this.

omitting an initializer baseclass name is obsolete

The original C++ language definition did not support multiple inheritance, so a constructor initializing a base class did not need to state the base class name - there could be only one. For example:

```
class A
{
// . . . .
};
class B: public A
{
B(): () { }
};
```

This remains legal under the draft ANSI/ISO C++ standard, but is now considered obsolete. It should be written as:

```
class A
{
// . . . .
};
class B: public A
{
B(): A() { }
};
```

baseclass can't be omitted with multiple derivation

The original C++ language definition did not support multiple inheritance, so a constructor initializing a base class did not need to state the base class name - there could be only one. For example:

```
class A
{
// . . . .
};
class B: public A
{
B(): () { }
};
```

This remains legal under the draft ANSI/ISO C++ standard, but is now considered obsolete. However, when multiple inheritance is used, you must name which base class is being initialized. For example:

```
class A
{
// . . . .
};
class B
{
// . . . .
};
class C: public A, public B
{
C(): A(), B() { }
};
```

'<classname>' is not derived from any base classes

The original C++ language definition did not support multiple inheritance, so a constructor initializing a base class did not need to state the base class name - there could be only one. For example:

```
class A
{
// . . . .
};
class B: public A
{
B(): () { }
};
```

This remains legal under the draft ANSI/ISO C++ standard, but is now considered obsolete. However, you used this syntax for a class which has no baseclass like this:

```
class C
{
C(): () { }
};
```

prefix 'operator ++' used as postfix
prefix 'operator --' used as postfix

The original **C++** language definition did not support separate operator functions for the prefix and postfix versions of **'++'** and **'--'**. The ANSI/ISO standard does, like this:

```
class S
{
S& operator ++();        // prefix version
S& operator ++(int);           // postfix version
};
```

The standard states that the argument will be 0 unless explicitly called for the postfix version. For compatibility, if no postfix version is defined, then a postfix **'++'** will use the prefix version, but produce this warning. You should define a postfix operator function.

You can control these reports specifically with the ***prefix operator used as postfix*** checkbox in the Options | Lint Options | C++ dialog (or **`wpre`** for the command line program).

'<identifier>' previously declared 'static' at file(NNN)

Both C and C++ allow you to declare a function or variable as **static**, and later define or declare it without the **static** storage class specifier. CLint++ considers this to be bad practice. For example:

```
static void f();
void f()
{
}

static int i;
extern int i;
```

You can control these reports specifically with the ***previously static or previously extern*** checkbox in the Options | Lint Options | General dialog (or **`wpsc`** for the command line program).

no cast or sizeof allowed in #if
no cast or sizeof allowed in #elif

The **ANSI C** standard states that you can't use casts or sizeof in **#if** or **#elif** expressions. Some pre-ANSI compilers attempt to use sizeof to determine the sizes of runtime types, and casts to determine if char is signed or not. If you have one of these compilers, (like the supported Archimedes compiler) you can support this with the ***Allow Sizeof or cast in #if/#elif*** checkbox in Options | Lint Options dialog (or **-G** for the command line program).

constructor '<constname>' declared with return type

Constructors may not return a value. You declared a constructor which attempts to return a value.

'<base>' is not an unambiguous base class of '<class>'

When a class is derived from base classes which in turn derive from another base, this error can occur. For example:

```
class A
{
int f();
};

class B: public A
{
};

class C: public A
{
};

class D: public B, public C
{
};

void g()
{
D d;
A a = d;
}
```

provokes this error, because it is ambiguous whether B::A or C::A should be copied to a. If D's bases declared their bases **virtual**, this would not occur.

destructor '<destname>' declared with return type

Destructor may not have arguments or return a value. You declared a destructor which attempts to return a value.

template arguments missing from '<templname>'

When template names are used in a template function definition, other than as a constructor a destructor name, the argument types must bee given. For example:

```
template <class TYPE> class T
{
T<TYPE> *f();
};
template <class TYPE> T<TYPE>* T<TYPE>::f()
{
return 0;
}
```

is correct, but:

```
template <class TYPE> class T
{
T *f();
};
template <class TYPE> T* T<TYPE>::f()
{
return 0;
}
```

is not, and will provoke two warnings, one for each use of **T***.

missuse of 'this'

The keyword **this** may only be used in the bodies of non-static member functions.

function returns a value which is not used

You called a function which returns a value which is not used. You can control these reports specifically with the ***warn of return value unused*** checkbox in the Options | Lint Options | Type dialog (or **`-wrvi`** for the command line program).

'//' comment in C file

In C files only, you can warn of C++ comments with the ***Report '//' comments in C*** checkbox in the Options **|** Lint Options **|** Style dialog (or **`-wcpc`** for the command line program).

qualified member names not allowed

Member names in class definitions may be qualified only by the name of the class being defined. For example,

```
class A
{
int i;
};

class B
{
int B::i;          // legal
int A::j;          // not legal
};
```

'<member>' has differing protections in base class '<base>'

An access declaration or using declaration can be used to make an otherwise hidden member of a base class available in the derived class. For example:

```
class Base
{
public:
      int i;
};

class Derived: private Base
{
public:
// Base::i is not accessible
using Base::i;
Base::I;          // access declaration - deprecated
// Now Base::I is accessible
};
```

However, an access declaration or using declaration does not specify which of a number of overloaded members are intended. So:

```
class Base
{
void f(long);
public:
void f(int);
};

class Derived: private Base
{
public:
using Base::f;          // error f(long) and f(int) have differing
protection
};
```

In this case, an attempt is made to grant access to both Base::f(long) (which is private anyway), and Base::f(int). Their differing protections makes this impossible.

access declarations may not be 'private'

An access declaration or using declaration can be used to make an otherwise hidden member of a base class available in the derived class. However, for this to be effective, the declaration must be itself **public**. For example:

```
class Base
{
public:
      int i;
};

class Derived: private Base
{
Base::I;    // error - private declaration
};
```

must use '> >' here instead of '>>'

When nested templates are declared, the declaration looks like this:

```
template <class T> class test1: public T
{
};

template <class T> class test2: public T
{
};

template class test1<test2<int> >;
```

but we saw this:

```
template class test1<test2<int>>;
```

access declaration would change protection of '<member>'

An access declaration can be used to make an otherwise hidden member of a base class available in the derived class. However, for this to be effective, the member must be at least as visible as the access declaration. For example:

```
class Base
{
protected:
      int i;
};

class Derived: private Base
{
public:
Base::i;    // error - would make i public
};
```

It is not permitted to grant greater access to a class member in a derived class than was granted by the original declaration.

access declaration illegal, '<member>' already defined

An access declaration or using declaration can be used to make an otherwise hidden member of a base class available in the derived class. However this can only be done if the derived class has not overriden the base. For example:

```
class Base
{
public:
      void f(long);
};

class Derived: private Base
{
public:
void f(int);       // overrides Base::f(long)
Base::I;           // error - Derived::f() declared
};
```

should use 'class<type>' instead of 'class'

Whenever a template class name is used other than in the declaration of a constructor or destructor, the class name must be qualified by the formal arguments. For example:

```
template <class T> class test
{
test();                 // legal - constructor
~test();                // legal - destructor
void f1(test&)          // illegal - argument needed
void f2(test<T>&);      // OK
}';
```

There are a number of compiler vendor's headers which do this.

template '<template>' must be declared before use

It is an error to refer to a template class before it has been declared. In this, template classes work just like normal classes. For example:

```
template <class T> class test: base<T>
{
};
```

is illegal unless:

```
template <class T> class base;
```

or a complete definition of ***base*** preceded it. Unfortunately, there are a number of vendor's headers which do this.

const_cast<> will change more than const or volatile

The ANSI/ISO const_cast<> has the syntax:

```
const_cast<T>(e)
```

and casts e to the type T provided only that the types of T and e differ only in constness or volatility. The cast used attempted to change more than just constness or volatility.

can't define a 'class' in the return value of a function
can't define a 'struct' in the return value of a function
can't define a 'union' in the return value of a function
can't define a 'enum' in the return value of a function

You may not define a type as part of the return value of a function. For example:

```
            class X { int i} f();
```

is illegal.

can't use an argument or automatic in default expressions

Default expressions may not use a prior argument or an automatic. For example:

```
void f()
{
int i;
extern void g(int a = I);            // illegal - automatic
extern void h(int a; int b = a);     // illegal - argument
}
```

can't use a non-static member in default expressions

Default expression in member functions may not use non-static members. For example:

```
class X
{
int i;
void f(int a = i);              // illegal - non-static
voif h(int a = j);              // legal - static
static int j;
};
```

unions can't have class members with constructors

Any member of a union which is a class may not have constructors. This is because it is impossible to determine what value a union has, and therefore what (if any) constructors to call. For example:

```
class X
{
	X();
};

union U
{
	X x;			// error - class has constructors
	int I;
};
```

can't assign to 'object' - no operator=()

Assignment to a class object requires that the object either have no constructors or **operator=()** defined (in which case default operations are provided by the compiler), or that the user provide a copy constructor and/or an assignment operator. However, the default operations won't be generated if the class has:

a const member
a reference member
a private operator=
a base with a private operator=
a member with a private operator=

For example:

```
class X
{
int& ri;
X(int& r);
};

void f(int I)
{
X x(I);
X y(I);
x = y;             // fails - no operator=
}
```

operator functions can't have default arguments

An operator function overrides a standard operator. If default arguments were allowed, this would change the number of operands required by a built in operator. For example:

```
class X
{
X& operator+(int I =1); // illegal - fortunately
};
```

would allow is to write:

```
void f()
{
X x;
X y = x +;  // illegal, but might mean y = x.operator+(1)
}
```

const member '<member>' not initialized

All constructors of a class with **const** members must initialize all such members. For example:

```
class X
{
const char *cp;
X() { }                  // error - cp not initialized
};
```

'object' needs initializing - class has no default constructor

Initialization of a class object requires that the object either have no constructors or defined (in which case default operations are provided by the compiler), or that the user provide a constructors. However, the default constructor won't be generated if the class has:

a const member
a reference member
a private constructor
a base with a private constructor
a member with a private constructor

For example:

```
class X
{
const void *vp;
};

X x;        // error - no default constructor
```

must include <typinfo.h> to use typeid()

The value of `typeid(expression)` is of type `const typeinfo&`. That type is defined by the compiler vendor in `<typinfo.h>`. Clint declares a type for `typeinfo` to allow the parse to proceed.

misuse of 'typeid'

The syntax of **typeid** is:

```
typeid(expression)
```

or

```
typeid(typename)
```

In both cases the return value is of type `const typeinfo&`.

missing '}' to close namespace block

The syntax of for **namespace** is:

```
namespace name
{
declarations;
}
```

or

```
namespace
{
declarations;
}
```

The closing brace was not seen.

'namespace' used incorrectly

The syntax of for **namespace** is:

```
namespace name
{
declarations;
}
```

or

```
namespace
{
declarations;
}
```

or

```
namespace name = namespace_name;
```

Either the open brace, the identifier, or '='   was not seen.

namespace declarations must be at global level

The syntax of for **namespace** is:

```
namespace name
{
declarations;
}
```

or

```
namespace
{
declarations;
}
```

or

```
namespace name = namespace_name;
```

Namespace declarations may be nested, but may not occur within class definitions, or function bodies.

'name' redeclared as a namespace name

The syntax of for **namespace** is:

```
namespace name
{
declarations;
}
```

However, ***name*** has already been defined as something else.

'name' redeclared as a namespace alias

The syntax of for a **namespace** alias is:

```
            namespace name = namespace_name;
```

However, ***name*** has already been defined as something else.

'using' used incorrectly

The syntax of **using** is:

```
            using namespace namespace_name;
```

or

```
            using identifier;
```

Some other use than this was seen.

'name' is not a namespace name

The syntax of for a **namespace** alias is:

```
            namespace name = namespace_name;
```

However, ***namespace_name*** is not a namespace name or namespace alias.

using directives may not occur here

The syntax of **using** is:

```
            using namespace namespace_name;
```

or

```
            using identifier;
```

The first form is known as a ***using directive***, and may not occur in an argument list, or class definition.

'<identifier>' declared but not used

A name declared in an unnamed **namespace** block like this:

```
namespace
{
void f();
}
```

is visible only in the translation unit in which it occurs. Therefore, if at the end of the file the name was not used, this warning occurs.

'<identifier>' assigned to but not used

A name declared in an unnamed **namespace** block like this:

```
namespace
{
int i;
}
```

is visible only in the translation unit in which it occurs. Therefore, if at the end of the file the name was assigned to but not used, this warning occurs.

'static' used

With the introduction of anonymous namespaces, there is no need for global static declarations, and they are deprecated. You can control these reports specifically with the ***Deprecate the use of 'static'*** checkbox in the Options | Lint Options **|** C++ dialog (or **-At** for the command line program).

access declaration used

With the introduction of using declarations, access declarations are redundant, and deprecated by ANSI/ISO. You can control these reports specifically with the ***Deprecate the use of access declarations*** checkbox in the Options | Lint Options **|** C++ dialog (or **-Ad** for the command line program).

using declaration would change protection of '<member>'

An using declaration can be used to make an otherwise hidden member of a base class available in the derived class. However, for this to be effective, the member must be at least as visible as the access declaration. For example:

```
class Base
{
protected:
      int i;
};

class Derived: private Base
{
public:
using Base::i;    // error - would make i public
};
```

It is not permitted to grant greater access to a class member in a derived class than was granted by the original declaration.

using declarations may not be 'private'

An access declaration or using declaration can be used to make an otherwise hidden member of a base class available in the derived class. However, for this to be effective, the declaration must be itself **public**. For example:

```
class Base
{
public:
      int i;
};

class Derived: private Base
{
Base::I;    // error - private declaration
};
```

using declaration illegal, '<member>' already defined

An access declaration or using declaration can be used to make an otherwise hidden member of a base class available in the derived class. However this can only be done if the derived class has not overriden the base. For example:

```
class Base
{
public:
      void f(long);
};

class Derived: private Base
{
public:
void f(int);      // overrides Base::f(long)
using Base::I;           // error - Derived::f() declared
};
```

'if' initializer '<name>' not initialized
'while' initializer '<name>' not initialized
'switch' initializer '<name>' not initialized
'for' initializer '<name>' not initialized

ANSI/ISO now allows local variables to be declared in the condition parts of **if**, **while**, and **for**. The parenthesized expression of **switch** is also supported. These variable go out of scope at the end of the clause in which they occur, so if they are not given values when declared, they will have junk values. You can control this specifically with the ***Support ANSI/ISO if, while etc.*** checkbox in the Options | Lint Options | C++ dialog (or **-Ai** for the command line program).

can't initialize member here

Members in class declarations may only be initialized if the member is static **const** and being initialized by an integral constant.

only constructors may be 'explicit'

The ANSI/ISO keyword **explicit** may only be used with single argument constructors in class definitions. For example:

```
class X
{
explicit X(int);
}
```

This means that implicit conversion of int types to X using that constructor is not permitted. For example:

```
X x = 1;                // illegal - must use explicit
construction
X x2 = X(1);// legal, explicit construction
```

implicit call of explicit constructor 'function()'

The ANSI/ISO keyword **explicit** is used to prevent implicit conversion of arguments of constructors with a single non-default argument. For example:

```
class X
{
explicit X(int);
}
```

This means that implicit conversion of **int** types to X using that constructor is not permitted. For example:

```
X x = 1;                // illegal - must use explicit
construction
X x2 = X(1);// legal, explicit construction
```

'explicit' constructors must have one non-default argument

The ANSI/ISO keyword **explicit** is used to prevent implicit conversion of arguments of constructors with a single non-default argument. For example:

```
class X
{
explicit X(int);                        // legal - one argument
explicit X(char, int = 1);     // legal - one non-default argument
explicit X(long, char);        // illegal - two arguments
}
```

misuse of 'explicit'

The ANSI/ISO keyword **explicit** may only be used with constructors in class definitions. It has been used in some other way.

member initialization requires constant expression

Members in class declarations may only be initialized if the member is static **const** and being initialized by an integral constant. We saw a non-constant expression.

can't use automatic from outer function

When classes local to a function are declared, it is not legal to use the automatic variables or arguments of the function in the class. For example:

```
void f()
{
int I;

class X
{
      int k;
      X()
      {
         k = I;    // illegal, can't use automatic
      }
};
}
```

Cursor Movement Keys

Key(s)	Function
Arrow key	Moves the cursor left, right, up, or down in a field.
End or Ctrl+Right Arrow	Moves to the end of a field.
Home or Ctrl+Left Arrow	Moves to the beginning of a field.
Pape Up or Page Down	Moves up or down in a field, one screen at a time.
Ctrl+Page Up	Moves to the top of a file
Ctrl+Page Down	Moves to the end of a file

Dialog Box Keys

Key(s)	Function
Tab	Moves from field to field (left to right and top to bottom).
Shift+Tab	Moves from field to field in reverse order.
Alt+letter	Moves to the option or group whose underlined letter matches the one you type.
Arrow key	Moves from option to option within a group of options.
Enter	Executes a command button.
	Or, chooses the selected item in a list box and executes the command.
Esc	Closes a dialog box without completing the command. (Same as Cancel)
Alt+Down Arrow	Opens a drop-down list box.
Alt+Up or Down Arrow	Selects item in a drop-down list box.
Spacebar	Cancels a selection in a list box.
	Selects or clears a check box.
Ctrl+Slash	Selects all the items in a list box.
Ctrl+Backslash	Cancels all selections except the current selection.
Shift+ Arrow key	Extends selection in a text box.
Shift+ Home	Extends selection to first character in a text box.
Shift+ End	Extends selection to last character in a text box

Editing Keys

Key(s)	Function
Backspace	Deletes the character to the left of the cursor. Or, deletes selected text.
Delete	Deletes the character to the right of the cursor. Or, deletes selected text.

Help Keys

Key(s)	Function
F1	Gets Help and displays the Help Index for the application. If the Help window is already open, pressing F1 displays the "Using Windows Help" topics. In some Windows applications, pressing F1 displays a Help topic on the selected command, dialog box option, or system message.

Menu Keys

Key(s)	Function
Alt	Selects the first menu on the menu bar.
Letter key	Chooses the menu, or menu item, whose underlined letter matches the one you type, when a menu has focus.
Alt+Letter key	Pulls down the menu whose underlined letter matches the one you type.
Left or Right Arrow	Moves among menus of the main menu bar.
Up or Down Arrow	Moves among menu items within a drop-down menu.
Enter	Chooses the selected menu item.

System Keys

The following keys can be used from any window, regardless of the application you are using.

Key(s)	**Function**
Ctrl+Esc	Switches to the Task List.
Alt+Esc	Switches to the next application window or minimized icon, including full-screen programs.
Alt+Tab	Switches to the next application window, restoring applications that are running as icons.
Alt+PrtSc	Copies the entire screen to Clipboard.
Ctrl+F4	Closes the active window.
F1	Gets Help and displays the Help Index for the application. (See Help Keys)

Text Selection Keys

Key(s)	Function
Shift+Left or Right Arrow	Selects text one character at a time to the left or right.
Shift+Down or Up	Selects one line of text up or down.
Shift+End	Selects text to the end of the line.
Shift+Home	Selects text to the beginning of the line.
Shift+Page Down	Selects text down one window. Or, cancels the selection if the next window is already selected.
Shift+Page Up	Selects text up one window. Or, cancels the selection if the previous window is already selected.
Ctrl+Shift+Left or Right Arrow	Selects text to the next or previous word.
Ctrl+Shift+Up or Down Arrow	Selects text to the beginning (Up Arrow) or end (Down Arrow) of the paragraph.
Ctrl+Shift+End	Selects text to the end of the document.
Ctrl+Shift+Home	Selects text to the beginning of the document.

Window Keys

Key(s)	Function
Alt+Spacebar	Opens the Control menu for an application window.
Alt+Hyphen	Opens the Control menu for a document window.
Alt+F4	Closes the application.
Alt+Esc	Switches to the next application window or minimized icon, including full-screen programs.
Alt+Tab	Switches to the next application window, restoring applications that are running as icons.
Alt+Enter	Switches a non-Windows application between running in a window and running full screen.
Arrow key	Moves a window when you have chosen Move from the Control menu. Or, changes the size of a window when you have chosen Size from the Control menu.