WHAT'S NEW IN C++LINT Last update: 20-Oct-2014

This document details the new and improved features of C++lint 2.0--3.5, including those made possible with new information available from version 9.00L of the PC-lint/Flexelint core engine.

This document is organized by functionality rather than version history, which makes it hard for existing users to see new changes at a glance. So as of version 3.1, we added Section 0, next.

0.0 Update History

Version 3.5:

- GUI: Dramatic improvement in startup time
- GUI: Clicking on message number in Analysis Report brings up a detailed discussion of the analysis message; see Section 1.7.
- IDE: New Rowley Crossworks
- IDE: New support for Visual Studio through 2014
- Core: Powerful support for MISRA 2012 and updated support for MISRA C++ 2008
- Core: 109 new error messages for a total of 1162
- Core: Support for 116 compilers and versions of compilers
- Core: Numerous feature enhancements. With version 3.5, we are adding a companion document, readnewcore.txt, located in your 'doc' subdirectory. Newest features are at the *bottom*. Version 9.00L is the most significant release of the core in some time, so we encourage you to read its section at the bottom of this readme file.

Version 3.1:

• Many key executables have now been compiled for x64 – a significant performance improvement – including a 64-bit PC-lint core, a Cleanscape exclusive. The installer handles the selection of 32- or 64-bit executables for your machine; on Linux there is a script in the bin subdirectory to select 32/64.

1.0 Support for IDEs

With the 3.5 release, we now support

- Visual Studio from 6 through new 2014
- New Rowley Crossworks

With version 3.1 we integrate with four IDEs (many more if you count versions!)

- Visual Studio from 6 through new 2012
- New IAR Embedded Workbench
- New Microchip MPLAB
- New Renesas High-Performance Embedded Workshop (HPEW)

By "integrate", we mean

- We read the project files created by the IDE to obtain the source file list (omitting files manually excluded by the user), include directories, defines/undefines, and any compiler settings which may affect our analysis.
- From this data we build a <project>.Int file to make the analysis exactly track the compilation.
- The analysis can be started from the IDE.
- Results are presented in the output window of the IDE.
- Results are formatted so that double-clicking the analysis message jumps to the "offending" line of source code in the IDE's internal editor

1.1 Support for bypass headers (GUI) or precompiled headers (IDE)

Header processing, especially for large application frameworks, can represent a large – even a majority – of the time spent analyzing the application. Consequently, by managing the "long poles in the header tent", the time savings can be quite substantial.

Bypass headers can greatly improve the speed of your C++lint analysis by physically scanning the specified include (header) files only on their first encounter; the analyzer recreates the effect of these files and their subheaders during subsequent encounters.

Precompiled headers, like stdafx.h in Visual Studio compilations, can be specified in the project's std.lnt file using the -pch() command option; when processed, the C++lint analyzer builds a .lph (lint precomiled header) binary file which is referenced thereafter.

1.2 Inheritance Report – a Cleanscape Exclusive new in version 3.0

This report shows the relationships between classes in the selected C++ source files. It is a hyperlinked graph of the class hierarchy, indicating parents and unconnected nodes.

```
1 Analysis 2 Statistics 3 Inheritance 4 Include Tree
                                           5 Stack
                                                   6 Prototypes
InheritanceRoot
      +-SubClassOne
            1
             +-SubClassTwo
              Additional parent: TwosOtherParent
                    +-SubClassFive
                      Additional parent: Other
                      Additional parent: ThirdParent
Other+
     +-SubClassFive
       Additional parent: SubClassTwo
       Additional parent: ThirdParent
ThirdParent
      1
      +-SubClassFive
       Additional parent: SubClassTwo
       Additional parent: Other
TwosOtherParent
      1
      +-SubClassTwo
       Additional parent: SubClassOne
             1
             +-SubClassFive
               Additional parent: Other
               Additional parent: ThirdParent
Unconnected classes:
SubClassSeven
SubClassSix
```

1.3 Include Tree Report – a Cleanscape Exclusive new in version 3.0

This report demonstrates the nesting structure of the Include files used by the selected C/C++ source file(s). Filenames in angle brackets indicate system include files; these can optionally be excluded from the report. Also optional is the "Redundant Includes" check: a scan of all directories in the Include path to determine if multiple occurrences of the same Include file name are present. This could indicate problems if different users define their Include paths differently. Filenames in red are hyperlinked to open it using the specified external editor.

1 Analysis	2 Statistics	3 Inheritance	4 Include Tree	5 Stack	6 Prototypes	
BEGIN INC	LUDE TREE	1				
Source Fi	le evample	3 con				
SubClass	Two.cop	to.opp				
SubClas	sOne.cpp					
Inheri	tanceRoot.	cpp				
TwosOth	erParent.c	pp				
SubClass	Six.cpp					
SubClas	sSeven.cpp	2				
ThirdPar	ent.cpp					
Other.cp	q					
Source Fi	le fpatch.	.c				
<stdio.h< td=""><td>1></td><th></th><th></th><td></td><td></td><td></td></stdio.h<>	1>					
<crtdef< td=""><td>s.h></td><th></th><th></th><td></td><td></td><td></td></crtdef<>	s.h>					
<sal.h< td=""><td>></td><th></th><th></th><td></td><td></td><td></td></sal.h<>	>					
<sour< td=""><td>ceannotati</td><th>lons.h></th><th></th><td></td><td></td><td></td></sour<>	ceannotati	lons.h>				
<vader< td=""><td>8.n></td><th></th><th></th><td></td><td></td><td></td></vader<>	8.n>					
Zvadef	a h					
<stdlib.< td=""><td>h></td><th></th><th></th><td></td><td></td><td></td></stdlib.<>	h>					
<crtdef< td=""><td>s.h></td><th></th><th></th><td></td><td></td><td></td></crtdef<>	s.h>					
<sal.h< td=""><td>></td><th></th><th></th><td></td><td></td><td></td></sal.h<>	>					
<vadef< td=""><td>s.h></td><th></th><th></th><td></td><td></td><td></td></vadef<>	s.h>					
<limits< td=""><td>.h></td><th></th><th></th><td></td><td></td><td></td></limits<>	.h>					
<crtde< td=""><td>fs.h></td><th></th><th></th><td></td><td></td><td></td></crtde<>	fs.h>					
<sal.< td=""><td>h></td><th></th><th></th><td></td><td></td><td></td></sal.<>	h>					
<vade< td=""><td>fs.h></td><th></th><th></th><td></td><td></td><td></td></vade<>	fs.h>					
<ctype.h< td=""><td></td><th></th><th></th><td></td><td></td><td></td></ctype.h<>						
<crtder< td=""><td>3.12</td><th></th><th></th><td></td><td></td><td></td></crtder<>	3.12					
(Jal.n	is h					
(string	h>					
Kcrtdef	a.h>					
<sal.h< td=""><td>></td><th></th><th></th><td></td><td></td><td></td></sal.h<>	>					
<vadef< td=""><td>s.h></td><th></th><th></th><td></td><td></td><td></td></vadef<>	s.h>					
REDUNDANT	INCLUDE S	SEARCH RESUL	TS			
dimina h		11-	- 2 0) wal in a	Inde \ 1 in	nite h	
<11m102.m	==>c:\pro	ogra~1\micro	s~1.0\vc\inc	lude\lin	mits.h	
Inheritan	ceRoot.cpp	==>c:\progr	a~1\cleans~1	\lpp\ex	amples\incs	ab~1\inheri~1.cpp
			a.r/orcans.r	(Thb (cw	ampres (rime.	11-1.0pp
SubClassO	ne.cpp==>c ==>c	:\progra~1\ :\progra~1\	cleans~1\lpp cleans~1\lpp	\example \example	es\incsub~1 es\subcla~3	<pre>\subcla~1.cpp .cpp</pre>
			0.01.001.0001	and a base of		
<ctype.n></ctype.n>	==>c:\prog	ra~1\micros	~1.0\vc\incl	ude\cty	pe.n pe.h	
<string.h< td=""><td>>==>c:\pro</td><th>ogra~1\micro</th><th>s~2.0\vc\inc</th><td>lude\st:</td><td>ring.h</td><td></td></string.h<>	>==>c:\pro	ogra~1\micro	s~2.0\vc\inc	lude\st:	ring.h	
	==>c:\pro	ogra~1\micro	s~1.0\vc\inc	lude\st:	ring.h	
<crtders.< td=""><td>n>==>c:\pi</td><th>rogra~1\micr</th><th>03~2.0\VC\1n</th><td>clude (c</td><td>rtdeis.h</td><td></td></crtders.<>	n>==>c:\pi	rogra~1\micr	03~2.0\VC\1n	clude (c	rtdeis.h	
	==>c:/pi	rogra~1\micr	03~1.U\VC\1D	clude/c:	rtdeis.n	
<stdlib.h< td=""><td></td><th>ara~1\micro</th><th>s-2.0\vc\inc</th><td>lude\st</td><td>dlib.h</td><td></td></stdlib.h<>		ara~1\micro	s-2.0\vc\inc	lude\st	dlib.h	
CO GALLO	==>c:\pro	gra~1\micro	s~1.0\vc\inc	lude\st	dlib.h	
		gan a (mara		2000 (00)		
<stdio.h></stdio.h>	==>c:\prog	gra~1\micros	~2.0\vc\incl	ude\std:	io.h	
	==>c:\prog	gra~1\micros	~1.0\vc\incl	ude\std:	io.h	
<sal.h>==</sal.h>	>c:\progra	a~1\micros~2	.0\vc\includ	e\sal.h		
	>c:\progra	~1\micros~1	.0\vc\includ	e\sal.h		
dan mentana a	dalla i	1	1		1	1-1
<swprintf< td=""><td>.101>==>C:</td><th>\progra~1\m</th><th>10108~2.0\V0</th><td>linclud</td><td>e swprintr.</td><td>ini ini</td></swprintf<>	.101>==>C:	\progra~1\m	10108~2.0\V0	linclud	e swprintr.	ini ini
	==>0;	/progra~1/m	10108~1.0/00	/Include	e (swprintr.)	1111
<vadefs.h< td=""><td>>==>c:\pro</td><th>gra~1\micro</th><th>s~2.0\vc\inc</th><td>lude\va</td><td>defs.b</td><td></td></vadefs.h<>	>==>c:\pro	gra~1\micro	s~2.0\vc\inc	lude\va	defs.b	
1999 St. 1995 C. 1997 C.C.	==>c:\pro	ogra~1\micro	s~1.0\vc\inc	lude\va	defs.h	

1.4 Stack Usage Report – a Cleanscape Exclusive

This report is a tabular list of auto storage and stack requirements for each function in your project; it is very useful for embedded systems development where the amount of stack (and overall memory) can be mission critical.

1 Analysis 2 Statistics	3 Stack	4 Proto	otypes	
Function Name	Auto	Stack	Category	Calls
assemblytest	16	16	finite	<none></none>
catchme	8	8	finite	<none></none>
catchme2	12	12	finite	<none></none>
guardme	16	16	finite	<none></none>
interfunc1	8	8	finite	<none></none>
interfunc10	24	44	finite	interfunc11
interfunc11	12	12	finite	<none></none>
interfunc2	12	28	finite	interfunc1
interfunc3	0	0	finite	<none></none>
main	12	0	calls_recursive	recursivefunc2
memory	20	60	finite	malloc
pointer	24	24	finite	<none></none>
recursivefunc1	8	0	recursive_loop	recursivefunc1
recursivefunc2	8	0	calls_recursive	recursivefunc1
returnvalue	12	12	finite	<none></none>
showall	28	68	finite	malloc

1.5 Statistics/Summary Report – a Cleanscape Exclusive

This report is the melding information from several sources to produce a report with multiple categories, as shown in the sample on the next page:

1 Analysis 2 Statistics 3 Stack 4 Prototypes Analysis options used to generate this report were: +fdi +macros +linebuf +linebuf -width(0) -zero(400) -ic:\progra~1\cleanscape\cpplint\main\lbin\lf co.lnt -iC:\PROGRA~1\MICROS~1.NET\Vc7\include -u -summary(c:\progra~1\cleanscape\cpplint\main\reports\cpplint.stt) +stack(&file=c:\progra~1\cleanscape\cpplint\main\reports\cpplint.stk) -od100(c:\progra~1\cleanscape\cpplint\main\reports\cpplint.pro) -wlib(1) -w2 -passes(2) +byph(common.h) -vf -d" WIN32" -"format stack=%f\t%a\t%n\t%t\t%c" +e974 +program info(output prefix=c:\progra~1\cleanscape\cpplint\main\reports\cpplint) C:\temp\benchmark files.lnt >>> Statistics: Number of source files: 1 Number of files processed: 4 Total number of symbols: 135 110 auto static 3 function 18 member_function 0 instance member 0 enumerator 0 type 0 label 0 class template 0 function template 0 namespace 1 template parameter 0 using declaration 0 <indeterminate> 3 Individual message summary ------Count Type Number Text Expecting _ 4 Error 10 Likely use of null pointer '___' in ___argument to operator ' 1 Warning 413 Custodial pointer '___' (___) has not been freed or returned 16 Warning 429 4 Warning 438 Last value assigned to variable '___' (___) not used 2 Warning 449 Pointer variable '___' previously deallocated Highest operation, ____', lacks side-effects 15 Warning 522 2 Warning 527 Unreachable code at token ' Symbol '___' (___) not subsequently referenced Symbol '___' (___) not initialized 2 Warning 529 4 Warning 530 function '___' should___ return a value (see 1 Warning 533) 11 Warning 534 Ignoring return value of function '___' (compare with Symbol '___' (___) not accessed variable '___' depends on order of evaluation 2 Warning 550 Symbol ' 2 Warning 564 Custodial pointer '___' (___) possibly not freed or returned 4 Warning 593 Possible use of null pointer '___' in ___argument to operator 22 Warning 613 5 Variable '__' (__) may not have been initialized Warning 644 1 974 Worst case function for stack usage: ____ See +stack for a full Note Errors Warnings Notes Info ---- ----- ----- ----1 0 4 93 Total messages: 98

1.6 Functional upgrades: IDE

- In version 3.1 you can now lint an entire Visual Studio solution.
- Also in version 3.1, better integration with GUI for preparing inheritance, include tree, stack usage, and statistics reports, and added to the VS Tools menu.
- The std.Int file created by the C++lint integration into Visual Studio includes a -pch(stdafx.h) line (commented by default).
- The "Open .lnt" tool in Visual Studio now opens project. Int in addition to std.lnt.
- Icon added for 1-button creation of IDE std.lnt file from the Cleanscape GUI:



- Visual studio indirect files (.1nt) reordered for improved operation.
- Streamlined handshakes between GUI, VS integration mode, and core PC-lint engine.
- A number of small bug fixes, including VS integration when installing and not "owner" of the machine and VS tool removal.

1.7 Functional upgrades: GUI

- Greatly improved startup performance in version 3.5.
- Projects within the Cleanscape GUI may now have their own project. Int file, which may be used to
 - refine or override some of the GUI controls,
 - add PC-lint commands such as -strong() not controlled from the GUI, and
 - supercede default options for the Target System options specified in associated the co-xxx.lnt file

The *project.lnt* file is created by clicking the View... button on the Misc Options tab, which also opens the co-xxx.lnt file in the user's specified External Editor.



- Some hard-coded controls now made into checkboxes (e.g., number of analysis passes, whether to search source directories for double-quoted header files see sample screen in 1.0 intro).
- System ("library") include directories added by C++lint are now also flagged as such with +libh() PC-lint directive.
- Icon added for 1-button creation of IDE/command line std.lnt file from the Cleanscape GUI:



- May now list source code in the analysis report; great for documentation.
- Numerous visibility enhancements and small bug fixes



• In version 3.5, clicking on an error message *number* (as opposed to the *line number* causing the message) opens a detailed description of the error message in your system browser. So for example, if you clicked on Warning 429 in the GUI's Analysis Report, your system browser would open up to Message 429 like this:

-	C B file ((i)) (Teasscrape (conject) dec type	der html#429		- DurkDurkfor	The lot in a set in the set of the		
	C B mey//c/cleanscape/cp/modoc/cne	0.3.1010-425		Contraction	Chief Line and Annual Annual		
79	Custodial pointer 'Symbol' (Location) has not	A pointer of auto storage class was al	located storage which was	neither freed nor returned to the	caller. This represents a "memory		
	been freed or returned	leak". A pointer is considered custodial if it uniquely points to the storage area. It is not considered custodial if it has been copied. Thus:					
		int *p = new int[20]; // p is a custodial pointer					
		<pre>int *q = p; // p is no longer custodial</pre>					
		<pre>p = new int[20];</pre>					
		q = p + 0;	// p remains custodial				
		Here p does not lose its custodial property by merely participating in an anthematic operation.					
		A pointer can lose its custodial property by passing the pointer to a function. If the parameter of the function is typed pointer to const or i the function is a library function, that essumption is not made. For example					
		<pre>p = malloc(10);</pre>					
		strcpy (p, "hello");					
		Then p still has custody of storage allocated.					
		It is possible to indicate via semantic options that a function will take custody of a pointer.					
0	Character "@", token to specify variable location, is not standard C/C++	Many compilers for embedded system example: int × B0x2000; specifies that variable x is actually loc	is have a declaration syntax ation 0x2000. This message	that specifies a location in place is a reminder that this syntax is i	of an initial value for a variable. For non-standard (although quite common		
-		If you are using this syntax on purpose, suppress this message.					
1	number Integer	A template object parameter (as oppo	osed to a type parameter) w	as not provided with an identifier	, was this an oversight?		
432	Suspicious argument to malloc	The following pattern was detected:					
		<pre>malloc(strlen(e+1))</pre>					
		where e is some expression. This is suspicious because it closely resembles the commonly used pattern:					
		<pre>nalloc(strlen(a)+1)</pre>					
		If you really intended to use the first pattern then an equivalent expression that will not raise this error is:					
		nalloc(strlen(e)-1)					
433	Allocated area not large enough for pointer	An allocation was assigned to a point library allocation routines such as mal	er whose reach extends bey loc and calloc. For example	ond the area that was allocated.	This would usually happen only with		
		<pre>int *p = malloc(1);</pre>					
		This message is also provided for user-declared allocation functions. For example, if a user's own allocation function is provided with the following semantic:					
		-sem(ouralloc,@Pmall	loc(1n))				

So to summarize mouse clicking in the Reports window:

- If you click on any message number (colors vary), your system browser will open the error message file so you can view a detailed description of the error.
- If you click on a source line number in bright red, that source line will open in your favorite code editor for immediate review/ correction!

1.8 Functional upgrades: Command line

• Icon added for 1-button creation of command line/IDE std.lnt file from the Cleanscape GUI:



• That save now captures files you select during the GUI session, speeding up the process of setting up a command line analysis project. A sample std.lnt file is shown below.

```
// *** This file created by Cleanscape C++lint GUI on 21-Nov-08 ***
// Refer to the PC-Lint Users Manual for details on control settings.
+fdi
+macros
 +linebuf
+linebuf
-width(0)
  zero(400)
 -ic:\progra~1\cleanscape\cpplint\main\lbin\lf
 // Unconnent the next line(s) for connand line (non-IDE) operation:
// co- msc71. lnt
//- - iC: \PROGRA~1\MICROS~1. NET\Vc7\include
//+libdir(C: \PROGRA~1\MICROS~1. NET\Vc7\include)
- summary(c: \progra~1\cleanscape\cpplint\main\reports\cpplint.stt)
+stack(&file=c: \progra~1\cleanscape\cpplint\main\reports\cpplint.stk)
- wlib(1)
- w2
 passes(2)
+byph(comon. h)
- d_WIN32
// Add any additional PC-lint controls next:
// For command line operation, uncomment the sourcefile names below.
// additional files as necessary; enclose in double quotes if name h
//C: \PROGRA~1\cleanscape\cpplint\examples\example1.c
//C: \PROGRA~1\cleanscape\cpplint\examples\example2.c
                                                                                                                                Add
                                                                                                                           has spaces.
```

- Operational improvements in the cpplint.bat control program.
- Extensive updates to the documentation and the Online Help system. Since Online Help provides important operational data on each GUI function and it associated PC-lint command, Online Help has been formatted as a .pdf document for reference/review.

1.9 Fully automated gcc/g++ compiler setup

Since gcc and g++ are custom-built at each site, it is not possible to know beforehand all the defines, macros, include directories, etc., specified during the build. Cleanscape provides an *exclusive* tool (external program setupgcc) to automate extraction of this information and place it in helper (.h and .lnt) files later consumed by C++lint.

The result? A comprehensive analysis of code intended for gcc/g++ with no false positives!

A setupgee session on Linux is shown next; user input has been colored green.

Redhat: /usr/local/cleanscape/bin# setupgee
This program extracts unique settings (internal defines/macros and default
include directories) for gcc/g++ on this system for use by C++lint. An
interactive command line session confirms the correct compiler, gathers
the info, and stores it. If your gnu compiler is rebuilt, then force
this program to retrieve new data by running setupgec -FORCE
Both gcc and g++ found; which do you use (gcc/g++)? G++
To get the right predefined macros, enter the compiler options you specify
during builds, or <ENTER> to accept the default in parentheses below.
(-0)
Built /hone/lint/build/unixbcsi/prod/cpplintgui.dir/main/lbin/lf/mygpp.lnt
and /home/lint/build/unixbcsi/prod/cpplintgui.dir/main/lbin/lf/mygpp.h.
Command line: Add 'co-gpp.lnt' as a parameter when running 'cpplint'.
GUI: Select 'g++' in the Target System dropdown.
To test your configuration, try the command line or GUI on file
'testcfg.cpp' located in the examples subdirectory.
Press <ENTER> to exit...

1.10 Semi-automated process for adding new compilers to C++lint

If gcc/g++ integration can be automated, why not *any* compiler (namely, those that are not yet integrated into the analyzer)? It turns out that we can do a great deal of automation in this regard, although ultimately the solution depends on experience with the compiler and the lint analysis of source code intended for that compiler.

Cleanscape provides an *exclusive* tool (scavenge) that extracts the built-in macros for your compiler, then prepares a template co-name.lnt file with descriptive section headers for you to fill in other important data about your compiler, and finally provides a mechanism to input your resulting compiler options file into the Cleanscape GUI. A sample scavenge session is shown next; user input has been colored green.

C:\cleanscape\cpplint\bin> scavenge
This program extracts built-in macros for compilers not yet supported in
C++lint. An interactive command line session obtains compiler basics,
gathers the info, and creates a co-xxx. Int template which you later edit.
In this interactive session, you will need the following data:
 The full pathname of your compiler
 The full pathname to or the compiler options for an internal preprocessor
 The compiler options specified during builds (or within makefile)
 The preprocessor options to output to a mamed file
 The directories containing the compiler's header files (the ones specified
 using angle brackets in your #include statements)
If you don't have this info, hit <CTRL-C> now to exit and rerun later, or
press <ENTER> to continue...
Great! Let's get started.
We need a short label name for your compiler which will be used to name your
co-xxx. Int compiler options file would be named 'co-GH2960. Int'.
Enter a label name for your compiler: BCC55
Is your project C or C++? (c/cpp): cpp
Enter the full pathname of your compiler: c:\borland\bcc55\bin\bcc32, exe

Now we need some preprocessor information. Is it a unique program or just compiler switches? (P/S): p Enter the full pathname of the preprocessor: c:\borland\bcc55\bin\cpp32.exe Now enter the preprocessor options to Enable preprocessor output, if necessary.
Disable any source line info, if necessary.
Output to a specified file; use the word FILE place the filename. If the preprocessor only outputs to 'stdout', type the word STDOUT. EX. 1: Microsoft. /EP STDOU EX. 2: Borland. -P- - oFILI EX. 3: Generic Unix. -P STDOUT /EP STDOUT -P- - oFILE Enter the preprocessor options: - P- - oFILE Next, we need the directories containing the compiler's include files (the 'system header files'). If already present in the INCLUDE environment variable, type just the single word INCLUDE; type DONE when done. NOTE: The contents of INCLUDE env. Var. are Excluded by default, so you must specify the word INCLUDE if you want its contents. Enter dir path, INCLUDE, or DONE: c:\borland\bcc55\include c:\borland\bcc55\include added OK. Enter dir path, INCLUDE, or DONE: done Enter the compiler options you specify during builds (e.g., as specified in your makefile), or press <ENTER> to accept the default in parentheses below. (-0) Obtaining built-in macros; this may take a couple minutes...done. Processing macro list...done. Cleaning up... done. Building co-BCC55. 1nt... done. Adding bcc32.exe to GUI Target System dropdown...done. To renove it later, delete the BCC55 line from file C:\PROGRA~1\cleanscape\cpplint\bin\myconpiler.lst Now opening in Notepad; add data for each section as described in the comments. Now you are ready to test your configuration:
Add co-BCC55. Int on the command line - OR-Select BCC55 in the GUI Target System dropdown.
Specify '-wl' (command line) - OR- select 'Errors_only' (GUI).
Specify '+fdh' (command line) - OR- select 'Append . h' (GUI) if your compiler uses the same . h header file for both C and C++.
Use input file 'testcfg.cpp' located in the examples subdirectory.

Press <**ENTER**> to exit...

The co-BCC55.lnt file resulting from the above session is shown below:

- 🗆 × morePLUS - co-BCC55.Int - Line 1 (100%) co-BCC55.Int - Compiler options file for bcc32.exe Template by: Cleanscape Software Date: 30 Nov 2008 Author: <your name here> // This file can be used to create a PC-lint compiler options file for // compilers not yet standardized at Gimpel. // A similar file is created when you run program scavenge (located in the bin
// subdirectory); scavenge will fill in the macro section for you. // Each comment below is a section header of required information; when filled // in, you will get robust and accurate analyses using your new compiler! See // any of the existing co-xxx.lnt files in main/lbin/lf as examples. // Section 1: include GENERIC co.lnt options file co.lnt // Section 2: Compiler/machine architecture data sizes and alignment // Format: -sb# /* &etc. Numerous; see section 5.3 of the manual */ // Section 3: Exclude/include compiler reserved words
// Format: -/+rw(word1, word2...) /* See section 5.7 of manual */ // Section 4: Simple defines (e.g., machine architecture; MACROS -> Sec. 9) // Format: -dname[=value] /# See sec. 5.7; NO space after -d #/ // Section 5: Flag options (e.g., wprintf formats, scoping, header naming)
// Format: -/+f(op) /* See section 5.5 of manual */ Section 6: Exclude/include error messages Formats: -/+e# -/+esym(#,sym1,sym2...) -/+elib(#) /* See section 5.2 of manual */ // Section 7: System (library) Include directories supplied with compiler // Format: -ipath /* See sec. 5.7; NO space after -i */ // +libdir(path) /* Controls analysis depth with -wlib() */ -ic:\borland\bcc55\include +libdir(c:\borland\bcc55\include) // Section 8: Miscellaneous controls, e.g., -function(), -wprintf(), -/+v, -\$ // Section 9: Macro definitions - extracted automatically by scavenge program -d_FLAT__{1} -d_UCHAR_T(5) -d_UCHAR_T_DEFINED() -d_cplusplus(1) -d_cplusplus(1) -d_LX86(300) -dALX86(300) -dALX86(300) -dALX86(300) -d_SHPOP_SUPPORTED() -d_TURBOC__{0x0551} -d_CGVER__{0x0200} -dalloca(alloca) C__TURBOC__(0x0551) C__CGVER__(0x0200) dalloca(alloca) dINTEGRAL_MAX_BITS(64) dSTDCALL_SUPPORTED() dstromp(stromp) dstropy(stropy) -dwIN32(1) -d_WIN32(1) -d_WIN32__(1) -d_WIN32__(1) -d_BORLANDC__(0x0551) -d_BORLANDC__(0x0551) 100% Fwd Back ## Top End n> n< Num preserVe tAb Quit Help More->

WHAT'S NEW IN PC-LINT COMMAND LINE (CORE) PROGRAM

As of version 3.5, we only hit the highlights of new features and instead include the readme file from Gimpel as to the changes in the core analyzer. This file is named readnewcore.txt and is located in your 'doc' subdirectory. It is organized by release, with the latest release notes at the *bottom*.

Version 9.00L is the most significant release of the core in some time. You are encouraged to read its section at the bottom of readnewcore.txt and is located in your 'doc' subdirectory.

2.1 Major New Features

- Pre-compiled headers, as users of C and C++ systems are well aware, can dramatically reduce the time spent in processing multiple modules. See Section 7.1 "Pre-compiled Headers".
- We now incorporate variables of static storage duration in our value tracking. These include not only variables that are nominally static, as local to a function and local to a module, but also external variables. See –static_depth(n) and flag –fsv.
- We examine multi-threaded programs for correct mutex locking and report on variables shared by multiple threads that are used outside of critical sections. See Chapter 12. Multithread Support
- Stack usage: We can report (Note 974) on the overall stack requirements of any program whose function calls are non-recursive and deterministic (i.e. calls not made through function pointers). This is very useful for embedded systems development where the amount of stack required can be mission critical. A complete detailed report of stack usage for each function is available as well (See +stack() in Section 5.7 Other Options).
- We now support through the strong type mechanism the classical dimensional analysis that engineers and physicists have traditionally employed in verifying equations. A 'type' can now be a ratio or product of other types and the compound types are checked for consistency across assignment boundaries. See Section 9.4 Multiplication and Division of Strong Types.
- The user may deprecate particular symbols in any of the following categories: function, keyword, macro and variable. See -deprecate.
- Message Enhancement and Control See readnewcore.txt, located in your 'doc' subdirectory.
- Enhanced MISRA Checking

Significantly enhanced support for MISRA C3 (MISRA C 2012). A number of new messages to support MISRA C3 have been added and the au-misra3. Int author file has been updated accordingly. We now provide some level of support for all but 3 of the decidable rules and 4 of the undecidable rules, with full support being provided for the vast majority of the rules.

We have broken out the 1960/1963 diagnostics (used to support MISRA C++) into individual messages to provide a greater level of control and flexibility, especially with regards to message suppression. The 1960/1963 messages will continue to be supported for backwards compatibility.

A new author file, au-misra-cpp-alt.lnt, can be used to take advantage of the new messages. **Cleanscape note:** The GUI uses this new author file for MISRA_CPP_2008.

A comprehensive collection of information about your program is optionally provided yielding information on files, types, symbols and macros for simple viewing or in a manner absorbable by a database or spreadsheet. This information can be used for many purposes, including naming-style conventions. See +program_info.

Cleanscape note: The GUI supplies this information in the Statistics Report.

• Macro Scavenging: This feature turns PC-lint/FlexeLint into a seeker of built-in macros supported by a compiler and lying about within compiler header files. This is perfect for the unknown compiler with long and forgotten macros ready to trip up a third party processor such as PC-lint/FlexeLint. See

-scavenge. **Cleanscape note:** This is the basis for the Cleanscape-exclusive scavenge program.

- We now provide significantly improved support for VS2012 through an updated co-msc110.lnt file, available on our patches page.
- Patch 9.00L introduces support for range-based for statements, the 'final' class specifier, standard layout types, improved handling of null pointer constants, significantly improved handling of C++11 core language features used in Visual Studio 2012 headers. As with other C++ 2011 features, these are available using the option: -A(C++2011).
- We now support binary literals and digit separators from C++14. These features are available using the option: -A(C++2014).

2.2 New Messages of Note

See file readnewcore.txt, located in your 'doc' subdirectory.

2.3 New Error Inhibition Options – See Section 5.2 Error Inhibition Options

See file readnewcore.txt, located in your 'doc' subdirectory.

2.4 New Verbosity Options – See Section 5.4 Verbosity Options

See file readnewcore.txt, located in your 'doc' subdirectory.

2.5 New Flag Options – See Section 5.5 Flag Options

See file readnewcore.txt, located in your 'doc' subdirectory.

2.6 New Message Presentation Options – See Section 5.6.1 Message Height Options

and 5.6.3 Message Format Option

See file readnewcore.txt, located in your 'doc' subdirectory.

2.7 Additional Other Options – See Section 5.7 Other Options

See file readnewcore.txt, located in your 'doc' subdirectory.

2.8 Compiler Adaptation – See Section 11.1.1 Special Functions See file readnewcore.txt, located in your 'doc' subdirectory.

2.9 New Messages See file readnewcore.txt, located in your 'doc' subdirectory.