PC-lint for C/C++ v7.5

Features

Platforms:

***PC-lint*:**

- Windows 95 / Windows NT
- DOS (built-in DOS extender)
- OS/2 (32 bit)

***FlexeLint*:**

- Unix and Unix-like platforms (AIX, HP-UX, Sun OS, Solaris, Ultrix, SCO, GNU, etc.)
- VAX VMS
- IBM's VM, MVS
- OS-9
- virtually any platform supporting C

Compatibility:

- supports K&R C, ANSI C, ANSI/ISO C++
- explicit support for Borland, Microsoft, GNU and most other major compilers and libraries
- support for most major embedded-system compilers including bit addressing.
- numerous options to support rogue compilers
- scalars sizes can be specified for cross-compiling

Message Suppression:

- by number
- by number and symbol (including wild cards)
- one-line suppression
- by macro
- for library headers, by number (a header is library depending on how it is included; this can be overridden via user options)
- for specified functions, by number
- for expressions

Flexibility:

- indirect files (nested to any depth) can contain filenames, options, environment variables
- format of error messages can be customized to support a wide variety of editors/IDE's
- all options can be embedded in user code

Special Checking Facilities:

- optional strong type checking (`typedef`-based) with a rich option set to detect nominal type differences. You can even form a fully checked type hierarchy of scalar types using only `typedef`
- checks flow of control for possibly uninitialisedvariables.
- value tracking to detect subtle initialization and value misuse problems
- with value tracking as an enabling technology, we support 'semantics' checking for almost 100 library functions, this checking can be extended to user functions (see function mimicry)
- user-defined semantic checking for function arguments and return values
- find unused macros, `typedef`'s, classes, members, declarations, etc. across the entire project (see weak definials)
- other special torture tests

Performance:

- fast one-pass operation
- robust - tables will expand as needed to handle large applications

Representative Checks

PC-lint/FlexeLint Version 7.50 has more than 700 error messages. For detailed information, browse through the ASCII file, **`msg.txt`** which is a listing of all the error messages that PC-lint/FlexeLint will generate. If you prefer you can download **`msg.zip`** (PKZipped File of `msg.txt`)

PC-lint/FlexeLint will detect - For C++

- order of initialization dependencies
- class members not initialized by constructor
- pointer members not deleted by destructors
- base class destructors that are not virtual
- names hiding other names
- improperly formed or missing assignment operators and copy constructors
- missing destructors from classes using dynamic allocation
- out-of-order constructor initializers
- creation of temporaries
- undefined and unreferenced class members initialization of a non-`const` reference with a non-lvalue
- assignment operator not first checking for assignment to `this`
- inconsistent use of `extern "C"`
- `operator delete` not checking argument for `NULL`
- static variables in in-line functions in headers
- exposing privileged data
- failure to copy a base class, or to use the base class copy constructor
- failure to assign members and base classes
- issuing `throw` within a destructor
- assignment of an array to a base class pointer
- inconsistent or incomplete exception specifications
- failure to reference a virtual member function
- a virtual function with a default parameter
- redundant access specifiers
- binary operators that should be non-member functions or that return references, or that shouldn't be user defined or operators that should be defined
- function parameters that could be declared `const` reference
- ill-defined increment and decrement operators
- `catch` parameters that are not references
- An examination is made of all the base class hierarchies in the entire project to determine non-virtual classes included twice, or virtual classes not included twice in any class hierarchy.

PC-lint/FlexeLint will detect - For C and C++ ...

- from value tracking information we can detect under many circumstances:
  - ⇒ use of `NULL` pointer in unary `*` or `->`
  - ⇒ creation and access of out-of-bounds pointers
  - ⇒ subscript out-of-bounds
  - ⇒ division by zero
  - ⇒ passing `NULL` pointers to selected library functions
  - ⇒ data over-run conditions on selected library functions
  - ⇒ Booleans that always evaluate true or evaluate false
  - ⇒ inappropriate deallocation
  - ⇒ memory leaks
  - ⇒ unusual values passed to functions based on user-defined semantic specifications
- from a special macro scan we can find
  - ⇒ passing an expression to an unparenthesized macro parameter
  - ⇒ passing an expression with side effects to a repeated macro parameter
  - ⇒ unparenthesized expression-like macros
- intermodule type inconsistencies
- uninitialized variables (auto, `static` and global scalars, arrays and `structs`)
- unused variables and functions
- assigned but not accessed variables (including globals)
- unreachable code
- unusual expressions such as: `flags & 4 == 0` (precedence error)
- constant Booleans as in: `if( x = 0 ) ...`
- indentation checking
- suspicious use of semi-colons as in `if( a > b );` not followed by `else`
- strict and loose enumeration checking
- `printf-scanf` format checking
- order of evaluation errors as in: `a[i] = i++;`
- unsigned comparisons with `0`
- wide variety of loss of precision errors such as `int` to `char` featuring our exclusive precision tracking
- excessive shift values
- loss of sign
- suspicious cast
- mixed signed and unsigned quantities
- comments within comments
- unused compile time objects, including macros, `typedef`'s, declarations, `class`'es, `union`'s, `enum`'s
- ANSI quiet changes
- unused headers
- returning pointers to `auto` addresses and assigning auto address to `static`
- externals that can be made `static` and hence hidden
- declarations that can be offloaded from headers
- name clashes within the first *count* characters
- strong type checking based on `typedef` types.
- possibly uninitialised variables based on flow of control.
- overflow while processing arithmetic constants (E.g. for 16 bit integers, `200*200` overflows)
- constant expressions that reduce to zero
- suspicious truncations
- suspicious loss of fraction
- initialization irregularities (too few, too many, incorrect shape, string concatenations in)

Version 7.50 Designer's Notes

“Ever since we've added our much heralded value tracking to our arsenal of program analysis and since we've endowed about 100 built-in functions with special checking of arguments and return value deductions, many of our users have requested a more general facility so that user functions, and entire libraries, conventional and class libraries, may be similarly checked. Version 7.50 responds to these petitions by providing a complete but familiar language (the language of C expressions) to specify constraints for arguments and return values. To provide problem-oriented constraint specifications, all forms of C constant manifests are supported including macros, enumeration's, and const variables. The result is both powerful and flexible (but what else would you expect?)”. See example f.cpp

“We have greatly expanded our pointer checking; noting carefully the origins of pointer values (new, malloc, address of auto , increment, etc.) so that obvious incompatibilities can be spotted and such menaces as the inappropriate deallocation and the dreaded memory leak can be caught early and dealt with appropriately”.

“One reviewer (Scott Meyers as it turns out) discovered that we were deficient in detecting anomalies heralded by a prominent C++ authority. Red-faced we increased our C++ sensitivity training, and implemented a whole new batch of subtle defect checks based on available C++ literature. This suite of new checks, all by itself, makes the upgrade worthwhile.”

ex.cpp - How many bugs can you find in this program?

How many can your compiler find?

```
1   #include <string.h>
2   #include <stdlib.h>
3   #include <stdio.h>
4
5   #define Extract(ch) (ch) & 0xFf
6   #define Value(ch) ((ch) > '0' && \
7           (ch) <= '9' ? (ch) - '0' : 0)
8   #define Abs(x) ( (x) < 0 ? -x : (x) )
9
10  void readline( char *fn )
11      {
12      FILE *f;
13      char buf[100];
14
15      if( !fn ) printf( "bad file\n" );
16      f = fopen( fn, "r" );
17      (void) fgets( buf, 101, f );
18      fclose( f );
19      }
20
21  int compute( char *s )
22      {
23      int sum = 0;
24      while( *s )
25          sum = sum + Value(Extract(*s++));
26      return Abs( sum - 100 );
27      }
28
29  class String
30      {
31    private:
32      char *a;
33      unsigned len;
34    public:
35      String( char *s = 0 )
36          {
37          if( s )
38              {
39              len = strlen(s);
40              a = new char[len];
41              strcpy( a, s );
42              }
43          }
44      ~String() { len = 0; }
45      String( const String & );
46      String & operator=( const String &s )
47          {
48          len = s.len;
49          a = new char[len];
50          memcpy( a, s.a, len );
51          return s;
52          }
53     };
```

OK - What’s Wrong?

ex.cpp - Lint Output

```
--- Module:   ex.cpp
                           _
#define Extract(ch) (ch) & 0xFf
ex.cpp(5) : Info 773: Expression-like macro 'Extract' not parenthesized
                        _
    f = fopen( fn, "r" );
ex.cpp(16) : Warning 668: Possibly passing a null pointer to function
    fopen(const char *, const char *), arg. no. 1
                               _
    (void) fgets( buf, 101, f );
ex.cpp(17) : Warning 668: Possibly passing a null pointer to function
    fgets(char *, int, struct _iobuf *), arg. no. 3
ex.cpp(17) : Warning 419: Apparent data overrun for function fgets(char *, int,
    struct _iobuf *), argument 2 exceeds argument 1
               _
    fclose( f );
ex.cpp(18) : Warning 668: Possibly passing a null pointer to function
    fclose(struct _iobuf *), arg. no. 1
                                       _
        sum = sum + Value(Extract(*s++));
ex.cpp(25) : Warning 666: Expression with side effects passed to repeated
    parameter 1 in macro Value
                          _
    return Abs( sum - 100 );
ex.cpp(26) : Warning 665: Unparenthesized parameter 1 in macro Abs is passed an
    expression

                          _
            strcpy( a, s );
ex.cpp(41) : Warning 668: Possibly passing a null pointer to function
    strcpy(char *, const char *), arg. no. 1

        _
        }
ex.cpp(43) : Warning 1541: member String::a (line 32) possibly not initialized
    by constructor
ex.cpp(43) : Warning 1541: member String::len (line 33) possibly not
    initialized by constructor

                         _
    ~String() { len = 0; }
ex.cpp(44) : Warning 1540: pointer member String::a (line 32) neither freed nor
    zero'ed by destructor

                         _
        a = new char[len];
ex.cpp(49) : Warning 423: Creation of memory leak in assignment to
    'String::a'

                             _
        memcpy( a, s.a, len );
ex.cpp(50) : Warning 668: Possibly passing a null pointer to function
    memcpy(void *, const void *, unsigned int), arg. no. 1

                _
        return s;
ex.cpp(51) : Warning 605: Increase in pointer capability (return)

        _
        }
ex.cpp(52) : Warning 1529: 'String::operator=(const String &)' not first
    checking for assignment to this

    --- Wrap-up for Module: ex.cpp

Info 753: local class String (line 29, file ex.cpp) not referenced
Info 754: local structure member String::String(const String &) (line 45, file
    ex.cpp) not referenced
Info 766: Header file 'c:\msdev\include\stdlib.h' not used in module 'ex.cpp'
```

Options Summary

Error Inhibition Options

(**-** inhibits and **+** enables messages)

-/+**e***#*	message number(s) #
-e(*#*[,*#*] ...)	message number(s) # for the next expression
--e(*#*[,*#*] ...)	message number(s) # for the current expression
-/+**eai**	argument sub-Integer
-/+**ean**	arguments differing nominally
-/+**eas**	arguments same size (different type)
-/+**eau**	arguments differing signed-unsigned
-/+**efile(***#,file***)**	control message by number, filename
-/+**efunc(***#,Symbol***)**	control message(s) within a function
-/+**elib**(*#***)**	control message(s) in library headers
-/+**elibsym(***#***)**	control message(s) for a library symbol
-/+**emacro(***#,Symbol***)**	control message(s) within a macro
-/+**epn**	pointer to nominally different types
-/+**epnc**	pointer to chars nominally different
-/+**epp**	pointer vs. pointer (same size)
-/+**eps**	pointer to same-size
-/+**epu**	pointer to unsigned (vs. signed)
-/+**epuc**	pointer to signed/unsigned chars
-/+**esym(***#,Symbol***)**	control message by number, symbol
-/+**etd(***TypeDiff***)**	ignore certain type differences
!e*#*	one-line error suppression
-w*level*	set warning level
-wlib(*level***)**	set warning level for library

Size Options

-sb*#*	number of bits in a byte
-sc*#*	**sizeof(char)** becomes #
-slc*#*	**sizeof(long char)** becomes #
-ss*#*	**sizeof(short)** becomes #
-si*#*	**sizeof(int)** becomes #
-sl*#*	**sizeof(long)** becomes #
-sf*#*	**sizeof(float)** becomes #
-sd*#*	**sizeof(double)** becomes #
-sld*#*	**sizeof(long double)** becomes #

-sll#	**sizeof(long long)** becomes #
-sp#	size of all pointers becomes #
-spD#	size of Data pointer becomes #
-spP#	size of Program pointer becomes #
-spN#	size of **near** pointer becomes #
-spF#	size of **far** pointer becomes #
-spND#	size of **near** Data pointer becomes #
-spNP#	size of **near** Program pointer becomes #
-spFD#	size of **far** Data pointer becomes #
-spFP#	size of **far** Program pointer becomes #
-smp#	size of Member Pointer becomes #
-smpD#	size of Member Data Pointers
-smpP#	size of Member Program Pointer
-smpFP#	size of **far** Member Program Pointers
-smpNP	size of **near** Member Program Pointers
-sw#	size of **wchar_t** becomes #

Verbosity Options

(- output to **stdout**, + to **stderr** and **stdout**) The default is **-vm**

-v	Turn off all work in progress messages
-/+vm	Module names only
-/+vf	File and module names only
-/+v*n*	Every *n* lines with file and module names
-/+vi...	Indirect filenames
-/+vo...	output Options
-/+vs...	append a Storage report
-/+vh...	dump the type Hierarchy
-/+v#...	tag filenames with an id number

Flag Options

(**+** sets,**-** resets, **++** increments and **--** decrements flag)

fab	ABbreviated structure flag
fan	ANonymous union flag
fas	Anonymous struct flag
fba	Bit Addressability flag
fbc	Boolean Constant flag
fbo	Boolean flag
fbu	Bit Fields are unsigned flag
fcd	CDecl is significant flag
fce	Continue-on-Error flag
fcp	C++ flag
fct	Create Tag flag
fcu	Char-is-Unsigned flag
fdc	Distinguish-plain-Char flag
fdh	dot-h flag
fdi	Directory of Including file flag
fdl	pointer-Difference-is-Long flag
fdr	Deduce-Return-mode flag
feb	allow Enumerations as Bit fields flag
fem	Early Modifiers flag
ffb	**for** loop creates separate block flag
ffd	promote Floats to Doubles flag
fff	Fold Filenames flag
ffn	Full (file) Name flag
ffo	Flush Output files flag
fhg	Hierarchy Graphics flag
fhd	The strong-Hierarchy-Down flag
fhs	Hierarchy-of-Strong-types flag
fhx	Hierarchy-of-strong-indeXes flag
fie	Integer-model-for-Enum flag
fil	Indentation-check-on-Labels flag
fim	Include-Multiple flag
fiq	The Ignore-default-Qualifier flag
fis	Integral-constants-are-Signed flag
fkp	K&R Preprocessor flag
flb	LiBrary flag
flc	The long-char flag
flf	process Library-Function flag
fll	The long-long flag
fln	#LiNe directives flag
fmd	Multiple Definitions flag
fna	Allow **operator new[]** flag
fnc	Nested Comments flag
fnt	Nested Tag flag
fod	Output-Declared-objects flag
fol	Output-Library-objects flag

fpa	PAuse flag
fpc	Pointer Casts retain lvalue flag
fpm	Precision is limited to the Max. args. flag
fpn	The Pointer-parameter-may-be-NULL flag
fps	Parameters within Strings flag
frb	Read Binary flag
fsa	Structure-Assignment flag
fsc	String constants are **const** char
fsh	SHared reading flag
fsu	String Unsigned flag
ftf	raw-Template-Function flag
ful	Unsigned **long** flag
fva	Variable Arguments flag
fvo	VOid data type flag
fvr	Varying-Return-mode flag
fwc	**wchar_t** is built-in flag
fwu	**wchar_t** is Unsigned flag
fxa	eXact-Array flag
fxc	eXact-Char flag
fxf	eXact-Float flag
fxs	eXact-Short flag
fzl	siZeof-is-Long flag
fzu	siZeof-is-Unsigned flag

Message Presentation Options

```
-h[s][F][f][a][b][r][mn][m][m/M/][I]N
         controls the height of messages - Default is -ha_3
             s  blank line between messages
             F  always include a filename
             f  out-of-sequence file information
             r  repeat source line
             mn no macro display
             m  undo effect of mn
             m/M/ assigns new prefix M for macro display
             I  error location mark
                 a  I appears above the source line
                 b  I appears below the source line
             N  number of lines in error message

-width(Width,Indent)  controls width of messages - Default is -width(79,4)
             Width  - Width of output device
             Indent - Indent continued lines

-format=    message format for msg. height < =3
                      Default is "%(%f  %l  %)%t %n: %m"

-format4a=  format of first line for msg. height == 4
                      Default is "%(File %f, Line %l\n%)"

-format4b=  format of fourth line for msg. height == 4
                      Default is    "%(     %)%t %n: %m"
```

Other Options

-A	requests only ANSI keywords
-/+**b**	suppresses or diverts the Banner line
-c*code*	specifies a particular Compiler
-/+**cpp(***extension***)**	add (+) or remove (-) a C++ extension
-d*name*[**=***value*]	Define preprocessor variables
+d*name*[**=***value*]	like -d except that the definition is locked in
-D*name*[**=***value*][;...]	like -d except that a list of name-value pairs is supported
+ext(*extension*[*,extension*]...**)**	specify list for default extensions
-father(*Parent,Child*[*,Child*]...**)**	adds strict relationship to strong type heirarchy
-function(*f0, f1...***)**	assign special attributes
-header(*filename***)**	automatically read header
-i*directory*	**#include** file directory
-incvar(*name***)**	specifies an environment variable to be used in place of INCLUDE
-ident(*String***)**	augment identifier characters
-idlen(*n*,*opt***)**	check for identifier clashes
-index(*flags,ixtype,sitype*,...**)**	specifies strong index
-library	current or next module is a library
+libclass(*...***)**	default library headers (default: **foreign,angle**)
-/+**libdir(***directory*,...**)**	library header directory
-/+**libh(***file*,...**)**	library header
-limit(*n***)**	specify limit on number of messages
-/+**lnt(***extension*[*,extension*]...**)**	add or remove an Indirect File extension
-lobbase(*filename***)**	establish a lob base file
+macros	increases the size of macro storage.
-mS	**near** data and program (Default)
-mD	**far** Data, **near** program
-mP	**far** Program, **near** data

-mL	**far** data and program
-od*opts*[*width*]**(***filename***)**	Output Declarations (prototypes)
+od*opts*[*width*]**(***filename***)**	Append Declarations
	opts are: **f** only functions
	i internal (static) functions also
	s structs also
	width set output width
-oe(*filename* **)**	redirects output for **stderr** to the named file
+oe(*filename* **)**	Append output for **stderr** to the named file
-ol(*filename* **)**	Output Library
-oo[**(** *filename* **)**]	Output Object module (default: *Name***.lob**)
-os(*filename* **)**	Output Standard out to file
+os(*filename* **)**	Append Standard out to file
-p[**(***width***)**]	run just the Preprocessor
-parent(*Parent, Child...***)**	adds to strong type hierarchy
-ppw(*word,...***)**	removes PreProcessor Word(s)
+ppw(*word,...***)**	adds PreProcessor Word(s)
+pragma(*name***,***action***)**	specify *action* for **#pragma** *name*
-printf(*n,name,...***)**	**printf**-like functions
-restore	resets error inhibition state
-rw(*word,...***)**	removes Reserved Word(s)
+rw(*word,...***)**	adds Reserved Word(s)
-save	saves error inhibition state
-scanf(*n,name,...***)**	**scanf**-like functions
-sem(*name*[,sem]...**)**	check functions for user-defined semantics
-size(*flags,amount***)**	issue a message when data variable's size >= *amount*
-strong(*flags,name,...***)**	specifies strong types
-t*#*	set Tab size to # (default = 8)
-u	unit checkout (suppresses inter-module messages)
-unreachable	a point in a program is unreachable
-u*name*	Undefines *name*
--u*name*	inhibits macro *name* from becoming defined
-w*Level*	set Warning level
-wlib(*Level***)**	set Warning level for messages within library.
-wprintf(*n,name,...***)**	wide-character version of **-printf** option
-wscanf(*n,name,...***)**	wide-character version of **-scanf** option
-zero	sets exit code to 0
-zero(*#***)**	same unless a message number < *#*
-$	**$** is an identifier character

Compiler Dependency Options

-a#*predicate*	Assert the # predicator (Unix)
-#d*name***=***Replacement*	**#include** Define
-d*name***()=***Replacement*	Define function-like macro
-overload(*X***)**	sets flags which can affect overload resolution
-plus(*char***)**	define an alternate option character for '+'
-template(*X***)**	fine-tune template processing

Compiler Dependency Keywords

_bit	one bit wide
_gobble	causes the next token to be gobbled
_to_brackets	ignore next expression
_to_semi	ignore until **;**

Tel: +44(0) 1364 654100
Fax: + 44(0) 1364 654200
Email: MailDesk @ GreyMatter.co .uk
WebSite: http : / / www.GreyMatter .co .uk
Post: Grey Matter Ltd, Prigg Meadow, Ashburton, Devon, TQ13 7DF