

- ★★★★★ excellent
- ★★★★☆ very good
- ★★★☆☆ average
- ★★☆☆☆ disappointing
- ★☆☆☆☆ don't bother

GIMPEL PC-LINT VERSION 8.0

by Herb Marselas

Gimpel PC-lint 8.0 (Lint) is arguably one of the most powerful, and yet daunting, tools for C/C++ programmers. The power of Lint lies in the exhaustive number of warnings, errors, and informational messages that its static code analysis generates, informing you of virtually every potential peril and pitfall waiting in your code. The messages generated by Lint range from strict C/C++ error information to coding recommendations from Dan Saks and Scott Meyers (the author of *Effective C++*). The latest version of Lint doesn't disappoint, adding nearly ninety new messages and forty enhanced messages and options to the range of information supplied.

The problem with all this power is that it can lead some programmers to dismiss Lint's bountiful and seemingly extraneous messages. This complaint is not unjustified. Although our own code base at Ensemble compiles without a problem on our compiler's highest warning level, a fresh install of Lint generated a fifty-six thousand-line message file when run on just one of our files. Within a few hours, I was able to identify and disable all of the extraneous messages and get down to a meaningful set of potential problems that needed to be examined and addressed.

Beyond its power of static code analysis, Lint's appeal lies in the broad range of platforms and compilers that it supports. For Windows users, Lint is released as an executable. But for other platforms, Lint is released as a shrouded, or obfuscated, set of source files. Lint also doesn't favor one compiler or library over another, supporting configurations for more than thirty compilers and many common programming libraries out of the box.

One of Lint's few compiler-specific features is its new support for Microsoft Visual C++ Developer Studio Project files (.DSP). Visual C++ users can now feed their .DSP file directly into Lint, eliminating the need to copy the preprocessor settings from the Visual C++ IDE into Lint's configuration files. However, Visual C++ users must still manually add other settings, such as the include directories, from the Tools Directories menu to the Lint con-

figuration files. As long as Lint already supports the Visual C++ project files, Gimpel should have gone the extra step and included the ability to read additional configuration information directly from the Microsoft Windows registry.

Perhaps the most important new feature in Lint 8.0 is the interactive value tracking. When it's enabled, interactive value tracking attempts to verify and track values passed between functions by making up to six passes over the code being analyzed. This function can help find value-out-of-range conditions or other situations in which parameter passing could cause problems. While I was able to see this functionality at work with the sample programs that Gimpel supplies with the Lint package, in practice I was unable to find any issues of this type using a sampling of files from Ensemble's large source code base.

Another change with this version is that the documentation is now completely online in Adobe Acrobat format. The documentation is relatively complete, though it could include more examples illustrating the messages that Lint generates. NuMega's BoundsChecker, for instance, includes a small code example for each of its errors; a similar glossary could only enhance Lint's usability.

The only major problem that I encountered while testing Lint 8.0 was a crash bug that surfaced while parsing some of our template code. However, Gimpel's technical support staff was able to get us a workaround after just a few days and a half-dozen e-mails. Unfortunately, we had to wait almost a month for a patch to fix the crash, so in the meantime we had to modify our code whenever we wanted to run Lint.

I also found that Lint doesn't correctly add itself to the path during installation if you're running Microsoft Windows NT, 2000, or XP. This seems like a glaring oversight, as Gimpel has added other Windows NT/2000/XP features, such as the ability to reduce the priority of the process if you want to run Lint as a background task.

If you don't have Lint, get it today. If you already have the last major version, carefully evaluate whether these new features are enough to justify the upgrade. Regardless of your choice in compiler and

platform, using Lint can only improve the quality of your code if you take the time to configure it properly and use it regularly.

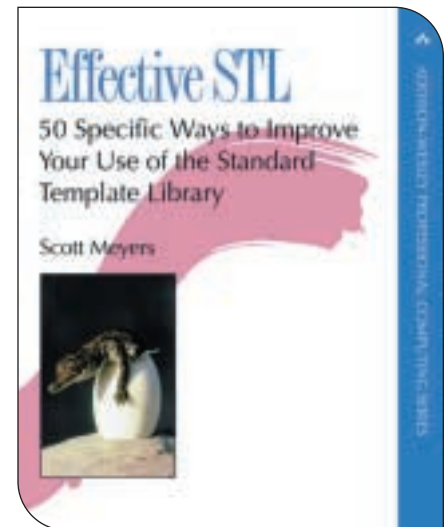
★★★★★ | Gimpel Software | PC-lint 8.0
www.gimpel.com

Herb Marselas is a programmer at Ensemble Studios.

EFFECTIVE STL: 50 SPECIFIC WAYS TO IMPROVE YOUR USE OF THE STANDARD TEMPLATE LIBRARY BY SCOTT MEYERS

review by Noel Llopis

Have you read *Effective C++*, also by Scott Meyers? No? Go buy it right now, read it, reread it, and then come back here. I guarantee that it will make a huge difference in the way you work.



OK, welcome back. Here's the good news: *Effective STL* is to STL what *Effective C++* is to C++. Meyers's latest book is equally great and is written in the same light, conversational tone that made reading his earlier work such a pleasure. It assumes that the reader has a basic knowledge of STL (and C++) and builds on that knowledge to show how to use it effectively and avoid common pitfalls.

The Standard Template Library (STL) is made up of generic data structures and algorithms that can be used across many compilers and platforms, including most current consoles. The library is implemented with templates, so the resulting code is

quite efficient, possibly even more so than hand-coded structures and algorithms. Additionally, STL already has been implemented, tested, debugged, and optimized by thousands of people, so it's generally reliable code. STL also empowers its users by putting powerful constructs at their fingertips. Previously, a common approach was to throw a bunch of objects into an array and search through them in linear time (does that sound familiar?). STL lets you put these objects into a hash table and thus have constant-time access to them.

Effective STL begins with a fairly thorough discussion of containers (vector, list, map, and so on) and iterators. Meyers immediately goes beyond a typical description of the containers and their $O(n)$ performance characteristics by addressing important, real-world questions: Is the memory for the elements allocated contiguously? Are the iterators invalidated when the elements change? What is the most efficient way of removing elements for a specific container?

Meyers then moves on to algorithms and functors. Just as with his discussion of containers, rather than simply listing all the available algorithms, he points out common mistakes and how to deal with them. Meyers reveals, for example, different ways of sorting elements, or how `std::remove` really works (and why it doesn't really remove anything).

The final chapters present more general, but still very useful, information on the STL. Meyers discusses when to use STL algorithms and when to use your own, style guidelines, or even how to deal with the broken STL implementation and template support in Microsoft's Visual C++.

But STL isn't perfect. Readers will gain some ideas of where STL is lacking, but these issues aren't spelled out explicitly. Sometimes you'll need to read between the lines and consider how STL applies to game development. For example, memory allocation can be an issue, especially for those doing console development; game developers will probably want to write their own allocators. *Effective STL* has a brief discussion of allocators, but doesn't provide enough information to guide developers who need to write their own.

Another of STL's oft-cited shortcomings is the difficulty debugging STL code.

Developers working with STL often face cryptic multi-line error messages and experience difficulty viewing the elements of a container in the debugger. *Effective STL* shows readers how to parse the intimidating error messages and includes pointers to STL resources on the Internet.

Effective STL works well both as a book to read cover to cover and as a reference later on. Meyers only uses source code where it's necessary; he doesn't bore his readers with pages and pages of pointless code. As a matter of fact, this book won't bore you at all since it packs a lot of information into a mere 250 pages. Overall, you simply must read *Effective STL* before you decide to use (or not to use) STL in your next game or tools. If you're already using STL, then this book should already be on your bookshelf.

 | Addison-Wesley
Effective STL | www.aristeia.com

Noel Llopis a software engineer at Meyer/Glass Interactive, where he specializes in 3D graphics.