

COMPILER

GNU Compiler Collection 3.3

GCC has always been one of the most crucial components of the Linux world, and in recent times it has gone through a great deal of change. Paul Hudson looks at what the latest release has to offer...



BUYER INFO

The GNU Compiler Collection is the essential development tool

- **DEVELOPER** GNU
- **PRICE** Free
- **WEB** www.gnu.org

What does the C statement `i = i++`; do? It's actually not as obvious as you might first think – in fact, the return is undefined, which means that compiler writers are free to do as they please in this situation. If `i` started off as 4, it might end up as 4, or perhaps 5. There's even the chance of it becoming an embarrassed-looking elephant – as I said, the result is undefined, which means relying on it returning 4 isn't a smart move as it can be different in other compilers, or even newer versions of the same compiler. Up until the release of the GCC 3.x series, GCC had quite a few "features" in it that were relied up on to perform a certain task and to return a consistent value – with the release of GCC 3.0, 3.1, 3.2, and now 3.3, the development team are slowly ridding themselves of these inconsistencies, adding heaps of new features, and providing much more standards-compliant C++ compilation.

If you have used GCC 3.2, you'll know it's a lot slower than 2.9x – particularly when C++ templates were involved. So, what does GCC 3.3 bring to the table?

Fixes and features

This release has just under 200 bugs fixed over GCC 3.2, which is an amazing improvement by any standards. However, it's not all bug fixes – there are several new front-end features that you can make use of right now. Perhaps most importantly, a lot of work has been done to enable easier code profiling. Pavel Nejedly contributed a lot of

work to enable much more accurate profiling of code, and this data is passed to the optimizer to help it identify hot spots in code on a global basis, which results in better code. Another excellent (and very complicated!) improvement is a new scheduler using deterministic finite automata, which should hopefully allow instruction re-ordering during compilation to provide an extra speed boost for run-time execution.

Several parts of GCC have been deprecated or removed, although none are surprising in light of GCC's continuing quest towards standards. For example, the preprocessor no longer accepts multi-line string literals – a feature deprecated in GCC 3.0, 3.1, and 3.2, and now totally removed in 3.3. This makes the following C++ statement illegal:

```
MySTLMap["foo"] = "Hello, world!";
```

Removal of support for things like the above have been a long time coming, and it's good to see the GCC team aren't afraid to make such changes even now. While this might bite the few programmers who make use of such odd hacks, it's better in the long run.

Performance tradeoff

I'll be blunt: GCC 3.3 is no faster to compile than GCC 3.2, which means it's *slow*. However, like 3.2, you'll generally find that what you trade off in compilation time you'll get back with run-time performance, which is generally much more important. With GCC 3.3, you'll generally find some programs will really fly once recompiled, and this is thanks to the fact that GCC now supports both the SSE2 and 3DNow! instruction set extensions – perfect for programs that do a lot of multimedia work. There have also been some big steps forward on the x86-64 port, so Opteron early adopters will be particularly eager to get their hands on this release.

The general slow-down in speed isn't a bad thing, because it's easily justified by all the extra lengths GCC goes to make sure your code is correct. If you're only now switching from 2.9x you'll find GCC 3.3 issues many more warnings and errors than 2.9x, which is very good in the long run as it makes programmers write better-structured code.

More breakage

Pretty much as soon as GCC 3.3 was out in the wild, programmers were complaining they couldn't compile the kernel in GCC 3.3, and the basic reason for this is that the Linux kernel has for a very long time relied on various bugs, holes, and misunderstandings in the GCC compiler, which means that as GCC is brought more and more into line with the C and C++ standards, more and more programs break. In this situation the problem is that GCC had a special `__inline__` extension that Linux developers took advantage of, which has now been removed.

While some say that GCC should just leave those kind of features alone, it's hard to draw the line – if GCC is careful not to break the kernel, should it also be careful not to break XFree86? Or *OpenOffice.org*? The answer is simply that GCC is much, much bigger than the Linux kernel. Yes, Linux is a wonderful OS, but GCC works on dozens of platforms, from Windows to Macintosh, and from Solaris to Amiga – it needs to stay true to the standards, and programmers who rely on undocumented features should come into line with everyone else.

GCC will hopefully soon get enough standards compliance that people need to worry about breaking programs when upgrading, but it's a two-part affair – programmers need to upgrade as soon as possible and check to make sure their code is compliant with the new release, and

to make sure they take deprecation notices seriously.

Three point four

GCC 3.4 is already on the horizon, albeit still quite a way away, and is looking to follow up 3.3's feature bash with a solid focus on speed. Yes, you read that correctly – the 3.x series is finally going to get some tuning. 3.3 is a great release – particularly to help developers get their code in line – but I wouldn't say it's necessary for everyone. While it does retain backwards compatibility with prior versions, and does add quite a bit of interesting new functionality, it's something primarily of interest for developers.

If you *are* a developer, then this is a definite upgrade. If you're thinking anything like "I'll stick 2.9x for now and wait till they stop making changes", then you'll find that when you finally choose to upgrade (to version 3.x or even 4.x), you'll find you have thousands of errors and warnings. Chances are you'll fare better by upgrading now and fixing errors along the way.

So, to conclude, this is another solid step forward for GCC, keeping it firmly at the forefront of compiler technology. At this rate I don't think it will be that long before GCC comes close to rivalling the runtime performance of Intel's compiler – only time will tell. **LXF**

VERDICT

Features	9/10
Performance	7/10
Ease of use	10/10
Documentation	8/10

The world's most popular compiler gets another worthy upgrade. Not essential, though – consider waiting for 3.4.

LINUX FORMAT RATING
 9/10