G++ FAQ

Frequently asked questions about the GNU C++ compiler
September 15, 1995

Joe Buck

This is a list of frequently asked questions (FAQ) for g++ users; thanks to all those who sent suggestions for improvements. Thanks to Marcus Speh for doing the index.

Please send updates and corrections to the FAQ to `jbuck@synopsys.com`. Please do *not* use me as a resource to get your questions answered; that's what gnu.g++.help is for and I don't have the time to support the net's use of g++.

Many FAQs, including this one, are available on the archive site rtfm.mit.edu, in the directory '`pub/usenet/news.answers`'. This FAQ may be found in the subdirectory g++-FAQ.

This FAQ is intended to supplement, not replace, Marshall Cline's excellent FAQ for the C++ language and for the newsgroup comp.lang.c++. Especially if g++ is the first C++ compiler you've ever used, the question "How do I do <X> with g++?" is probably really "How do I do <X> in C++?". You can find this FAQ on rtfm.mit.edu under '`pub/usenet/comp.lang.c++`'.

1 The latest poop – gcc-2.7.0

This section is intended to describe more recent changes to g++, libg++, and such. Some things in this section will eventually move elsewhere.

The big news is that version 2.7.0 has just been released. This, of course, means new FAQs. I haven't had a lot of time to whip out this section yet, so suggestions for improvement are welcome.

1.1 gcc-2.7.0 breaks declarations in "for" statements!

gcc-2.7.0 implements the new ANSI/ISO rule on the scope of variables declared in for loops.

```
for (int i = 1; i <= 10; i++) {
        // do something here
}
foo(i);
```

In the above example, most existing C++ compilers would pass the value 11 to the function `foo`. In gcc 2.7 and in the ANSI/ISO working paper, the scope of `i` is only the for loop body, so this is an error. So that old code can be compiled, the new gcc has a flag `-fno-for-scope` that causes the old rule to be used.

1.2 What's new in version 2.7.0 of gcc/g++

The long-awaited version 2.7.0 of gcc/g++ (along with a matching version of libg++) have now been released. This represents a great deal of work on the part of the g++ maintainers to fix outstanding bugs and move the compiler closer to the current ANSI/ISO standards committee's working paper, including supporting many of the new features that have been added to the language. I recommend that everyone read the NEWS file contained in the distribution (and that system administrators make the file available to their users). I've borrowed liberally from this file here.

If any features seem unfamiliar, you will probably want to look at the recently-released public review copy of the C++ Working Paper. For PostScript and PDF (Adobe Acrobat) versions, see the archive at ftp://research.att.com/dist/stdc++/WP. For HTML and ASCII versions, see ftp://ftp.cygnus.com/pub/g++. On the World Wide Web, see http://www.cygnus.com/~mrs/wp-draft.

- As described above, the scope of variables declared in the initialization part of a for statement has been changed; such variables are now visible only in the loop body. Use `-fno-for-scope` to get the old behavior. You'll need this flag to build groff version 1.09, Ptolemy, and many others.
- Code that does not use #pragma interface/implementation will most likely shrink dramatically, as g++ now only emits the vtable for a class in the translation unit where its first non-inline, non-abstract virtual function is defined.
- Support for automatic template instantiation has *not* been enabled in the official distribution, due to a disagreement over design philosophies. But you can get a patch from Cygnus to turn it on; retrieve the patch from '`ftp://ftp.cygnus.com/pub/g++/gcc-2.7.0-repo.gz`'.
- Support for exception handling has been improved; more targets are now supported, and throws will use the RTTI mechanism to match against the catch parameter type. You must give the `-fhandle-exceptions` to turn it on. Optimization is *not supported* with `-fhandle-exceptions`; no need to report this as a bug. You'll probably get an internal compiler error if you try it.

  For exception handling to work your CPU must be a SPARC, RS6000/PowerPC, 386/486/Pentium, or ARM. Other platforms are missing the function to unwind the stack.
- Support for Run-Time Type Identification has been added with `-frtti`. This support is still in alpha; one major restriction is that any file compiled with `-frtti` must include `<typeinfo>` (*not* `typeinfo.h` as the NEWS file says). Also, all code you link with (including libg++) has to be built with `-frtti`, so it's still tricky to use.
- Synthesis of compiler-generated constructors, destructors and assignment operators is now deferred until the functions are used.
- The parsing of expressions such as `a ? b : c = 1` has changed from `(a ? b : c) = 1` to `a : b ? (c = 1)`. This is a new C/C++ incompatibility brought to you by the ANSI/ISO standards committee.
- The operator keywords and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor and xor_eq are now supported. Use `-ansi` or `-foperator-names` to enable them.
- The `explicit` keyword is now supported. `explicit` is used to mark constructors and type conversion operators that should not be used implicitly.
- Handling of user-defined type conversion has been improved.
- Explicit instantiation of template methods is now supported. Also, `inline template class foo<int>;` can be used to emit only the vtable for a template class.
- With -fcheck-new, g++ will check the return value of all calls to operator new, and not attempt to modify a returned null pointer.
- collect2 now demangles linker output, and c++filt has become part of the gcc distribution.

- Improvements to template instantiation: only members actually used are instantiated.

1.3 How do I use the new repository code?

Because there is some disagreement about the details of the template repository mechanism, you'll need to obtain a patch from Cygnus Support to enable the 2.7.0 repository code. You can obtain the patch by anonymous FTP: '`ftp://ftp.cygnus.com/pub/g++/gcc-2.7.0-repo.gz`'.

After you've applied the patch, the `-frepo` flag will enable the repository mechanism. The flag works much like the existing `-fno-implicit-templates` flag, except that auxiliary files, with an '`.rpo`' extension, are built that specify what template expansions are needed. At link time, the (patched) collect program detects missing templates and recompiles some of the object files so that the required templates are expanded.

Note that the mechanism differs from that of cfront in that template definitions still must be visible at the point where they are to be expanded. No assumption is made that '`foo.C`' contains template definitions corresponding to template declarations in '`foo.h`'.

Jason Merrill writes: "To perform closure on a set of objects, just try to link them together. It will fail, but as a side effect all needed instances will be generated in the objects."

1.4 The GNU Standard C++ Library

The GNU Standard C++ Library (also called the "GNU ANSI C++ Library" in places in the code) is not libg++, though it is included in the libg++ distribution. Rather, it contains classes and functions required by the ANSI/ISO standard. The copyright conditions are the same as those for for the iostreams classes; the LGPL is not used. See See Chapter 5 [legalities], page 21.

This library, libstdc++, is in the libg++ distribution in versions 2.6.2 and later. It requires at least gcc 2.6.3 to build the libg++-2.6.2 version; use at least gcc 2.7.0 to build the libg++ 2.7.0 version. It contains a hacked-up version of HP's implementation of the Standard Template Library (see See Section 4.16 [Standard Template Library], page 19). I've successfully used this Standard Template Library version to build a number of the demos you'll see on various web pages.

As of version 2.7.0, the streams classes are now in libstdc++ instead of libg++, and libiostream is being phased out (don't use it). The g++ program searches this library.

2 Obtaining Source Code

2.1 What is the latest version of gcc, g++, and libg++?

The latest "2.x" version of gcc/g++ is 2.7.0, released June 16, 1995. The latest version of libg++ is 2.7.0a, released June 19, 1995 (2.7.0 had an error in a makefile and was almost immediately replaced). . Don't use 2.5.x, with x less than 5, for C++ code; there were some serious bugs that didn't have easy workarounds. 2.5.8 is the most solid 2.5.x release. 2.6.3 is the most solid 2.6.x release.

For some non-Unix platforms, the latest port of gcc may be an earlier version (2.5.8, say). You'll need to use a version of libg++ that has the same first two digits as the compiler version, e.g. use libg++ 2.5.x (for the latest x you can find) with gcc version 2.5.8.

The latest "1.x" version of gcc is 1.42, and the latest "1.x" version of g++ is 1.42.0. While gcc 1.42 is quite usable for C programs, I recommend against using g++ 1.x except in special circumstances (and I can't think of any such circumstances).

2.2 How do I get a copy of g++ for Unix?

First, you may already have it if you have gcc for your platform; g++ and gcc are combined now (as of gcc version 2.0).

You can get g++ from a friend who has a copy, by anonymous FTP or UUCP, or by ordering a tape or CD-ROM from the Free Software Foundation.

The Free Software Foundation is a nonprofit organization that distributes software and manuals to raise funds for more GNU development. Getting your copy from the FSF contributes directly to paying staff to develop GNU software. CD-ROMs cost $400 if an organization is buying, or $100 if an individual is buying. Tapes cost around $200 depending on media type. I recommend asking for version 2, not version 1, of g++.

For more information about ordering from the FSF, contact gnu@prep.ai.mit.edu, phone (617) 542-5942 or anonymous ftp file '`ftp://prep.ai.mit.edu/pub/gnu/GNUinfo/ORDERS`' (you can also use one of the sites listed below if you can't get into "prep").

Here is a list of anonymous FTP archive sites for GNU software. If no directory is given, look in '`/pub/gnu`'.

```
ASIA: ftp.cs.titech.ac.jp, utsun.s.u-tokyo.ac.jp:/ftpsync/prep,
cair.kaist.ac.kr, ftp.nectec.or.th:/pub/mirrors/gnu

AUSTRALIA: archie.oz.au:/gnu (archie.oz or archie.oz.au for ACSnet)

AFRICA: ftp.sun.ac.za

MIDDLE-EAST: ftp.technion.ac.il:/pub/unsupported/gnu

EUROPE: irisa.irisa.fr, ftp.univ-lyon1.fr:, ftp.mcc.ac.uk,
unix.hensa.ac.uk:/pub/uunet/systems/gnu, ftp.denet.dk,
src.doc.ic.ac.uk:/gnu, ftp.eunet.ch, nic.switch.ch:/mirror/gnu,
ftp.informatik.rwth-aachen.de, ftp.informatik.tu-muenchen.de,
ftp.win.tue.nl, ftp.funet.fi, ftp.stacken.kth.se, isy.liu.se,
ftp.luth.se:/pub/unix/gnu, ftp.sunet.se, archive.eu.net

SOUTH AMERICA: ftp.unicamp.br, ftp.inf.utfsm.cl

WESTERN CANADA: ftp.cs.ubc.ca:/mirror2/gnu

USA: wuarchive.wustl.edu:/systems/gnu, labrea.stanford.edu,
ftp.digex.net, ftp.kpc.com:/pub/mirror/gnu,
f.ms.uky.edu:/pub3/gnu, jaguar.utah.edu:/gnustuff,
```

```
ftp.hawaii.edu:/mirrors/gnu, vixen.cso.uiuc.edu:/gnu,
mrcnext.cso.uiuc.edu, ftp.cs.columbia.edu:/archives/gnu/prep,
col.hp.com:/mirrors/gnu, gatekeeper.dec.com:/pub/GNU,
ftp.uu.net:/systems/gnu
```

The “official site” is prep.ai.mit.edu, but your transfer will probably go faster if you use one of the above machines.

Most GNU utilities are compressed with “gzip”, the GNU compression utility. All GNU archive sites should have a copy of this program, which you will need to uncompress the distributions.

UUNET customers can get GNU sources from UUNET via UUCP. UUCP-only sites can get GNU sources by “anonymous UUCP” from site "osu-cis" at Ohio State University. You pay for the long-distance call to OSU; the price isn’t too bad on weekends at 9600 bps. Send mail to uucp@cis.ohio-state.edu or osu-cis!uucp for more information.

OSU lines are often busy. If you’re in the USA, and are willing to spend more money, you can get sources via UUCP from UUNET using their 900 number: 1-900-GOT-SRCS (900 numbers don’t work internationally). You will be billed $0.50/minute by your phone company.

Don’t forget to retrieve libg++ as well!

2.3 Getting gcc/g++ for the HP Precision Architecture

If you use the HP Precision Architecture (HP-9000/7xx and HP-9000/8xx) and you want to use debugging, you’ll need to use the GNU assembler, GAS (version 2.3 or later). If you build from source, you must tell the configure program that you are using GAS or you won’t get debugging support. A non-standard debug format is used, since until recently HP considered their debug format a trade secret. Thanks to the work of lots of good folks both inside and outside HP, the company has seen the error of its ways and has now released the required information. The team at the University of Utah that did the gcc port now has code that understands the native HP format.

Some enhancements for the HP that haven’t been integrated back into the official GCC are available from the University of Utah, site jaguar.cs.utah.edu. You can retrieve sources and prebuilt binaries for GCC, GDB, binutils,and libg++; see the directory ‘`/dist`’.

The libg++ version is actually the same as the FSF 2.6. The Utah version of GDB can now understand both the GCC and HP C compiler debug formats, so it is no longer necessary to have two different GDB versions.

I recommend that HP users use the Utah versions of the tools (see above), though at this point the standard FSF versions will work well.

HP GNU users can also find useful stuff on the site geod.emr.ca in the ‘`/pub/UNIX/GNU-HP`’ directory.

Jeff Law is leaving the University of Utah, so the Utah prebuilt binaries may be discontinued.

2.4 Getting gcc/g++ binaries for Solaris 2.x

"Sun took the C compiler out of Solaris 2.x. Am I stuck?"

No; prep.ai.mit.edu and its mirror sites provide GCC binaries for Solaris. As a rule, these binaries are not updated as often as the sources are, so if you want the very latest version of gcc/g++, you may need to grab and install binaries for an older version and use it to bootstrap the latest version from source.

The latest gcc binaries on prep.ai.mit.edu and its mirror sites are for version 2.5.6 for Solaris on the Sparc, and version 2.4.5 for Solaris on Intel 386/486 machines. There are also binaries for "gzip", the GNU compression utility, which you'll need for uncompressing the binary distribution. On any GNU archive site, look in subdirectories '`i486-sun-solaris2`' or '`sparc-sun-solaris2`'.

The ftp directory /pub/GNU on site ftp.quintus.com contains various GNU and freeware programs for Solaris2.X running on the sparc. These are packaged to enable installation using the Solaris "pkgadd" utility. These include GNU emacs 19.27, gcc (and g++) 2.6.0, Perl 4.036, and others.

2.5 How do I get a copy of g++ for (some other platform)?

The standard gcc/g++ distribution includes VMS support. Since the FSF people don't use VMS, it's likely to be somewhat less solid than the Unix version. Precompiled copies of g++ and libg++ in VMS-installable form are available by FTP from mango.rsmas.miami.edu. See also the site ftp.stacken.kth.se (in Sweden), directory /pub/GNU-VMS/contrib, which has gcc-2.5.8 and libg++-2.5.3.

There are two different versions of gcc/g++ for MS-DOS: EMX and DJGPP. EMX also works for OS/2 and is described later. DJGPP is DJ Delorie's port. It can be found on many FTP archive sites; its "home" is on oak.oakland.edu, directory '`~ftp/pub/msdos/djgpp`'.

The latest version of DJGPP is 1.12.maint1. This version runs under Windows 3.x. It includes a port of gcc 2.6.0, plus support software.

FSF sells floppies with DJGPP on them; see above for ordering software from the FSF.

A new Usenet group, `comp.os.msdos.djgpp`, has recently been created.

For information on Amiga ports of gcc/g++, retrieve the file '`/pub/gnu/MicrosPorts/Amiga`' from prep.ai.mit.edu, or write to Markus M. Wild <wild@nessie.cs.id.ethz.ch>, who I hope won't be too upset that I mentioned his name here.

A port of gcc to the Atari ST can be found on the site "atari.archive.umich.edu", under '`/atari/Gnustuff/Tos`', along with many other GNU programs. This version is usually the same as the latest FSF release. See the "Software FAQ" for the Usenet group "comp.sys.atari.st" for more information.

There are two different ports of gcc to OS/2, the so-called EMX port (which also runs on MS-DOS), and a port called "gcc/2". The latter port is no longer supported, since the EMX port includes all of its functionality. The EMX port's C library attempts to provide a Unix-like environment. For more information ask around on "comp.os.os2.programmer.misc".

The EMX port is available by FTP from

```
ftp.uni-stuttgart.de(129.69.1.12) in /pub/systems/os2/emx-0.9a
src.doc.ic.ac.uk(146.169.2.1) in /pub/packages/os2/unix/emx09a
ftp.informatik.tu-muenchen.de(131.159.0.198) in
        /pub/comp/os/os2/devtools/emx+gcc
```

Eberhard Mattes did the EMX port. His address is mattes@azu.informatik.uni-stuttgart.de.

I'm looking for more information on gcc/g++ support on the Apple Macintosh. Until recently, this FAQ did not provide such information, but FSF is no longer boycotting Apple as the League for Programming Freedom boycott has been dropped.

Mike White (cons116@twain.oit.umass.edu) says: "Versions 1.37.1 and 2.3.3 of gcc were ported by Stan Shebs and are available at ftp.cygnus.com under /pub/shebs. They are both interfaced to MPW. Shebs is apparently working on a cross compiler of 2.6.3 to create Mac apps from Unix boxes."

I don't know anything about more recent versions.

2.6 But I can only find g++-1.42!

"I keep hearing people talking about g++ 2.5.8 (or some other number starting with 2), but the latest version I can find is g++ 1.42. Where is it?"

As of gcc 2.0, C, C++, and Objective-C as well are all combined into a single distribution called gcc. If you get gcc you already have g++. The standard installation procedure for any gcc version 2 compiler will install the C++ compiler as well.

One could argue that we shouldn't even refer to "g++-2.x.y" but it's a convention. It means "the C++ compiler included with gcc-2.x.y."

3 Installation Issues and Problems

3.1 I can't build g++ 1.x.y with gcc-2.x.y!

"I obtained gcc-2.x.y and g++ 1.x.y and I'm trying to build it, but I'm having major problems. What's going on?"

If you wish to build g++-1.42, you must obtain gcc-1.42 first. The installation instructions for g++ version 1 leave a lot to be desired, unfortunately, and I would recommend that, unless you have a special reason for needing the 1.x compiler, that C++ users use the latest g++-2.x version, as it is the version that is being actively maintained.

There is no template support in g++-1.x, and it is generally much further away from the ANSI draft standard than g++-2.x is.

3.2 OK, I've obtained gcc; what else do I need?

First off, you'll want libg++ as you can do almost nothing without it (unless you replace it with some other class library).

Second, depending on your platform, you may need "GAS", the GNU assembler, or the GNU linker (see next question).

Finally, while it is not required, you'll almost certainly want the GNU debugger, gdb. The latest version is 4.14, released March 2, 1995. Other debuggers (like dbx, for example) will normally not be able to understand at least some of the debug information produced by g++.

3.3 Should I use the GNU linker, or should I use "collect"?

First off, for novices: special measures must be taken with C++ to arrange for the calling of constructors for global or static objects before the execution of your program, and for the calling of destructors at the end. (Exception: System VR3 and System VR4 linkers, Linux/ELF, and some other systems support user-defined segments; g++ on these systems requires neither the GNU linker nor collect. So if you have such a system, the answer is that you don't need either one).

If you have experience with AT&T's "cfront", this function is performed there by programs named "patch" or "munch". With GNU C++, it is performed either by the GNU linker or by a program known as "collect". The collect program is part of the gcc-2.x distribution; you can obtain the GNU linker separately as part of the "binutils" package. The latest version of binutils is 2.5.2, released November 2, 1994.

(To be technical, it's "collect2"; there were originally several alternative versions of collect, and this is the one that survived).

There are advantages and disadvantages to either choice.

Advantages of the GNU linker:

It's faster than using collect – collect basically runs the standard Unix linker on your program twice, inserting some extra code after the first pass to call the constructors. This is a sizable time penalty for large programs. The GNU linker does not require this extra pass.

GNU ld reports undefined symbols using their true names, not the mangled names (but as of 2.7.0 so does collect).

If there are undefined symbols, GNU ld reports which object file(s) refer to the undefined symbol(s).

As of binutils version 2.2, on systems that use the so-called "a.out" debug format (e.g. Suns running SunOS 4.x), the GNU linker compresses the debug symbol table considerably.

Advantages of collect:

If your native linker supports shared libraries, you can use shared libraries with collect. This used to be a strong reason *not* to use the GNU linker, but recent versions of GNU ld support linking with shared libraries on many platforms, and creating shared libraries on a few (such as Intel x86 systems that use ELF object format).

Note: using existing shared libraries (X and libc, for example) works very nicely. Generating shared libraries from g++-compiled code is another matter, generally requiring OS-dependent tricks if it is possible at all. But progress has been made recently.

As of 2.7.0, building C++ shared libraries should work fine on supported platforms (HPUX 9+, IRIX 5+, DEC UNIX (formerly OSF/1), SunOS 4, and all targets using SVR4-style ELF shared libraries).

However, as of libg++ 2.6.2, the libg++ distribution contains some patches to build libg++ as a shared library on some OSes (those listed above). Check the file '`README.SHLIB`' from that distribution.

The GNU linker has not been ported to as many platforms as g++ has, so you may be forced to use collect.

If you use collect, you don't need to get something extra and figure out how to install it; the standard gcc installation procedure will do it for you.

In conclusion, I don't see a clear win for either alternative at this point. Take your pick.

3.4 Should I use the GNU assembler, or my vendor's assembler?

This depends on your platform and your decision about the GNU linker. For most platforms, you'll need to use GAS if you use the GNU linker. For some platforms, you have no choice; check the gcc installation notes to see whether you must use GAS. But you can usually use the vendor's assembler if you don't use the GNU linker.

The GNU assembler assembles faster than many native assemblers; however, on many platforms it cannot support the local debugging format.

If you want to build shared libraries from gcc/g++ output and you are on a Sun, you must *not* use GNU as, as it cannot do position-independent code correctly yet.

On HPUX or IRIX, you must use GAS (and configure gcc with the `--with-gnu-as` option) to debug your programs. GAS is strongly recommended particularly on the HP platform because of limitations in the HP assembler.

The GAS distribution has recently been merged with the binutils distribution, so the GNU assembler and linker are now together in this package (as of binutils version 2.5.1).

3.5 Should I use the GNU C library?

At this point in time, no. The GNU C library is still very young, and libg++ still conflicts with it in some places. Use your native C library unless you know a lot about the gory details of libg++ and gnu-libc. This will probably change in the future.

3.6 Global constructors aren't being called

"I've installed gcc and it almost works, but constructors and destructors for global objects and objects at file scope aren't being called. What did I do wrong?"

It appears that you are running on a platform that requires you to install either "collect2" or the GNU linker, and you have done neither. For more information, see the section discussing the GNU linker (See Section 3.3 [use GNU linker?], page 8).

On Solaris 2.x, you shouldn't need a collect program and GNU ld doesn't run. If your global constructors aren't being called, you may need to install a patch, available from Sun, to fix your linker. The number of the "jumbo patch" that applies is 101409-03. Thanks to Russell Street (r.street@auckland.ac.nz) for this info.

It appears that on IRIX, the collect2 program is not being installed by default during the installation process, though it is required; you can install it manually by executing

```
make install-collect2
```

from the gcc source directory after installing the compiler. (I'm not certain for which versions of gcc this problem occurs, and whether it is still present).

3.7 Strange assembler errors when linking C++ programs

"I've installed gcc and it seemed to go OK, but when I attempt to link any C++ program, I'm getting strange errors from the assembler! How can that be?"

The messages in question might look something like

```
as: "/usr/tmp/cca14605.s", line 8: error: statement syntax
as: "/usr/tmp/cca14605.s", line 14: error: statement syntax
```

(on a Sun, different on other platforms). The important thing is that the errors come out at the link step, *not* when a C++ file is being compiled.

Here's what's going on: the collect2 program uses the Unix "nm" program to obtain a list of symbols for the global constructors and destructors, and it builds a little assembly language module that will permit them all to be called. If you're seeing this symptom, you have an old version of GNU nm somewhere on your path. This old version prints out symbol names in a format that the collect2 program does not expect, so bad assembly code is generated.

The solution is either to remove the old version of GNU nm from your path (and that of everyone else who uses g++), or to install a newer version (it is part of the GNU "binutils" package). Recent versions of GNU nm do not have this problem.

3.8 Other problems building libg++

"I am having trouble building libg++. Help!"

On some platforms (for example, Ultrix), you may see errors complaining about being unable to open dummy.o. On other platforms (for example, SunOS), you may see problems having to do with the type of size_t. The fix for these problems is to make libg++ by saying "make CC=gcc". According to Per Bothner, it should no longer be necessary to specify "CC=gcc" for libg++-2.3.1 or later.

"I built and installed libg++, but g++ can't find it. Help!"

The string given to '`configure`' that identifies your system must be the same when you install libg++ as it was when you installed gcc. Also, if you used the `--prefix` option to install gcc somewhere other than '`/usr/local`', you must use the same value for `--prefix` when installing libg++, or else g++ will not be able to find libg++.

The toplevel Makefile in the libg++ 2.6.2 distribution is broken, which along with a bug in g++ 2.6.3 causes problems linking programs that use the libstdc++ complex classes. A patch for this is available from '`ftp.cygnus.com:pub/g++/libg++-2.6.2-fix.gz`'.

3.9 But I'm *still* having problems with `size_t`!

"I did all that, and I'm *still* having problems with disagreeing definitions of size_t, SIZE_TYPE, and the type of functions like `strlen`."

The problem may be that you have an old version of '`_G_config.h`' lying around. As of libg++ version 2.4, '`_G_config.h`', since it is platform-specific, is inserted into a different directory; most include files are in '`$prefix/lib/g++-include`', but this file now lives in '`$prefix/$arch/include`'. If, after upgrading your libg++, you find that there is an old copy of '`_G_config.h`' left around, remove it, otherwise g++ will find the old one first.

3.10 Do I need to rebuild libg++ to go with my new g++?

"After I upgraded g++ to the latest version, I'm seeing undefined symbols."

or

"If I upgrade to a new version of g++, do I need to reinstall libg++?"

As a rule, the first two digits of your g++ and libg++ should be the same. Normally when you do an upgrade in the "minor version number" (2.5.7 to 2.5.8, say) there isn't a need to rebuild libg++, but there have been a couple of exceptions in the past.

4 User Problems

4.1 How to silence "unused parameter" warnings

"When I use `-Wall` (or `-Wunused`), g++ warns about unused parameters. But the parameters have to be there, for use in derived class functions. How do I get g++ to stop complaining?"

The answer is to simply omit the names of the unused parameters when defining the function. This makes clear, both to g++ and to readers of your code, that the parameter is unused. For example:

```
int Foo::bar(int arg) { return 0; }
```

will give a warning for the unused parameter `arg`. To suppress the warning write

```
int Foo::bar(int) { return 0; }
```

4.2 g++ objects to a declaration in a case statement

"The compiler objects to my declaring a variable in one of the branches of a case statement. Earlier versions used to accept this code. Why?"

The draft standard does not allow a goto or a jump to a case label to skip over an initialization of a variable or a class object. For example:

```
switch ( i )
  {
    case 1:
      Object obj(0);
```

```
          ...
    break;
          case 2:
    ...
    break;
          }
```

The reason is that `obj` is also in scope in the rest of the switch statement.

As of version 2.7.0, the compiler will object that the jump to the second case level crosses the initialization of `obj`. Older compiler versions would object only if class Object has a destructor. In either case, the solution is to add a set of curly braces around the case branch:

```
          case 1:
           {
            Object obj(0);
            ...
    break;
           }
```

4.3 gcc 2.5.x broke my code! Changes in function overloading

"I have a program that worked just fine with older g++ versions, but as of version 2.5.x it doesn't work anymore. Help!"

While it's always possible that a new bug has been introduced into the compiler, it's also possible that you have been relying on bugs in older versions of g++. For example, version 2.5.0 was the first version of g++ to correctly implement the "hiding rule." That is, if you have an overloaded function in a base class, and in a derived class you redefine one of the names, the other names are effectively "hidden". *All* the names from the baseclass need to be redefined in the derived class. See section 13.1 of the ARM: "A function member of a derived class is *not* in the same scope as a function member of the same name in a base class".

Here's an example that is handled incorrectly by g++ versions before 2.5.0 and correctly by newer versions:

```
class Base {
public:
    void foo(int);
};

class Derived : public Base {
public:
    void foo(double);  // note that Base::foo(int) is hidden
};

main() {
    Derived d;
    d.foo(2); // Derived::foo(double), not Base::foo(int), is called
}
```

4.4 Where can I find a demangler?

A g++-compatible demangler named `c++filt` can be found in the '`binutils`' distribution. This distribution (which also contains the GNU linker) can be found at any GNU archive site.

As of version 2.7.0, `c++filt` is included with gcc and is installed automatically. Even better, it is used by the `collect` linker, so you don't see mangled symbols anymore.

4.5 Where can I find a version of etags for C++?

The libg++ distribution contains a version of etags that works for C++ code. Look in '`libg++/utils`'. It's not built by default when you install libg++, but you can cd to that directory and type

```
make etags
```

after you've installed libg++.

4.6 Linker reports undefined symbols for static data members

"g++ reports undefined symbols for all my static data members when I link, even though the program works correctly for compiler XYZ. What's going on?"

The problem is almost certainly that you don't give definitions for your static data members. If you have

```
class Foo {
...
void method();
static int bar;
};
```

you have only declared that there is an int named Foo::bar and a member function named Foo::method that is defined somewhere. You still need to defined BOTH method() and bar in some source file. According to the draft ANSI standard, you must supply an initializer, such as

```
int Foo::bar = 0;
```

in one (and only one) source file.

4.7 What does "Internal compiler error" mean?

It means that the compiler has detected a bug in itself. Unfortunately, g++ still has many bugs, though it is a lot better than it used to be. If you see this message, please send in a complete bug report (see next section).

4.8 I think I have found a bug in g++.

"I think I have found a bug in g++, but I'm not sure. How do I know, and who should I tell?"

First, see the excellent section on bugs and bug reports in the gcc manual (which is included in the gcc distribution). As a short summary of that section: if the compiler gets a fatal signal, for any input, it's a bug (newer versions of g++ will ask you to send in a bug report when they detect an error in themselves). Same thing for producing invalid assembly code.

When you report a bug, make sure to describe your platform (the type of computer, and the version of the operating system it is running) and the version of the compiler that you are running. See the output of the command `g++ -v` if you aren't sure. Also provide enough code so that the g++ maintainers can duplicate your bug. Remember that the maintainers won't have your header files; one possibility is to send the output of the preprocessor (use `g++ -E` to get this). This is what a "complete bug report" means.

I will add some extra notes that are C++-specific, since the notes from the gcc documentation are generally C-specific.

First, mail your bug report to "bug-g++@prep.ai.mit.edu". You may also post to gnu.g++.bug, but it's better to use mail, particularly if you have any doubt as to whether your news software generates correct reply addresses. Don't mail C++ bugs to bug-gcc@prep.ai.mit.edu.

If your bug involves libg++ rather than the compiler, mail to bug-lib-g++@prep.ai.mit.edu. If you're not sure, choose one, and if you guessed wrong, the maintainers will forward it to the other list.

Second, if your program does one thing, and you think it should do something else, it is best to consult a good reference if in doubt. The standard reference is the draft working paper from the ANSI/ISO C++ standardization committee, which you can get on the net. For PostScript and PDF (Adobe Acrobat) versions, see the archive at ftp://research.att.com/dist/stdc++/WP. For HTML and ASCII versions, see ftp://ftp.cygnus.com/pub/g++. On the World Wide Web, see http://www.cygnus.com/~mrs/wp-draft.

An older standard reference is "The Annotated C++ Reference Manual", by Ellis and Stroustrup (copyright 1990, ISBN #0-201-51459-1). This is what they're talking about on the net when they refer to "the ARM". But you should know that changes have been made to the language since then.

The ANSI/ISO C++ standards committee have adopted some changes to the C++ language since the publication of the original ARM, and newer versions of g++ (2.5.x and later) support some of these changes, notably the mutable keyword (added in 2.5.0), the bool type (added in 2.6.0), and changes in the scope of variables defined in for statements (added in 2.7.0). You can obtain an addendum to the ARM explaining these changes by FTP from ftp.std.com in '`/AW/stroustrup2e/new_iso.ps`'.

Note that the behavior of (any version of) AT&T's "cfront" compiler is NOT the standard for the language.

4.9 Porting programs from other compilers to g++

"I have a program that runs on <some other C++ compiler>, and I want to get it running under g++. Is there anything I should watch out for?"

Note that g++ supports many of the newer keywords that have recently been added to the language. Your other C++ compiler may not support them, so you may need to rename variables and members that conflict with these keywords.

There are two other reasons why a program that worked under one compiler might fail under another: your program may depend on the order of evaluation of side effects in an expression, or it may depend on the lifetime of a temporary (you may be assuming that a temporary object "lives" longer than the standard guarantees). As an example of the first:

```
void func(int,int);

int i = 3;
func(i++,i++);
```

Novice programmers think that the increments will be evaluated in strict left-to-right order. Neither C nor C++ guarantees this; the second increment might happen first, for example. func might get 3,4, or it might get 4,3.

The second problem often happens with classes like the libg++ String class. Let's say I have

```
String func1();
void func2(const char*);
```

and I say

```
func2(func1());
```

because I know that class String has an "operator const char*". So what really happens is

```
func2(func1().convert());
```

where I'm pretending I have a convert() method that is the same as the cast. This is unsafe in g++ versions before 2.6.0, because the temporary String object may be deleted after its last use (the call to the conversion function), leaving the pointer pointing to garbage, so by the time func2 is called, it gets an invalid argument.

Both the cfront and the old g++ behaviors are legal according to the ARM, but the powers that be have decided that compiler writers were given too much freedom here.

The ANSI C++ committee has now come to a resolution of the lifetime of temporaries problem: they specify that temporaries should be deleted at end-of-statement (and at a couple of other points). This means that g++ versions before 2.6.0 now delete temporaries too early, and cfront deletes temporaries too late. As of version 2.6.0, g++ does things according to the new standard.

For now, the safe way to write such code is to give the temporary a name, which forces it to live until the end of the scope of the name. For example:

```
String& tmp = func1();
func2(tmp);
```

Finally, like all compilers (but especially C++ compilers, it seems), g++ has bugs, and you may have tweaked one. If so, please file a bug report (after checking the above issues).

4.10 Why does g++ mangle names differently from other C++ compilers?

See the answer to the next question.

4.11 Why can't g++ code link with code from other C++ compilers?

"Why can't I link g++-compiled programs against libraries compiled by some other C++ compiler?"

Some people think that, if only the FSF and Cygnus Support folks would stop being stubborn and mangle names the same way that, say, cfront does, then any g++-compiled program would link successfully against any cfront-compiled library and vice versa. Name mangling is the least of the problems. Compilers differ as to how objects are laid out, how multiple inheritance is implemented, how virtual function calls are handled, and so on, so if the name mangling were made the same, your programs would link against libraries provided from other compilers but then crash when run. For this reason, the ARM *encourages* compiler writers to make their name mangling different from that of other compilers for the same platform. Incompatible libraries are then detected at link time, rather than at run time.

4.12 What documentation exists for g++ 2.x?

Relatively little. While the gcc manual that comes with the distribution has some coverage of the C++ part of the compiler, it focuses mainly on the C compiler (though the information on the "back end" pertains to C++ as well). Still, there is useful information on the command line options and the #pragma interface and #pragma implementation directives in the manual, and there is a useful section on template instantiation in the 2.6 version. There is a Unix-style manual entry, "g++.1", in the gcc-2.x distribution; the information here is a subset of what is in the manual.

You can buy a nicely printed and bound copy of this manual from the FSF; see above for ordering information.

For versions 2.6.2 and later, the gcc/g++ distribution contains the gcc manual in PostScript. Also, Postscript versions of GNU documentation in U.S. letter format are available by anonymous FTP to primus.com in /pub/gnu-ps. The same, in A4 format, are on liasun3.epfl.ch in /pub/gnu/ps-doc.

A draft of a document describing the g++ internals appears in the gcc distribution (called g++int.texi); it is still incomplete.

4.13 Problems with the template implementation

g++ does not implement a separate pass to instantiate template functions and classes at this point; for this reason, it will not work, for the most part, to declare your template functions in one file and define them in another. The compiler will need to see the entire definition of the function, and will generate a static copy of the function in each file in which it is used.

(The experimental template repository code (see See Section 1.3 [repository], page 3) that can be added to 2.7.0 does implement a separate pass, but there is still no searching of files that the compiler never saw).

For version 2.6.0, however, a new switch `-fno-implicit-templates` was added; with this switch, templates are expanded only under user control. I recommend that all g++ users that use templates read the section "Template Instantiation" in the gcc manual (version 2.6.x and newer). g++ now supports explicit template expansion using the syntax from the latest C++ working paper:

```
template class A<int>;
template ostream& operator << (ostream&, const A<int>&);
```

As of version 2.6.3, there are still a few limitations in the template implementation besides the above (thanks to Jason Merrill for this info):

1. Static data member templates are not supported. You can work around this by explicitly declaring the static variable for each template specialization:

   ```
   template <class T> struct A {
     static T t;
   };

   template <class T> T A<T>::t = 0; // gets bogus error
   int A<int>::t = 0;                // OK (workaround)
   ```

   (still a limitation in 2.7.0)

2. Template member names are not available when defining member function templates.

   ```
   template <class T> struct A {
     typedef T foo;
     void f (foo);
     void g (foo arg) { ... }; // this works
   };

   template <class T> void A<T>::f (foo) { } // gets bogus error
   ```

3. Templates are instantiated using the parser. This results in two problems:

   a) Class templates are instantiated in some situations where such instantiation should not occur.

   ```
   template <class T> class A { };
   A<int> *aip = 0; // should not instantiate A<int> (but does)
   ```

   b) Function templates cannot be inlined at the site of their instantiation.

   ```
   template <class T> inline T min (T a, T b) { return a < b ? a : b; }

   void f () {
     int i = min (1, 0);           // not inlined
   }

   void g () {
     int j = min (1, 0);           // inlined
   }
   ```

   A workaround that works in version 2.6.1 and later is to specify

```
extern template int min (int, int);
```

before `f()`; this will force it to be instantiated (though not emitted).

4. Member function templates are always instantiated when their containing class is. This is wrong.

4.14 I get undefined symbols when using templates

(Thanks to Jason Merrill for this section).

g++ does not automatically instantiate templates defined in other files. Because of this, code written for cfront will often produce undefined symbol errors when compiled with g++. You need to tell g++ which template instances you want, by explicitly instantiating them in the file where they are defined. For instance, given the files

'`templates.h`':

```
template <class T>
class A {
public:
  void f ();
  T t;
};

template <class T> void g (T a);
```

'`templates.cc`':

```
#include "templates.h"

template <class T>
void A<T>::f () { }

template <class T>
void g (T a) { }
```

main.cc:

```
#include "templates.h"

main ()
{
  A<int> a;
  a.f ();
  g (a);
}
```

compiling everything with `g++ main.cc templates.cc` will result in undefined symbol errors for '`A<int>::f ()`' and '`g (A<int>)`'. To fix these errors, add the lines

```
template class A<int>;
template void g (A<int>);
```

to the bottom of '`templates.cc`' and recompile.

4.15 I get multiply defined symbols using templates

You may be running into a bug that was introduced in version 2.6.1 (and is still present in 2.6.3) that generated external linkage for templates even when neither `-fexternal-templates` nor `-fno-implicit-templates` is specified. There is a patch for this problem at ftp.cygnus.com:pub/g++/gcc-2.6.3-template-fix. I recommend either applying the patch or using `-fno-implicit-templates` together with explicit template instantiation as described in previous sections.

This bug is fixed in 2.7.0.

4.16 Does g++ support the Standard Template Library?

From Per Bothner:

The Standard Template Library (STL) uses many of the extensions that the ANSI/ISO committee has made to templates, and g++ doesn't support some of these yet. So if you grab HP's free implementation of STL it isn't going to work. However, libg++-2.6.2 contains a hacked version of STL, based on work by Carsten Bormann, which permits gcc-2.6.3 to compile at least the containers. A full implementation is going to need improved template support, which will take a while yet.

As of libg++-2.7.0 and gcc-2.7.0, I've succeeded in making many short STL example programs work, though there are still a number of bugs and limitations.

4.17 What are the differences between g++ and the ARM specification of C++?

As of version 2.7.0, g++ has exception support on most but not all platforms (no support on MIPS-based platforms yet), but it doesn't work right if optimizaton is enabled, which means the exception implementation is still not really ready for production use.

Some features that the ANSI/ISO standardization committee has voted in that don't appear in the ARM are supported, notably the `mutable` keyword, in version 2.5.x. 2.6.x adds support for the built-in boolean type `bool`, with constants `true` and `false`. The beginnings of run-time type identification are present, so there are more reserved words: `typeid`, `static_cast`, `reinterpret_cast`, `const_cast`, and `dynamic_cast`.

As with any beta-test compiler, there are bugs. You can help improve the compiler by submitting detailed bug reports.

One of the weakest areas of g++ other than templates is the resolution of overloaded functions and operators in complex cases. The usual symptom is that in a case where the ARM says that it is ambiguous which function should be chosen, g++ chooses one (often the first one declared). This is usually not a problem when porting C++ code from other compilers to g++, but shows up as errors when code developed under g++ is ported to other compilers. (I believe this is no longer a significant problem in 2.7.0).

[A full bug list would be very long indeed, so I won't put one here. I may add a list of frequently-reported bugs and "non-bugs" like the static class members issue mentioned above].

4.18 Will g++ compile InterViews? The NIH class library?

The NIH class library uses a non-portable, compiler-dependent hack to initialize itself, which makes life difficult for g++ users. It will not work without modification, and I don't know what modifications are required or whether anyone has done them successfully.

In short, it's not going to happen any time soon (previous FAQs referred to patches that a new NIHCL release would hopefully contain, but this hasn't happened).

[From Steinar Bang <steinarb@idt.unit.no>]

InterViews 3.1 compiles and runs with gcc-2.3.3 and libg++-2.3, except that the "doc" application immediately dumps core when you try to run it. There is also a small glitch with idraw.

There is a patch for InterViews 3.1 from Johan Garpendahl <garp@isy.liu.se> available for FTP from site "ugle.unit.no". It is in the file

'`/pub/X11/contrib/InterViews/g++/3.1-beta3-patch`'.

This fixes two things: the Doc coredump, and the pattern menu of idraw. Read the instructions at the start of the file.

I think that as of version 2.5.6, the standard g++ will compile the standard 3.1 InterViews completely successfully. I'd appreciate a confirmation.

4.19 Debugging on SVR4 systems

"How do I get debugging to work on my System V Release 4 system?"

Most systems based on System V Release 4 (except Solaris) encode symbolic debugging information in a format known as 'DWARF'.

Although the GNU C compiler already knows how to write out symbolic debugging information in the DWARF format, the GNU C++ compiler does not yet have this feature, nor is it likely to in the immediate future.

Ron Guilmette has done a great deal of work to try to get the GNU C++ compiler to produce DWARF format symbolic debugging information (for C++ code) but he gave up on the project because of a lack of funding and/or interest from the g++ user community. If you have a strong desire to see this project completed, contact Ron at <rfg@netcom.com>.

In the meantime, you *can* get g++ debugging under SVR4 systems by configuring gcc with the `--with-stabs` option. This causes gcc to use an alternate debugging format, one more like that used under SunOS4. You won't need to do anything special to GDB; it will always understand the "stabs" format.

4.20 X11 conflicts with libg++ in definition of String

"X11 and Motif define String, and this conflicts with the String class in libg++. How can I use both together?"

One possible method is the following:

```
#define String XString
#include <X11/Intrinsic.h>
/* include other X11 and Motif headers */
```

```
#undef String
```

and remember to use the correct `String` or `XString` when you declare things later.

4.21 Why can't I assign one stream to another?

[Thanks to Per Bothner and Jerry Schwarz for this section.]

Assigning one stream to another seems like a reasonable thing to do, but it's a bad idea. Usually, this comes up because people want to assign to `cout`. This is poor style, especially for libraries, and is contrary to good object-oriented design. (Libraries that write directly to `cout` are less flexible, modular, and object-oriented).

The iostream classes do not allow assigning to arbitrary streams, because this can violate typing:

```
ifstream foo ("foo");
istrstream str(...);
foo = str;
foo->close ();  /* Oops! Not defined for istrstream! */
```

The original cfront implementation of iostreams by Jerry Schwarz allows you to assign to `cin`, `cout`, `cerr`, and `clog`, but this is not part of the draft standard for iostreams and generally isn't considered a good idea, so standard-conforming code shouldn't use this technique.

The GNU implementation of iostream did not support assigning to `cin`, `cout`, `cerr`, and `clog` for quite a while, but it now does, for backward compatibility with cfront iostream (versions 2.6.1 and later of libg++).

The ANSI/ISO C++ Working Paper does provide ways of changing the streambuf associated with a stream. Assignment isn't allowed; there is an explicit named member that must be used.

However, it is not wise to do this, and the results are confusing. For example: `fstream::rdbuf` is supposed to return the *original* filebuf, not the one you assigned. (This is not yet implemented in GNU iostream.) This must be so because `fstream::rdbuf` is defined to return a `filebuf *`.

5 What are the rules for shipping code built with g++ and libg++?

“Is it is possible to distribute programs for profit that are created with g++ and use the g++ libraries?”

I am not a lawyer, and this is not legal advice. In any case, I have little interest in telling people how to violate the spirit of the GNU licenses without violating the letter. This section tells you how to comply with the intention of the GNU licenses as best I understand them.

The FSF has no objection to your making money. Its only interest is that source code to their programs, and libraries, and to modified versions of their programs and libraries, is always available.

The short answer is that you do not need to release the source to your program, but you can't just ship a stripped executable either, unless you use only the subset of libg++ that includes the iostreams classes (see discussion below) or the new libstdc++ library (available in libg++ 2.6.2 and later).

Compiling your code with a GNU compiler does not affect its copyright; it is still yours. However, in order to ship code that links in a GNU library such as libg++ there are certain rules you must follow. The rules are described in the file COPYING.LIB that accompanies gcc distributions; it is also included in the libg++ distribution. See that file for the exact rules. The agreement is called the Library GNU Public License or LGPL. It is much "looser" than the GNU Public License, or GPL, that covers must GNU programs.

Here's the deal: let's say that you use some version of libg++, completely unchanged, in your software, and you want to ship only a binary form of your code. You can do this, but there are several special requirements. If you want to use libg++ but ship only object code for your code, you have to ship source for libg++ (or ensure somehow that your customer already has the source for the exact version you are using), and ship your application in linkable form. You cannot forbid your customer from reverse-engineering or extending your program by exploiting its linkable form.

Furthermore, if you modify libg++ itself, you must provide source for your modifications (making a derived class does not count as modifying the library – that is "a work that uses the library").

For certain portions of libg++ that implement required parts of the C++ language (such as iostreams and other standard classes), the FSF has loosened the copyright requirement still more by adding the "special exception" clause, which reads as follows:

> As a special exception, if you link this library with files compiled with GCC to produce an executable, this does not cause the resulting executable to be covered by the GNU General Public License. This exception does not however invalidate any other reasons why the executable file might be covered by the GNU General Public License.

If your only use of libg++ uses code with this exception, you may ship stripped executables or license your executables under different conditions without fear of violating an FSF copyright. It is the intent of FSF and Cygnus that, as the other classes required by the ANSI/ISO draft standard are developed, these will also be placed under this "special exception" license. The code in the new libstdc++ library, intended to implement standard classes as defined by ANSI/ISO, is also licensed this way.

To avoid coming under the influence of the LGPL, you can link with '`-liostream`' rather than '`-lg++`' (for version 2.6.x and earlier), or '`-lstdc++`' now that it is available. In version 2.7.0 all the standard classes are in '`-lstdc++`'; you can do the link step with `c++` instead of `g++` to search only the '`-lstdc++`' library and avoid the LGPL'ed code in '`-lg++`'.

Appendix A Concept Index

Table of Contents