Xref: bloom-picayune.mit.edu gnu.g++.help:2789 comp.lang.c++:34579 news.answers:4674 Path: bloom-picayune.mit.edu!enterpoop.mit.edu!news.media.mit.edu!micro-heart-of-gold.mit.edu!news.bbn.com!noc.near.net!uunet!zaphod.mps.ohio-state.edu!magnus.acs.ohio-state.edu!usenet.ins.cwru.edu!agate!agate!usenet From: jbuck@ohm.berkeley.edu (Joe Buck) Newsgroups: gnu.g++.help,comp.lang.c++,news.answers Subject: FAQ for g++ and libg++, texinfo version [Revised 16 Dec 1992] Followup-To: poster Date: 17 Dec 1992 00:21:43 GMT Organization: University of California, Berkeley Lines: 665 Approved: news-answers-request@MIT.edu Expires: +1 month Message-ID: ¡1goh6nINN4mo@agate.berkeley.edu¿ NNTP-Posting-Host: forney.berkeley.edu

Archive-name: g++-FAQ/texi Last-modified: 16 Dec 1992 Frequency: bimonthly

[This is the texinfo version. If you don't know what texinfo is, then you probably want to use the companion plain-text version.]

————- cut here ————————————————-

G++ FAQ

Frequently asked questions about the Gnu C++ compiler

Joe Buck

1 Obtaining Source Code

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

First, you may already have it if you have gcc for your platform; see the answer to question 3.

If you're trying to find g++, you can get it by anonymous FTP, by anonymous UUCP, by buying a tape from the Free Software Foundation, or by getting it from a friend that has it.

Here is a list of anonymous FTP archive sites for Gnu software.

```
Japan: ftp.cs.titech.ac.jp, utsun.s.u-tokyo.ac.jp:ftpsync/prep
Australia: archie.au:gnu
Europe: src.doc.ic.ac.uk:gnu, ftp.informatik.tu-muenchen.de,
    ftp.informatik.rwth-aachen.de:pub/gnu,
    nic.funet.fi:pub/gnu, ugle.unit.no, isy.liu.se,
    ftp.stacken.kth.se, sunic.sunet.se, ftp.win.tue.nl,
    ftp.diku.dk, ftp.eunet.ch, archive.eu.net
United States: wuarchive.wustl.edu, ftp.cs.widener.edu,
    uxc.cso.uiuc.edu, col.hp.com, gatekeeper.dec.com:pub/GNU,
    ftp.uu.net:packages/gnu
```

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

Users of the HP Precision Architecture (HP-9000/7xx and HP-9000/8xx) may want to use the "2.3.1-u2" version available from the University of Utah, site jaguar.cs.utah.edu. This version supports debugging provided that gas is used (you need the version of gas available from the same place) and has several bug fixes that weren't done in time for the 2.3.2 release. Precompiled binaries of gcc and gdb can also be obtained from that site. Hopefully this will all be folded in to 2.4.

UUNET customers can get Gnu sources from UUNET via UUCP. For information on how to order tapes from FSF, write to gnu@prep.ai.mit.edu.

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!

1.2 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.

DJ Delorie has ported gcc/g++ to MS-DOS; this port is popularly known as "DJGPP" (the P's stand for "plus"). It can be found on many FTP archive sites; its "home" is on grape.ecs.clarkson.edu, directory ~ftp/pub/msdos/djgpp. Make sure you're retrieving the current version, which should indicate that it is a port of gcc-2.2.2 (2.3.1 and 2.3.2 have been released since then, but I don't think the DOS port is done yet). It is also available on site wuarchive.wustl.edu, in directory mirrors/msdos/djgpp, and many other places as well.

For information on Amiga ports of gcc/g++, retrieve the file /pub/gnu/MicrosPorts/Amiga from prep.ai.mit.edu, or write to Leonard Norrgard <vinsci@nic.funet.fi>, who I hope won't be too upset that I mentioned his name here.

A port of gcc-2.3.1 to the Atari ST can be found on the site "atari.archive.umich.edu", under /atari/Gnustuff/Tos, along with many other Gnu programs. See the FAQ for the Usenet group "comp.sys.atari.st" for more information.

There are two different ports of gcc-2.2.2 (and g++) to OS/2, the so-called EMX port, which requires a particular Unix emulator, and a port called "gcc/2", which runs native. The latter port uses a rather buggy port of the BSD libc. For more information ask around on comp.os.os2.programmer. gcc/2 can be obtained by FTP from

```
ftp-os2.nmsu.edu (128.123.35.151) in /pub/os2/2.0/programming/gcc2-222
luga.latrobe.edu.au (131.172.2.2) in /pub/os2/2.0/programming/gcc2-222
```

Eberhard Mattes did the EMX port. Tevor Lampre did the gcc/2 port. Their addresses are mattes@azu.informatik.uni-stuttgart.de and mmtl@cc.flinders.edu.au, repectively.

Because the legal policies of Apple threaten the long-term goals of FSF, as well as the concept of free software, no support will be lent to efforts to port Gnu software to Macintosh or other Apple hardware.

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

"I keep hearing people talking about g++ 2.3.2 (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".

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

The latest "2.x" version of gcc/g++ is 2.3.2, released Nov 27, 1992. The latest version of libg++ is 2.3, released Dec 9, 1992.

For some non-Unix platforms, 2.2.2 may be the latest compiler that has been ported. libg++ 2.3 will not compile with gcc-2.2.2. Also, due to a newly introduced compiler bug, libg++ 2.2 will not compile with gcc-2.3.1 or gcc-2.3.2.

The latest "1.x" version of gcc is 1.42, and the latest "1.x" version of g++ is 1.42.0.

1.5 I have gcc-2.2.2 and libg++-2.2. Should I upgrade to the new versions?

Unfortunately, this question cannot be answered with a simple yes or no. gcc-2.3.x made some significant improvements in template support; however, at the same time, quite a few new bugs were introduced in 2.3.1, particularly in resolving of overloaded functions. 2.3.2 fixed some of them, but serious problems remain (for example, 2.3.2 cannot compile some perfectly correct code in libg++-2.2). Given this, I would recommend that those not using templates wait for a more stable release (gcc-2.3.3 should be out soon). If you're using templates and 2.2.2 won't cut it for you, by all means upgrade.

2 Installation Issues and Problems

2.1 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 g++-2.3.2, 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.

2.2 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).

2.3 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 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.

(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.

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

Advantages of collect:

If your native linker supports shared libraries, you can use shared libraries with collect. The Gnu linker does not (yet) support shared libraries.

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.

2.4 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.

2.5 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.

2.6 Problems building libg++ on Ultrix

"I am having trouble building libg++-2.2 on Ultrix [and possibly other systems]. I get errors referring to "dummy.o". Help!"

(The errors on Ultrix end with something like

```
ldopen: cannot open dummy.o
nm: Error: cannot open dummy.o
ldopen: cannot open dummy.o
nm: Error: cannot open dummy.o
nm failed to find FUNC in dummy.o!
sh: -1: bad number
*** Error code 1
```

and there may be similar problems on other systems).

The fix for this is to make libg++ by saying "make CC=gcc".

3 User Problems

3.1 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.

3.2 g++ won't accept the placement new syntax.

"I have a program that uses the "placement syntax" of operator new, e.g.

```
new (somewhere) T;
```

and g++ won't accept it."

Up until version 2.3.1, g++ accepted an alternate form of the placement syntax, for historical reasons; use

```
new {somewhere} T;
```

if you are using g++-2.2.2 or older.

As of 2.3.1, g++ finally fixed this, using the standard ARM syntax for "placement new". A few remaining glitches were fixed in 2.3.2. The only remaining problem is with declarators for pointers to functions;

```
new (void (*)(int));  // confuses gcc 2.3.2
new (a) (void (*)(int)); // ditto
```

These can be worked around with a typedef:

```
typedef void (*fun)(int);
new fun;
new (a) fun;
```

3.3 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. Same thing for producing invalid assembly code.

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

First, mail your bug report to "bug-g++@prep.ai.mit.edu". You may also post to gnu.bug.g++, but it's better to use mail, particularly if you 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-libg++@prep.ai.mit.edu. If you're not sure, you could send your bug to both lists.

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 Annotated C++ Reference Manual", by Ellis and Stroustrup (copyright 1990, ISBN #0-201-51459-1); the reference manual, without annotations, also appears in Stroustrup's "The C++ Programming Language, Second Edition" (copyright 1991, ISBN #0-201-53992-6). Both are published by Addison-Wesley.

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

3.4 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?"

First, see the questions on placement new syntax and static data members.

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, 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.

If you think this is ugly, you should know that the ANSI C++ committee is STILL debating the lifetime-of-temporaries problem.

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.

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

See the answer to the next question.

3.6 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 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.

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

Almost none. The gcc manual describes the C front end, and also the back end, which is shared by the C++ compiler, but there is almost no documentation for the C++ front end. There is a Unix-style manual entry, "g++.1", in the gcc-2.x distribution; this describes the extra command-line options that g++ supports, and the #pragma interface and #pragma implementation directives.

A draft of a document describing the g++ internals appears in the 2.3.2 distribution (called g++int.texi); it is still incomplete.

Work is proceeding on a user g++ document; with luck it will appear in the next release.

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

The chief thing missing from g++ that is in the ARM is exceptions (the other major compilers do not have exceptions either). There are bits and pieces of exception code present, but it is not presently usable.

While, as of 2.3.1, the "placement new" syntax finally agrees with the ARM, there are still some problems with it.

The template implementation is still new. The implementation in 2.3.2 represents a considerable improvement over that of previous releases, however. Still, it has many bugs.

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.

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

[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].

3.9 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.

Brendan Kehoe of Cygnus is working on getting NIHCL to build with g++. He says, "The NIHCL release will hopefully contain patches to gcc 2.3 to let it build."

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

The C++ compiler part of gcc-2.3 (when released) *should* be able to compile InterViews release 3.1 (when released) out of the box (no patches required on either side).

[Now that 2.3.1 is out, is this true?]

3.10 Debugging on SVR4 systems

"When I use the -g flag on C++ code on a System V Release 4 system, I get lots of undefined symbols at link time. Why? Help!"

[From Ron Guilmette:] The changes needed to get the g++ front-end to generate proper DWARF style debugging information for System V Release 4 are not yet completed, nor will they be until g++ version 2.4 (at the earliest).

There is nothing that you (as an end-user) can do to correct this problem. (It is actually *many* problems, and they are all very complex.) Until the g++ maintainers have time to fix this, you should simply *avoid* using the -g option when using g++ on SVR4.

4 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.

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").

Table of Contents