# Building GCC 9 on Ubuntu Linux

Posted on October 7, 2016 by Paul

## Updated 4 May 2019

This is a short article about compiling *GCC* 9.1 from sources on *Ubuntu* 18.04 64 bits. The default version of *GCC* on *Ubuntu* 18.04 is 7.3 which is not bad, however version 9 has complete C++11, C++14, C++17 support and experimental C++2a support. *GCC* 9 has *C11* and *C++14* support enabled by default, no need to add *-std=c11* or *-std=c++14*.

First, let's make sure that we have an up to date system:

```
1 sudo apt update
2 sudo apt upgrade
```

Now, install the default *GCC* toolchain with:

```
1 sudo apt install build-essential
```

Next, we'll download the *GCC 9* source and prerequisites from http://gcc.gnu.org/mirrors.html:

```
1 cd ~
2 wget https://ftpmirror.gnu.org/gcc/gcc-9.1.0/gcc-9.1.0.tar.gz
3 tar xf gcc-9.1.0.tar.gz
4 cd gcc-9.1.0
5 contrib/download_prerequisites
```

At this point, we can *configure* the build. In order to keep the system clean, we will use */usr/local/gcc-9.1* for the installation folder and append the suffix *-9.1* to the *GCC* compilers. You typically don't want to mess the system's default *GCC* because other packages may depend on the default version.

```
1 cd ~
2 mkdir build && cd build
3 ../gcc-9.1.0/configure -v --build=x86_64-linux-gnu --host=x86_64-linux-gnu --t
```

Now, we are ready to build *GCC*, you typically want to pass twice the number of your computer cores to the *make* command in order to speed up the build. I have a quad-

core system, so I will use 8 parallel jobs to build *GCC*:

```
make -j 8
```

Depending on the speed of your computer the build phase could take from about 30 minutes to a few hours.

Once the above phase is finished, you can install the built *GCC* with:

```
sudo make install-strip
```

If you want to permanently add the compilers to your system's path, add the next two lines at the end of your *.bashrc* file:

```
export export PATH=/usr/local/gcc-9.1/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/gcc-9.1/lib64:$LD_LIBRARY_PATH
```

Don't know how to open *.bashrc* ? No problem, you can find it from your Terminal:

```
cd ~
gedit .bashrc
```

Paste at the end *.bashrc* the above export lines, save the file and close gedit. Now, you just need to instruct Bash to reload .bashrc (this is automatically done when you restart your machine):

```
. .bashrc
```

Time to test our shiny new compilers. Open your favorite text editor and copy the next piece of code (I assume you'll save the file as *test_lambda.cpp*):

```
// C++14 generalized lambda (could use "auto" for the type of a parameter)
#include <iostream>

int main() {
    std::cout << [](auto a, auto b) { return a + b; } (5, 6) << std::endl;
    std::cout << [](auto a, auto b) { return a + b; } (5.23, 6.45) << std::end
    return 0;
}
```

The code uses a generalized lambda (we could use auto for the type of the parameters), this was introduced in the *C++14* standard. We can compile and test the above program with:

```
g++-9.1 -Wall -pedantic test_lambda.cpp -o test_lambda
./test_lambda
11
11.68
```

As mentioned at the beginning of this article, *GCC* 9.1 has complete support for

*C++17*. Let's test an example of using *C++17* modification to static_assert:

```
1  #include <type_traits>
2  #include <iostream>
3
4  struct A {
5      int foo;
6  };
7
8  struct B {
9      int foo = 0;
10 };
11
12 template <typename T>
13 void print(const T& a){
14     static_assert(std::is_pod<T>::value);
15     std::cout << a.foo << '\n';
16 }
17
18 int main() {
19     A x{1};
20     B y{2};
21     B z;
22
23     print<A>(x);
24     print<B>(y);
25     print<B>(z);
26
27     return 0;
28 }
```

You can compile the above code if you pass *std=c++17* to the compiler and you should get a compiler error triggered by lines 14 and 20 from the above code

```
1  g++-9.1 -std=c++17 -Wall -pedantic test_assert.cpp -o test_assert
2  test_assert.cpp: In instantiation of 'void print(const T&) [with T = B]':
3  test_assert.cpp:24:13:   required from here
4  test_assert.cpp:14:33: error: static assertion failed
5     14 |     static_assert(std::is_pod<T>::value);
6        |                                 ^~~~~
```

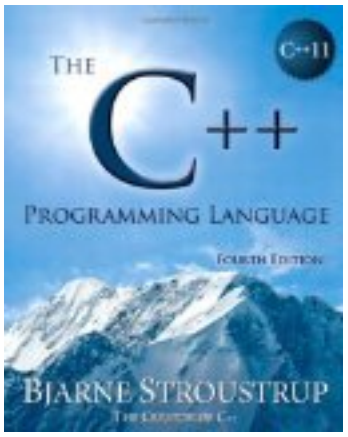
If you are a Fortran programmer, you can use some of the Fortran 2008 features like *do concurrent* with gfortran-9.1:

```
1  integer,parameter::mm=100000
2  real::a(mm), b(mm)
3  real::fact=0.5
4
5  ! initialize the arrays
6  ! ...
7
8  do concurrent (i = 1 : mm)
9      a(i) = a(i) + b(i)
10 enddo
11
12 end
```

The above code can be compiled with (supposing you've named it *test_concurrent_do.f90*s):

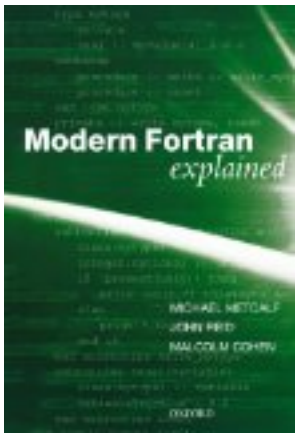
```
gfortran-9.1 test_concurrent_do.f90 -o test_concurrent_do
./test_concurrent_do
```

If you are interested in learning more about the new C++11 syntax I would recommend reading The C++ Programming Language by Bjarne Stroustrup.




or, Professional C++ by M. Gregoire, N. A. Solter, S. J. Kleper 4th edition:


If you need to brush your Fortran knowledge a good book is Modern Fortran Explained by M. Metcalf, J. Reid and M. Cohen:




Show Comments

## Categories

C# Charts C++11 Regex Scheme Multithreading Posix Books C++ C++14 C++17 OSX Python Objective-C Windows Clang Fortran CUDA Roguelike Perlin Cling C++20 Linux WSL Fractals OpenGL JavaScript OpenCV BeagleBone Productivity Raspberry Pi OpenMP iOS Node.js macOS NumPy SciPy Matplotlib GCC Swift C C99 C11 Arduino Videos Armadillo Chromebook ChromeOS Docker x86-64 F# dotnet C17 Lisp

## Recent posts

- Install OpenCV 4 on Raspberry Pi for C++ and Python development
- Building SBCL - Steel Bank Common Lisp on Windows
- Cross compiling OpenCV 4 for Raspberry Pi Zero
- Python - using C and C++ libraries with ctypes
- Install GNU Octave on macOS and getting started with the image processing package
- Batch convert images to PDF with Python by using Pillow or img2pdf
- C Programming - Reading and writing images with the stb_image libraries
- Building the Windows Terminal application with Visual Studio 2019

- C++17 STL Parallel Algorithms - with GCC 9.1 and Intel TBB on Linux and macOS
- Using Clang as a cross compiler for Raspberry Pi