Compiling C Programs with GCC

Written by Chris Gregg, with modifications by Nick Troccoli

Click here (https://youtu.be/3A6uapoPMfw) for a walkthrough video.

The compiler we will use for CS107 is called the **"GNU Compiler Collection" (gcc) (https://en.wikipedia.org/wiki/GNU_Compiler_Collection)**. It is one of the most widely used compilers, and it is both **Free Software (https://en.wikipedia.org/wiki/Free_software)**, and available on many different computing platforms.

`gcc` performs the compilation step to build a program, and then it calls other programs to *assemble* the program and to *link* the program's component parts into an executable program that you can run. We will learn a bit about each of those steps during CS107, but the nice thing is that `gcc` can produce the entire executable (runnable) program for you with one command.

In CS107, we will predominantly use **Makefiles (make)** to compile, assemble, and link our code, but the Makefile runs `gcc` to do the work. This is just a quick overview on how to compile and run your own programs should you decide to do so without a Makefile.

The simplest way to run `gcc` is to provide `gcc` a list of `.c` files:

```
$ gcc hello.c
$
```

Note that you do not put header files (`.h`) into the `gcc` command: it reads in the header files as it compiles, based on the `#include` statements inside `.c` files.

If the program compiled without errors or warnings, you don't get any output from `gcc` , and you will have a new file in your directory, called **`a.out` (https://en.wikipedia.org/wiki/A.out)**. To run this file, you need to tell the shell to run the file in the current directory, by using `./` before the name:

```
$ ./a.out
Hello, World!
$
```

We generally don't want our programs named `a.out` , so you can give `gcc` an option, `-o programName` , to tell it what to name the runnable file:

```
$ gcc hello.c -o hello
$ ./hello
Hello, World!
$
```

Note: be careful not to accidentally input a runnable file name that is the same as your input file - something like:

```
$ gcc hello.c -o hello.c
```

On `myth` , your profile has been set up to catch the error and not compile it, leaving your source file in tact. This is not the case on many other Linux systems, so be careful! On other systems, GCC would overwrite your source file with the new executable.

`gcc` takes many different command line options (flags) that change its behavior. One of the most common flags is the "optimization level" flag, `-O` (uppercase 'o'). `gcc` has the ability to optimize your code for various situations.

1. **-O or -O1**: Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. With `-O` , the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.
2. **-O2**: Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to `-O` , this option increases both compilation time and the performance of the generated code.
3. **-O3**: Optimize yet more. `-O3` turns on all optimizations specified by `-O2` and also turns on other optimizations. This is often the best option to use.
4. **-O0**: Reduce compilation time and make debugging produce the expected results. This is the default.
5. **-Os**: Optimize for size. `-Os` enables all `-O2` optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size.
6. **-Ofast**: Disregard strict standards compliance. `-Ofast` enables all `-O3` optimizations. It also enables optimizations that are not valid for all standard compliant programs.
7. **-Og**: Optimize debugging experience. `-Og` enables optimizations that do not interfere with debugging. It should be the optimization level of choice for the standard edit-compile-debug cycle, offering a reasonable level of optimization while maintaining fast compilation and a good debugging experience. We will use `-Og` in CS107 when we are debugging.

See the `man` page for `gcc` for more details on optimization (or see **here (https://gcc.gnu.org/onlinedocs/gcc-4.6.2/gcc/Optimize-Options.html)** for more information about optimizations).

Another common flag is the `-std=gnu99` option, which tells `gcc` to use the "gnu c version of the 1999 c standard." The standard provides syntax such as being able to define a variable inside a for loop declaration (e.g., `for (int i = ...)` . We will use this standard in CS107.

We will also use the `-g` flag, which allows us to use the debugger, **gdb (gdb)**, to give us exact line numbers in our code when we run it.

Example:

```
$ gcc -std=gnu99 -g -Og loop.c -o loop
```

If you're interested in even more information about `gcc` , check out Section 1 of this **Stanford Unix Programming Tools (http://cslibrary.stanford.edu/107/UnixProgrammingTools.pdf)** document, as well as the **full gcc manual (http://www.gnu.org/software/gcc/)**(GNU).