Compiling our First Program

Learn C With Babbo

The Direction of This Course

In the community of programmers, one of the most common mechanisms for introducing a language, especially to students of that language, is with a "Hello World" program. The program is extremely simple: it just outputs "Hello World!". In this directory, I've included a C source file entitled "helloworld.c". Now that we are actually going to start compiling and running programs, most lesson directories from now on will have a C source file, which contains a sample program and **comments** describing C programming concepts and what the program does. Comments are segments in source code that are meant to be read by humans and have no impact on what the program does. They usually provide clarifications to other programs and descriptions on how the program should be used. I will frequently litter C source files I provide with comments: these comments should be considered a crucial component of the course, and indeed from now on, less will be explained in PDF files and more will be explained in comments in the lesson source code. Sometimes, no PDF will be provided, and all the information will be contained in the source code comments, and other times, I will provide both. In this case, unless otherwise noted, read the PDF first and then the source code.

Additionally, many lessons will now contain a PDF file called "Exercises.pdf". This PDF will contain conceptual and programming assignments for you to complete. While the information needed to learn C is contained in the comments and PDF files, it is highly recommended that you complete these assignments: I can seldom recall instances in which I effectively learned programming concepts without hands-on experience.

Hello World!

Before you can run the "Hello World" program you'll need to get it onto your computer, either by downloading it or copying its contents into a file editor and saving it. Once you have done so, navigate with your terminal to the directory containing "helloworld.c", and enter

```
gcc helloworld.c -o helloworld
```

on Mac/Linux or

```
gcc helloworld.c -o helloworld.exe
```

on Windows. This **compiles** the program. To run it, enter

```
./helloworld
```

on Mac/Linux or

```
helloworld
```

on Windows. Hopefully, you see the **output** "Hello World!" in your console. As I said, the first command compiled the program: `gcc` is the C compiler command, where `helloworld.c` is the name of the file (source code) we are compiling. `-o` is what is called an **option**. By convention, options are denoted with a single hyphen when they consist of one letter. Here, `o` is short for "output". When we provide the flag `-o`, we are also expected to provide a file name to save the resulting program or **executable** as. The file name we provided was `helloworld` on Mac/Linux and `helloworld.exe` on Windows: this file name is **an argument to the option** `-o`. Note that the suffix `.exe` is not required on Windows, but it is a convention on Windows to have `.exe` as a suffix for executable files. Indeed, the following lines will have the same effect on Windows:

```
gcc helloworld.c -o helloworld
```

```
helloworld
```

Note that on Windows we can omit the suffix `.exe` when running an executable ending in `.exe`, as we did previously. Entering `helloworld.exe` is the same as entering `helloworld` in that case.
Putting all together,

```
gcc helloworld.c -o helloworld
```

means "compile `helloworld.c` and output (`-o`) the program into a file called `helloworld`". Note that the `-o` option, like most command options, is optional (it is actually not uncommon for some commands to require that an option be given). If we omit it, `gcc` will instead output the program to an executable called `a` on Mac/Linux or `a.exe` on Windows, which is obviously not very informative, so you should always specify the `-o` option. Note that `gcc` will replace any existing file in your output path with your program: thus, if we already have a file called `helloworld` and we run

```
gcc helloworld.c -o existing_file
```

and there is already a file called `existing_file` in your directory, it will replace it with your program. Usually, this is a convenient way for us recompile a program after making changes to it, but obviously you shouldn't do anything silly like giving the path to that new book you're writing for the `-o` argument :).

After we compiled our program, we ran it using its name. In Mac/Linux (Bash), we prefixed the program name with

```
./
```

so we ended up entering

```
./helloworld
```

Recall from lesson 0.3 (Terminals) that `.` represents our current directory, so we are basically saying "run the executable `helloworld` in the directory I am in". In contrast, you do not need the `./` prefix if you are in a different directory: for example, you could also execute "helloworld" from the parent directory as follows:

```
cd ..

<name of directory containing helloworld>/helloworld
```

However, if you tried to omit the `./` prefix in the directory containing `helloworld`, you would almost certainly get an error message such as

```
helloworld: command not found
```

This is because Bash, unlike Windows, does not search the current directory for executables when you have only provided a **base name**, or the rightmost component of a path, components being separated by forward slashes on UNIX and back slashes on Windows. On Windows, if there is an executable in the current directory, you may use just its basename as we did to run `helloworld.exe` earlier. Of course, you can run executables from other directories if you want:

```
chdir ..

<name of directory containing helloworld.exe>\helloworld
```

Compile and Run

From now on, I will frequently use the phrase "compile and run" to mean compile, and then run a program. For example, if on a future PDF I say "compile and run example.c", I mean to first compile `example.c` using

```
gcc example.c -o <name of executable>
```

where unless otherwise stated, you are free to name the executable whatever you like. Generally, when I compile a single source file, I simply name the executable with its extension omitted or replaced, for instance "example" on Mac/Linux or "example.exe" on Windows. But again, you are free to choose. After compiling it, you should run the executable using

```
./<name of executable (e.g., example)>
```

on Mac/Linux or just

```
<name of executable (e.g., example.exe)>
```

on Windows.

Now that you have compiled and run a program, answer the questions in "Exercises.pdf" to check your understanding of this material.