Tutorial 2
Hello World!

Learning Objectives

After this tutorial, you will be able to:

- Write and compile a simple C program using gcc
- Understand some compiler warnings
- Practice good code practices
- Submit tar files to Brightspace

It is also worth noting that these tutorials will give you a glimpse at some different ways to work in C than we discuss in class. The different approaches are all valid, and hopefully seeing a few different code styles will help you to learn the underlying concepts better. Anything that you learn in tutorials is perfectly valid in other class work!

Tutorial

For Tutorial 2, you will need your saved files from Tutorial 1. You can re-download the files from the previous tutorial if you do not have them saved; however, do try to save your tutorial files in the future for your own reference and to support tutorials which may carry over.

1. Open a terminal and use `vim` (or any other text editor of your choice, eg. `gedit`) to write a hello world program. These instructions will use vim so that you can get a small introduction to how it works. Try it out! To open a new file in vim, type:

   ```
   vim hello.c
   ```

   This launches vim. Hit the Insert key, or the i key, to enter **insert mode.** Write the following code (exactly):

   ```
   #include <stdio.h>

   int main(void)
   {
       printf("Hello World\n");
       return 0;
   }
   ```

   Note: Placing void as a parameter when a function takes no parameter is totally optional, but good practice as it specifies that this function can take **no** parameters.

   Hit `Esc` to exit insert mode, then type `:wq` and hit enter to save (write) your changes and quit vim. If you want to learn more about working in vim, you can check out this interactive vim tutorial here: https://www.openvim.com/

   Next, compile your "`hello.c`" source code with the gnu C/C++ compiler `gcc`:

   ```
   gcc -Wall -o hello -std=c99 hello.c
   ```

This command instructs gcc to compile the source code file "`hello.c`" and output the resulting program (linked machine code) to the file "hello" (`-o hello`). The compiler will enforce the conventions of the C99 standard for the C language (`-std=c99`) and the compiler will display all the warnings that it finds in compilation (`-Wall`).

Run your "`hello`" program. Remember that in bash, you need to tell the shell that "`hello`" is in the current directory when you want to run it; otherwise, it will default to assuming the program that you want to run is stored where it stores all of the other programs it has access to. We can do this by running `./hello`

At first in this course, you will compile all of your programs on the fly with a command like the one above. Soon, you will be introduced to Makefiles (and the `make` program), which helps to automate the compilation process; however, before automating things, we **must** make sure that we understand what each option in the compilation line above means.

Note: If you do not specify the output file name in gcc (using `-o`, as above), it will by default use the file name "`a.out`".

2. Warnings are hints from the compiler that something *might* not be right. They are not compile time errors and the compiler will still create an executable machine language program if there are no actual errors (although, you *can* tell gcc to treat them like errors and not compile your code into machine code if a warning exists). It is always best to fix your code so that you do not have any warnings when you compile.

   In the "`Warnings`" subdirectory from "`Tutorial1`", which you should have saved from Tutorial 1, there are four small C programs: `t0a.c, t0b.c, t0c.c,` and `t0d.c`. Compile each of these programs and see what warnings the compiler gives. For each warning, go to the line of code that triggered the warning (this information is given in the compiler output) and be sure that you understand why the compiler gave the warnings.

3. Write a program (called `divisor.c`) that has a function `int gcd(int a, int b)`. This function computes and returns the greatest common divisor (GCD) of two integers `a` and `b`.

   Your `main()` function should test your function with different input (giving appropriate output while testing, input values, expected output and actual output).

   Your code must be properly indented and commented. Make sure that you use proper names for your functions and variable names.

Saving and Submission

4. Create a readme file (name it `README.T02`, readme files may have different formats for different assessments) which includes:
   a. A preamble (program author, purpose of the program, list of source files)
   b. Exact compilation command
   c. Launching and operating instructions

5. Create a tar file called "`Tutorial02.tar`" that has your `hello.c, divisor.c`, and `README.T02` files in it. There should be no directories in the tar file.

   Submit your `Tutorial02.tar` file to the Tutorial 2 assignment on Brightspace.
   To receive full marks, you should have completed the submission on Brightspace. Part marks can be provided if you were able to start work and make good progress toward completing divisor.c, at TA discretion.