COMP 2401 -- 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
- submit tar files to *cuLearn*

Tutorial

For Tutorial 2, you will need your saved files from Tutorial 1.

1. Use `vim` (or the text editor of your choice) to write a hello world C program. To do this, begin by typing:

   ```
   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;
   }
   ```

   Hit `Esc` to exit insert mode, then type "`:wq`" and hit enter to save (write) your changes and quit `vim`. For a more resources on `vim`, check the resources section for the course in *cuLearn*.

   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 (machine code) to file 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 it finds when compiling (`-Wall`).

   Run your "`hello`" program. Remember that in bash, you will need to tell the shell that "`hello`" is in the current directory when you want to run it.

At first, you will compile 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, you need to understand what each option in the compilation line above means.

Note: If you do not specify the output file name in `gcc` (using `-o`), 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 intended and commented. Make sure you use proper names for your functions and variable names.

4. Create a `readme` file (“`README.T02`”) that includes:

   a. a preamble (program author, purpose, 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 in *cuLearn*.