ENGG1410: Introductory Programming for Engineers

Lab 0

Setting up the Development Environment for C Programming
Wrting, compiling and running your first program

Mohamad Abou El Nasr
School of Engineering, University of Guelph
Fall 2024

Start Date: Week #1 2024
Duration: 1 Week
Report Due Date: Sunday 15, 2024 - ungraded, just to test DropBox submission
Special thanks to Prof. Shawki Areibi for helping with the first version of the labs

1 Objectives:

The purpose of this lab is to:

- Understand the necessary tools for C development.
- Set up the VS code development environment.
- Introduce you to the GNU C Compiler.
- Acquaint you with the steps of producing an executable.
- introduce you to the different flags supported by the C Compiler.
- Test and validate your tools and the environment by creating, compiling, and running a simple C program.
- Teach you how to resolve simple syntax errors.

2 Tools:

There are two approaches to creating, compiling and running the C programs that you will write for this course: Either you can use a separate tool to perform each of these actions. Use a text editor to create the program file, a compile command to compile it, and another command to run it. Or, you can use an Integrated Development Environment (IDE), which allows you to perform many tasks from a single, usually graphical, application. This IDE can be set locally on your machine or it can be webased on the cloud. The different tools are thus:

- **Visual Studio Code (VS Code):** A free source-code editor made by Microsoft in which all tasks can be done.
- **Online platforms**
- **Command Line Interface (CLI):** using the terminal and writing commands to cary the different tasks

  GCC: The GNU Compiler Collection, a widely used C compiler invoked from the terminal in the command line mode

  Online platforms onlinegdp.com - Replit.com - etc... Are online platforms enabling developpers to write, run and debug C programs to streamline coding process in a cloud environemnt.

In this course, we will mainly use Visual Studio Code (VS Code) as our IDE and get familar with the command line interfade. We will also at times use www.onlinegdp.com. After finishing installing VS Code, your TA will show you these options and you will demonstrate your comprehension of the different tools and your ability to use them.

3 Setting up Visual Studio Code (VS Code)

3.1 Windows Users

If your machine is Windows follow the steps below:

1. Go to VS Code, and download VS Code.
2. Click on the downloaded file to open it. Accept the agreement, and press Next and Install when appropriate. Make sure to tick **Create a Desktop icon**. Then press Finish.
3. Go to MSYS2 and follow the installation steps from 1 to 6.
4. You will need to run step 5 in GCC on Windows, which asks you to run this command:

   ```
   $ pacman -S --needed base-devel mingw-w64-x86_64-toolchain
   ```

   in the MSYS2 terminal (is not part of the command). Press Enter to accept the default option.
5. Check GCC is installed by running the following command in the MSYS2 terminal.

   ```
   $ gcc --version
   ```

6. Go to **Settings** using the search at the bottom left. In **Find a setting**, search for "Edit environment variables for your account". Click on **Path**, and click **Edit**. In the new window, press **New** on the right and add C:\msys64\mingw64\bin and click **OK**.
7. Launch Visual Studio Code

3.2 Mac Users

If your machine is a MacOS follow the steps below:

1. Go to VS Code, and download VS Code.
2. Once the download is complete, open the zip file downloaded by double-clicking the .zip file. It will extract the zip file and install the application. Drag and drop the produced Visual Studio Code Application into the Applications folder.
3. Before being able to compile, run and debug your program, we need to install the necessary C compiler and debugger. To do so, click on **Spotlight Search**, type and select **Terminal**.
4. Type in the following command in the terminal:

   ```
   $ xcode-select --install
   ```

   Click "Agree" if you are prompted to agree on a license. Installation will take time, depending on your machine's capacity and Internet connection.be patient.
5. Check if your C compiler has been correctly, so you will see the version number of gcc installed by typing:

   ```
   $ gcc --version
   ```

6. Launch Visual Studio Code

4 From Source to Executables

Until now, you haven't really had to think about how programs are edited, compiled and run. In the comming sections, we'll dive into the details of how you compile C programs, and how you automate compilation in larger projects that consist of multiple source files.

4.1 What is Compiling

The compilation process involves four stages and utilizes different "tools" such as a preprocessor, compiler, assembler, and linker in order to create a final executable.

1. **Preprocessing**: is the first pass of any C compilation. It removes comments, expands include-files and macros, and processes conditional compilation instructions.
2. **Compilation**: is the second pass. It takes the output of the preprocessor, and the source code, and generates assember source code demo.s. Assembly language is a low-level programming language that is still human-readable but consists of mnemonic instructions that have strong correspondence to machine instructions.
3. **Assembly**" is the third stage of compilation. It takes the assembly source code and produces an object file (demo.o) which contains actual machine instructions and symbols (e.g. function names) that are no longer human-readable since they are in binary format.
4. **Linking**: is the final stage of compilation. It takes one or more object files and/or libraries as input and combines them to produce a single (usually executable) file. In doing so, it resolves references to external symbols, assigns final addresses to functions and variables, and revises code and data to reflect new addresses in memory (a process called relocation).

5 Using VS Code

1. To create a new folder, click on **Explorer** → **Open Folder**. Navigate to your desired folder and press **New Folder**. Name your folder **ENGG1410-Labs** and click **Create** → **Open**. Make sure your desired folder that contains ENGG1410-Labs does not contain any spaces in its path.
2. Create a new folder in the ENGG1410-Labs folder, for example lab0, by clicking on **New Folder** icon next to ENGG1410-Labs on the top left.
3. Click on lab0, and press the **New File** icon next to ENGG1410-Labs to create a new file.
4. To compile using VS Code, we need to install two necessary extensions. Go to Extensions below the Explorer icon.
5. Install **C/C++** and **C/C++ Runner** extensions by clicking on **Install'** after you click on each of them. **Note:** For Windows, please additionally install **C/C++ Extension Pack** and **CodeLLDB**. Close the tabs when installations are complete
6. On the bottom left of your window, you should see **Select Folder**. This helps you select a particular working folder. In our case, we want to select *ENGG1410-Labs/lab0*.
7. Write a test code in *lab0.c* file. Turn on the option of **Autosave**, so that you do not need to worry about saving your C program every time you make a change. Go to **File** → **Autosave**.
8. To compile your program, press the *Compile* button towards the bottom left of the screen. To run your program, press the *Run* button. Any output will be observed in the terminal to the lower half of your VS Code window.
9. Consistent code is easy to read. Consistency appears in the proper **indentation, spacing and appropriate line length**. To automatically ensure your code is consistent, you can use the default formatter in VS Code. To do so, go to **Settings** by clicking on the gear at the bottom left of VS Code window. In the search bar, type "formatter". Check the **Editor: Default Formatter** is set to **None**, and **Editor: Format on Save** is ticked. This will automatically format your code every time you save. In addition, in the search bar, type "format" and under **Extensions C/C++**, set the **C_{Cpp}:_{Clang}_{format}_{fallback} Style** to **Google**.
   This will ensure that your code is formatted according to the Google C++ Style Guide.
10. Walking through individual lines of your code step by step is called **debugging**. To debug your code, you need to add a **breakpoint**. A breakpoint is a line of code where your program will pause at when you click the *Debug* button at the bottom left next to the Compile and Run buttons. To create a breakpoint, you need to hover over the line you want your code to pause at and click there.
11. Click on the **Debug** button to start a debugging session. Your code will stop at your breakpoint without executing it, unless you press **Step over** button. As you step over, you will execute further lines and observe variable values changing on the left. If you want to continue running the program, without debugging further, you can press the **Continue** button. You don't need to worry about debugging at the moment, later labs will cover this aspect in details.
12. You can use the tempelate provided under labs in courselink to have a simple customized Launching and settings for your VS code program.

6 From Source to Executable Using Command Line

In this section we will briefly show you how to successfully edit and compile a c program. For a more detailed description please check the links below that take you step by step to edit your source file, compile it and create the executable.

To create an executable using the C compiler, you will need to follow these steps from within a terminal window:

1. Make sure you have the gcc compiler installed by issuing one of the following commands:
   - `whereis gcc` .. a possible output would be → gcc: /usr/bin/gcc /usr/lib/gcc
   - `which gcc` .. the system will indicate the directory it is installed .. → /usr/bin/gcc
   - `gcc --version` .. a possible output would be .. → gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-44)
2. Create a file called demo.c using any text editor of your choice. Examples include Atom, Notepad++, vi or emacs or vim (VS code will work too!)

```
#include<stdio.h>
/* demo.c:  My first C program */
int main(void)
{
   printf("Hello! This is a test prgoram.\n");
  return 0;
}
```

3. Use any one of the following syntax to compile the program called demo.c
   Notice that the -o flag allows you to place the output of any gcc program into the file specified by the argument immediately following the flag.
   - gcc program-source-code.c -o executable-file-name (i.e `gcc demo.c -o demo.exe`)
   - cc program-source-code.c -o executable-file-name (i.e `cc demo.c -o demo`)
4. If you wrote multiple files for your applications (i.e. demo1.c and demo2.c) then you can compile them into an executable called demo.exe by doing the following:
   `gcc demo1.c demo2.c -o demo.exe`
5. If there is no error in your code or C program then the compiler will successfully create an executable file called demo or demo.exe in the current directory. You can verify that the executable exists by issuing the command

```
$ ls -l demo*
```

6. Next, you will execute the program called demo or demo.exe by following the steps below:
   - `./demo.exe` (this tells the OS to use the current directory)
   - `/path/demo.exe` (in other words state the entire path where demo.exe exists)

6.1 Flags

The gcc compiler supports the use of hundreds of different flags, which we can use to customize the compilation process. Flags, typically prefixed by a dash or two (-<flag> or --<flag>), can help us in many ways from warning us about programming errors to optimizing our code so that it runs faster.

The general structure for compiling a program with flags is:

```
$ gcc <flags> <c-file> -o <executable-file-name>
```

1. Warning Flags:
   (a) **-Wall**: is one of the most common flags. It will cause the compiler to warn you about technically legal but potentially problematic syntax including (a) Uniinitialized and unused variables, (b) incorrect return types, (c) invalid type comparisons.
   Usage: `gcc -Wall demo.c -o demo.exe`
   (b) **-Werror**: this flag forces the compiler to treat all compiler warnings as errors, meaning that your code won't be compiled until your fix the errors. This may seem annoying at first, but in the long run, it can save you lots of time by forcing you to take a look at potentially problematic code.
   Usage: `gcc -Wall -Werror demo.c -o demo.exe`
   (c) **-Wextra**: this flag adds a few more warnings not covered by -Wall. Some problems that -Wextra will warn about include: (a) empty if/else statements, (b) unused function parameters.
   Usage: `gcc -Wall -Werror -Wextra demo.c -o demo.exe`
2. Debugging Flags:
   (a) **-g**: this flag requests the compiler to generate and embedd debugging information in the executable, especially the source code. This provides more specific debugging information when you're running your executable with gdb or address sanitizers.
   Usage: `gcc -g demo.c -o demo.exe`
3. Optimization Flags:
   (a) **-O0**: this will compile your code without optimization – it is equivalent to not specifying the -O option at all. Because higher optimization levels will often remove and modify portions of your original code, it's best to use this flag when you're debugging with gdb.
   Usage: `gcc -O0 demo.c -o demo.exe`
   (b) **-O3**: this will enable the most aggressive optimization, making your code run fastest.
4. Math Functions Enabler Flags: In order to compile your C program to use math functions you should pass the -lm option with gcc to link with the math libraries.
   Usage: `gcc demo.c -o demo.exe -lm`

7 Useful Resources

Here are other useful resources:

- Visual Studio Code. Check the tutorials specific to your system (Windows or MAC): C/C++ for Visual Studio Code
- The online IDE with code editor, compiler, and debugger GDB Online
- Getting started with replit IDE replit.com
- Atom text editor Atom
- A useful tutorial to create your source code using the "VI" editor: VI Editor with Commands
- A useful tutorial to compile and link a C program: Compiling, Linking, and Building C Applications

8 Requirements

Design, enter, compile and **execute** the program that implements the following functionality: Print a welcome message on the screen. This will be your lab0.c program.

9 Deliverables

1. **DEMO**:
   - You will need to demonstrate to the Teaching Assistant that your program fulfills the requirements.
   - You will need to demonstrate to the TA that you can write, edit , save, compile and run your program.
   - The DEMO will take place during the LAB hours and not outside these hours.
2. **Report and source file**
   - Just to experiment with DropBox which will be used for future graded lab reports, you are required to submit (upload to dropbox) a simple report that contain your group names and your feedback about the first lab as well as your lab0.c program you crearted.