ENGG1003 - Lab 1

Brenton Schulz

1 Introduction

This laboratory exposes you to the fundamental tools required to write computer programs in C. No prior programming experience is assumed

2 C Programming Basics

In order to write programs in C (and most other languages) the following software tools are required:

- An *editor*, to create and edit raw text files.
- A *compiler*, to convert your text files into an *executable* file.

A programming editor is very different to a *word processor* (eg: Microsoft Word) in that it displays and stores raw ASCII text only. What you see printed to the screen represents the *actual data* stored in the file. By contrast, Word will store a combination of text and display formatting and, as such, is not suitable for writing code.

Programming editors will generally have features optimised for coding, such as:

- Syntax highlighting
- Line numbering
- Auto completion
- Pre-emptive error notifications
- Communication with the compiler to highlight errors
- Automatic indenting
- Highlighting of matching blocks
  - ie: an easy method to find matching pairs of ( ), { }, ” ”, etc.

It is hoped that you will discover these features and learn to work with them. In time you will learn which features work well with your style and which simply get in the way.

For the time being the “compiler” noun will be used to colloquially reference a highly complex set of software tools which turn your source code into an executable binary file. You will be shielded from the details until otherwise necessary.

2.1 Introduction to OnlineGDB

OnlineGDB is a basic (*very* basic) browser-based development environment for a variety of programming languages. It gives you access to an editor, a small amount of cloud storage, compiler, and standard input / output. It also contains a *debugging* feature however for technical reasons[1] we won't be using it.

All compilation and execution is performed on the OnlineGDB server. As such, the service has an incredibly low barrier to entry: there is (almost) zero installation/configuration required to get started running code.

Task: Open a web browser and navigate to `http://www.onlinegdb.com`.

NB: If a demonstrator sees you using Edge or IE they may instinctively think you need more help than students using Chrome or Firefox.

[1] It only allows one debug session per IP address. The entire campus uses the same public IP so we can't use it in labs.

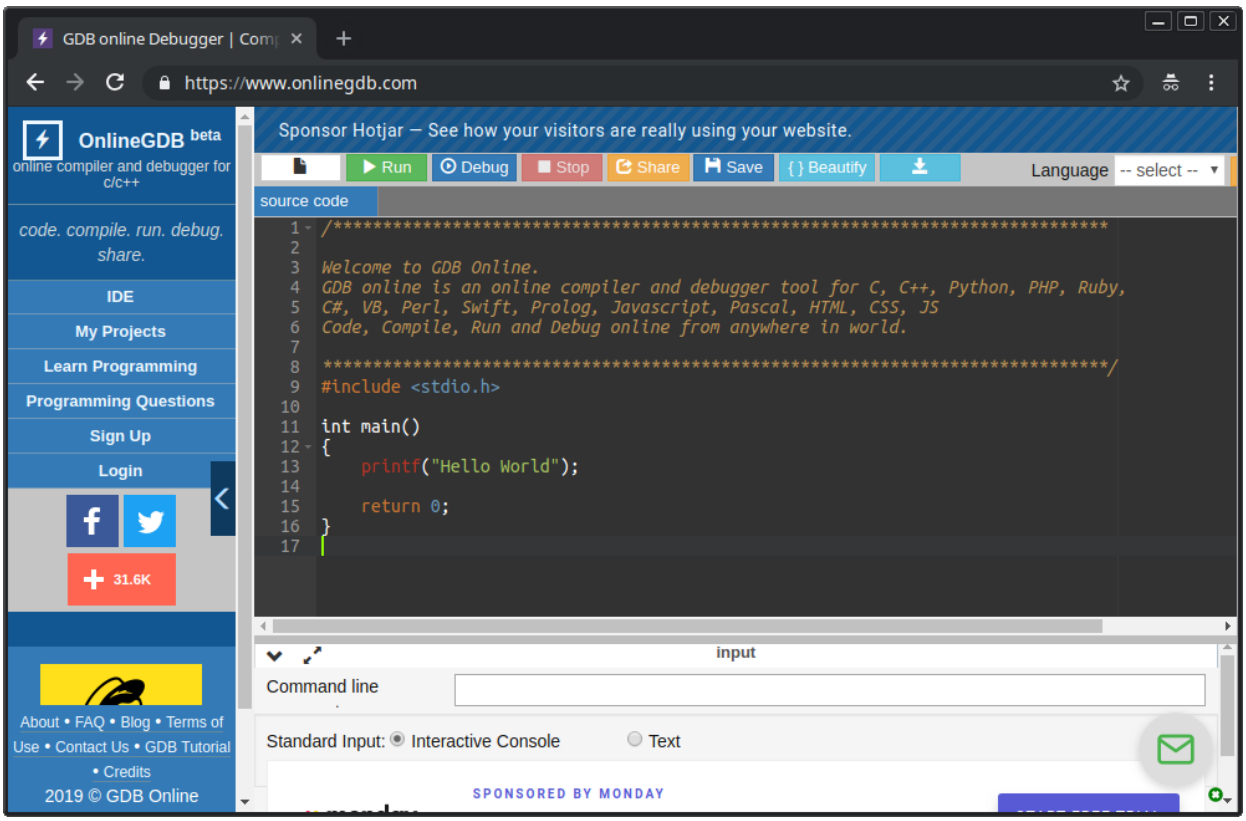

Figure 1: The view when OnlineGDB is first opened.

After OnlineGDB has loaded you will be greeted with the screen seen in Figure 1. The large area in the middle is the editor screen, this is where you will type C code. Immediately you can observe that this editor supports line numbering and syntax highlighting.

Above the editor is a toolbar which, from left to right, performs the following functions:

1. Create a blank new file
2. Run the project
3. Debug the project (not used in ENGG1003)
4. Stop execution of a running program
5. Share - Generates a link to your current source code
6. Save - When logged in this saves the project files to your personal cloud storage
7. { } Beautify - Modifies your code's whitespace to adhere to the OnlineGDB indenting style (NB: I tried this at time of writing and it didn't work on my personal computer. Go figure.)
8. Download - Downloads the currently viewed file.

The area below the editor is where standard output is written to and standard input read from. When the code is run its appearance changes to that of a basic console (ie: the GUI elements disappear and it becomes just text).

Task: Configure OnlineGDB to run C code by selecting "C" from the "Language" drop-down box in the upper-right. This website supports many languages[2] so feel free to come back here later if you're interested in learning any of the others. Python, although not taught in an Engineering degree, is a common choice for Engineering PhD students as a free MATLAB alternative and is probably worth playing around with.

[2]MATLAB is not one of them because it is a *very* expensive commercial package

Task: Click the green Run button. The box at the bottom of the screen will produce a "Compiling" animation and, after execution of the template code, will produce the output seen in Figure 2.

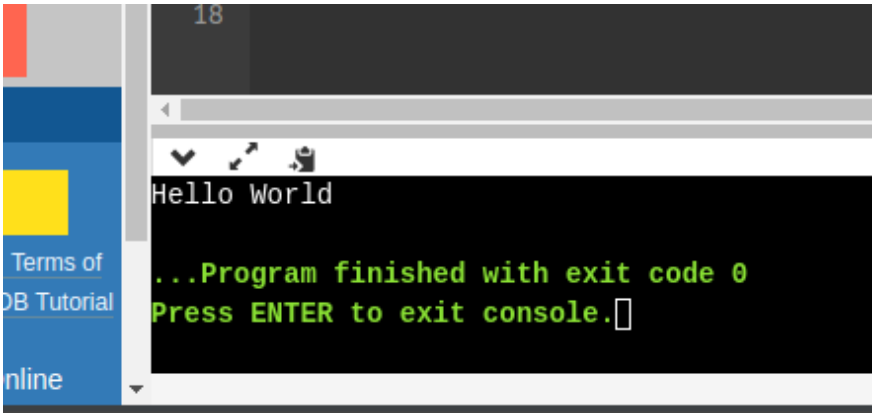

Figure 2: A cropped screenshot showing the "Hello World" program output.

Off-topic note: Remember the "returns zero to the operating system" comment in lecture 1? Well that's what the text "...Program finished with exit code 0" is referencing. The 0 is the number that `main()` *returned*. We will learn about *function return values* later in the semester.

Tasks:

1. Running the default template demonstrates *standard output*, modify the code to match that in Listing 1. While making the changes you will observe OnlineGDB's auto-complete features. When, for example, you type a double quote " character it *automatically* types two and places the cursor between them. It will also provide auto-complete suggestions, although many of them will be inappropriate (it is, after all, just a computer program; not a science fiction grade artificial intelligence).

   What other "helpful" editor behaviour did you notice? Some of it will be annoying at first (some of it will be annoying *forever*) but learning to work with the editor's features will improve your coding speed in the long term.

```c
#include <stdio.h>

int main() {
   int k;
   scanf("%d", &k);
   printf("You entered: %d\n", k);
   return 0;
}
```

Listing 1: A basic C program which demonstrates input and output.

2. After editing the code press Run. After it is compiled you will notice that the console is just displaying a cursor. This is because `scanf()` waits for data to be typed (specifically, it will wait until a new line character, ASCII value 10, is sent).
3. Type an integer and press `enter / return`. There will be some "lag" because the data is being sent to OnlineGDB's server before being displayed.
4. After pressing enter the console should display the text "`You typed: 123`".
5. Run the program again except this time don't type just an integer, try typing a word, or a word containing a number, or a number followed by letters (with and without a space). What is the behaviour each time? Are you getting annoyed by the slow compile time and lag yet? OnlineGDB may be simple but it is, at times, a compromise.

3 Git

3.1 What on Earth is git?

4 Getting started with C in Ubuntu Linux

Linux is not supported by university IT but, personally, I find it to be a fantastic development platform. The following instructions are completely optional, only do this if you are keen to learn.

Installing Ubuntu is beyond the scope of this document (and course). If you think this is a daunting task I would recommend only using the officially supported tools to complete ENGG1003. There are many thousands of websites and YouTube videos which will guide you through the Ubuntu (or Mint, Arch, etc.) installation process. **NB:** Installing a new operating system can *very easily* destroy all existing software on a machine. Don't do this if you don't know what you're doing.

That out of the way, here's how you can get started:

1. The C compiler in Linux (and OnlineGDB, and *many* other platforms) is gcc (the GNU C Compiler). To install it, open a terminal (`ctrl + alt + t`) and type:

   `$ sudo apt install gcc libc6-dev gedit`
   (the $ indicates the terminal prompt, don't type that character)

   When prompted, enter your password, wait a few seconds, press `enter` if it wants installation confirmation, and wait a few more seconds. An Internet connection is required for apt to download the required software.

   The `libc6-dev` package provides all the basic C libraries (`printf` etc.) and `gedit` is a basic text editor.

   To compile a basic C program:

2. Open a terminal (`ctrl + alt + t`). It will start to your "home directory" (ie: any commands typed will effect files in your home directory).

3. Lets make a new directory for writing C files, type:

   (a) `$ mkdir c`

   (b) `$ cd c`

   The first command creates a directory called "c" and the second changes into that directory.

4. Create a new .c file. We will do this in `gedit` (because it is easy and simple) but there are many others (the more nerdy among you may want to learn `vim` or `emacs`. They are *very* powerful editors).

   Type: `$ nohup gedit test.c &`
   (The & symbol at the end of a command runs the command "in the background". This gives you the command line back straight away, instead of having to quit gedit first. Preceding the command with `nohup` stops `gedit` from closing if you close the terminal window)

   then type out the code seen in Figure 3.

5. Click the Save button (or type `ctrl + s`).

Figure 3: The gedit window with some C code typed out.

6. Move the *keyboard focus* back to the terminal (ie: click the terminal window).
7. If you can see the command prompt yet, press `enter`
8. You can type `ls` to see a list of all files in the current directory. `test.c` should be there.
9. To compile the `.c` file run: `$ gcc test.c -o test`

   This will create a binary executable called `test`. If the `-o` *command line argument* is not given `gcc` the binary file defaults to the name `a.out`.
10. Run `test` by typing: `$ ./hello`

    The `./` is a special character string meaning "relative to the current directory". If you try to run `test` from any other directory nothing will happen because `test` is a built-in command. With most other names you will get a "Command not found..." error.
11. When the program runs you should see something similar to Figure 4.
12. Go back to `gedit` and keep coding as you desire. Return to the command line to run `gcc` to re-compile your code. **NB:** The command line has a *history* feature, pressing the up arrow will scroll through past commands, *you don't need to type them out from scratch.*
13. If you enjoyed this you're a bit weird, welcome to the club! Recommended further reading would be a tutorial on `make` followed by investigations into more powerful editors like `vim`.

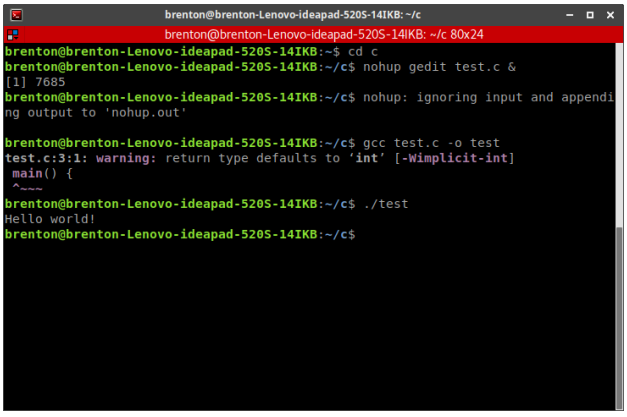

Figure 4: The complete command line sequence performed in these steps.