CSCE 155 - C

Lab 1.0 - Introduction
Online Version

Dr. Chris Bourke

Prior to Lab

In each lab there may be pre-lab activities that you are *required* to complete prior to attending lab. Failure to do so may mean that you are not prepared to complete the lab.

1. For this lab, you'll need to sign up with GitHub, a website that hosts *software repositories* using git (a distributed version control system). With GitHub you'll be able to store, manage and backup your source code. Git is an essential software development tool that you'll want to fully utilize.

   Go to `https://github.com` and sign up using your `@huskers.unl.edu` email address so that GitHub will recognize you as a student. This makes you eligible for a lot of free software and other access. After you have signed up, you can get the GitHub Student Developer Pack at `https://education.github.com/pack` (but it is not necessary to complete this lab or for the course).

2. Claim your CSE account. Your CSE account allows you to login to the cse server and will be used to hand stuff in and to grade yourself using the webgrader. To claim your account go to `https://cse-apps.unl.edu/amu/claim` and follow the instructions provided.

Some other department and university computing resources and policies can be found at the following:

- CSE Website: `http://cse.unl.edu`
- UNL Computing Policy: `http://www.unl.edu/ucomm/compuse/`
- CSE Academic Integrity Policy: `http://cse.unl.edu/academic-integrity-policy`

- CSE System Frequently Asked Questions (FAQ): `http://cse.unl.edu/faq`
- Account Management: `https://cse-apps.unl.edu/amu/`
- CSE Undergraduate Advising Page: `http://cse.unl.edu/advising`
- CSE Student Resource Center: `http://cse.unl.edu/src`

Peer Programming Pair-Up

For students in the online section: you may complete the lab on your own if you wish or you may team up with a partner of your choosing, or, you may consult with a lab instructor to get teamed up online (via Zoom).

For students in the face-to-face section: your lab instructor will team you up with a partner.

To encourage collaboration and a team environment, labs are be structured in a *peer programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

1 Lab Objectives & Topics

At the end of this lab you should be familiar with the following

- The CS50 IDE that you'll use for this course
- Basic unix commands
- Retrieving lab code from Github

- Modifying, compiling and executing your first C program
- Using CSE's web handin and web grader

2 Activities

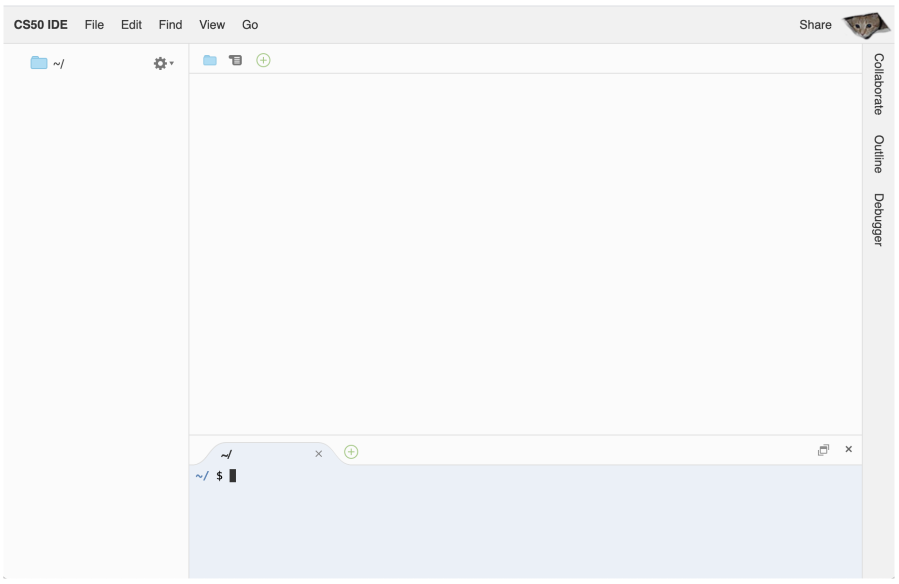
2.1 Getting Started

Let's get started. In order to develop programs in C you'll need a code editor (plain text editor), a compiler (to compile the C code into executable machine code) and a runtime environment (to actually execute the program). To do this, we'll run through how to use the CS50 Online IDE (Integrated Development Environment) developed by Harvard. This is a web browser-based IDE that means you don't have to install any software. It is free and you login using your GitHub account. It also offers a *persistent* environment: files you save or *persist* in the IDE and be available next time you login.

Point your browser to the CS50 website and Sign in with GitHub:

https://ide.cs50.io/

Once logged in, it should looks something like the following.



```
~/ $ 
```

Figure 1: CS50 IDE

First observe the major layout areas as depicted in Figure 2.

1. File Navigator - This is a graphical file explorer in which you can right click and create new directories (folders) and files, drag and drop to move things around, etc.

2. File Editor - Double click on a plain text file and it will open in this file editor so you can edit it. This is a tabbed environment so multiple files can be opened at once.

3. Console - this is a Text User Interface (TUI) console in which you can type and execute commands to compile and run (text-based) programs in a Linux environment.

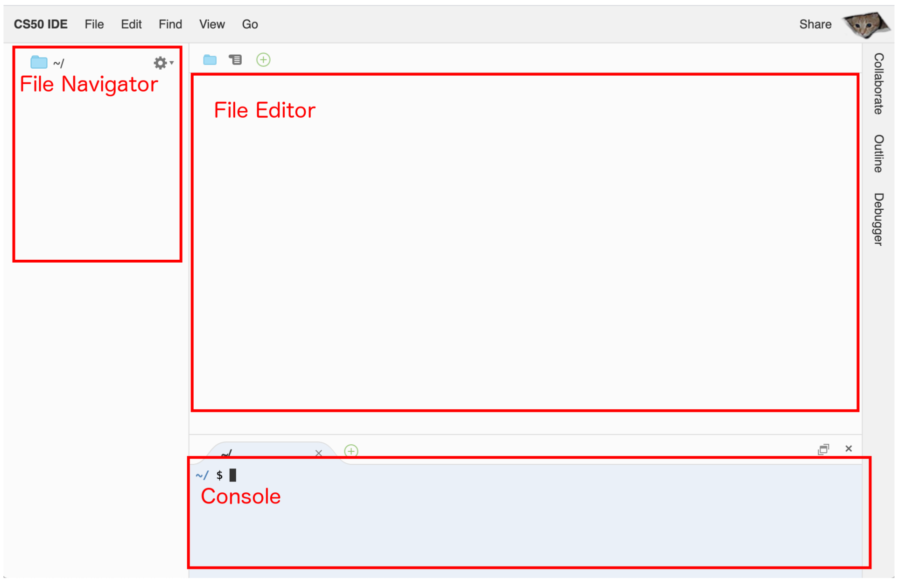


Figure 2: CS50 IDE with layout sections highlighted

Let's explore the basics:

1. Create folders (right click the *root* folder) named `labs` (in which we'll keep all our lab code, `hacks` (in which we'll do the same for hacks) and `misc`

2. Create a file named `info.md` in the `misc` directory (again, right click the folder). Double click the file and edit it (type your name and email). Be sure to save.

3. Now we'll get familiar with the console and some basic unix commands. Click in the console area. When we say type a command, we mean type the command and then hit enter to execute the command.

   a) Type the command `ls` (short for "list") and hit enter. This lists the files and directories in the *current working directory*. You should see your three directories.

   b) Let's change our current working directory so that we're *in* the `misc` directory. Type the following command `cd misc` (short for "change directory"). Type `ls` again and you'll see your `info.md` file.

   c) You can determine *where* in the directory structure you are by typing the command `pwd`. Try it; you'll see that your `misc` directory is actually under the `/home/ubuntu/` directory

d) You can go back "up" the directory structure by typing the command `cd ..` (`..` is short hand for the *parent* directory or one directory up in the hierarchy)

e) You can also remove a file using the `rm fileName` command which will *permanently delete* the file. Careful, there is no undo or recycle bin! You can also remove a file by right clicking and deleting it in the File Navigator.

Before we go on, let's setup our environment for the rest of the semester. Type the following command (you may want to copy and paste) to install several libraries:

```
sudo apt-get install libcmocka-dev libcmocka0 cmocka-doc unixodbc
odbcinst gtk+-3.0 libcurl4-gnutls-dev -y
```

A more comprehensive tutorial on unix commands is available here: http://www.math.utah.edu/lab/unix/unix-commands.html.

2.2 Editing Code

Programming requires that you write code in a *source file*, a plain text file that contains syntactically valid programming commands. In general, you can use any plain text editor to edit code (MS Word is not a plain text editor). However, it is much better to use a text editor designed for code that uses *syntax highlighting* to help you differentiate various programming elements by highlighting them in different colors and fonts. The CS50 IDE provides such an editor. Let's practice writing some code.

1. In your `misc` directory create a new file named `hello.c`
2. Open the file and edit its contents to look like the following but with your name and the current date.

```c
/**
 * Author: Your Name
 * Date: 20xx/xx/xx
 *
 * A simple hello world program in C
 *
 */
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv) {

  printf("Hello World!\n");

  return 0;
}
```

3. Save the file
4. Now let's *compile* this code into an executable program. In the console, type the following command (be sure you're still in the `misc` directory):

   `gcc hello.c`

   `gcc` (GNU Compiler Collection) is a C compiler. In general in unix/linux if a command is successful nothing will be displayed; "no news is good news." If you made a mistake the compiler will give you a (sometimes) helpful hint about what was wrong. If you made a mistake fix it and repeat until it successfully compiles.
5. When successful the compiler produces an *executable* file named `a.out`. Use `ls` to verify that the new file has been created (or observe it in the File Navigator).
6. Run your program by typing the following command: `./a.out` which should print `Hello World!` to the Console.

Congratulations on your first program!

2.3 Checking Out Code From Github

Each lab will have some starter code and other *artifacts* (data files, scripts, etc.) that will be provided to get you started. The code is hosted on GitHub (`https://github.com`) and you must *clone* a copy of it to your own workspace.

1. Change your current working directory to your `labs` directory. To do this you can use `cd ..` then `cd labs` (assuming you are currently in `misc`) or you can do this in one step:

   `cd ../labs`

   which means "go up one directory, then down to `labs`."
2. To clone, use the the following command:

   `git clone https://github.com/cbourke/CSCE155-C-Lab01`
3. A new directory, `CSCE155-C-Lab01` should be created, go into this directory by typing `cd CSCE155-C-Lab01`
4. List the contents of this directory by typing `ls`. If everything worked, there should be a `hello.c` file in your directory similar to the one you just wrote.

Boring. Been there done that. So, let's modify the program.

1. Open the `hello.c` source file (you now have two files with that name, so be sure you're editing the correct one or close the other).
2. Modify the file by changing the author to you (and your partner if you have one) and change the date.
3. Change the message that is produced by the program (the `printf` statement) to instead print you (and your partner's) name.
4. Save, compile and run your program to make sure it works.

2.4 Submitting Your Program

Many of your assignments will include a programming portion that will require you to hand in *soft copy* source files for graders to compile and evaluate. To do this, you will use a the CSE webhandin.

1. First, we need to get your source file to *your computer*. Right now, the files only exist on the CS50 IDE server. Right click on the `hello.c` file and click download. You can also download a copy of your entire project using File → Download Project (in fact, it is a good idea to periodically backup your project by downloading it, but using git is a much better solution).
2. Point a new browser tab to `https://cse-apps.unl.edu/handin`
3. Login with your CSE credentials
4. Click on this course and lab 01. You can either click the large "handin" area and select your downloaded `hello.c` file or you can drag-drop the file. You can (re)submit the same file as many times as you want up until the due date.

Some things to understand about webhandin:

- File names are case sensitive and you may only submit files with names as specified by the particular assignment
- There is no need to delete the file if you need to resubmit it, the old version will be overwritten
- If you make no changes to the file the resubmission will be rejected
- The most common mistake is handing in the *wrong version* of a file, so be aware of which file(s) you're handing in

2.5 Grading Your Program

Now that the file has been handed in, you can "grade" yourself by using the webgrader.

1. Open a new tab/window and point your browser to one of the following depending on which course you are in:
   - `https://cse.unl.edu/~cse155e/grade`
   - `https://cse.unl.edu/~cse155h/grade`
2. Fill the form with your CSE login and password, select the appropriate assignment (Lab 01), click "Grade" and observe the results.

Some things to understand about webgrader:

- For future assignments and labs, you can compare the results of your program with the "Expected Results". In general, the output does not need to match *exactly* as long as you report *at least* as much information as the expected output, you're probably good.
- If there are problems or errors with your program(s), you should fix/debug them and repeat the handin/grading process. You can do this as many times as you like up until the due date.
- Some programs and assignments will run test cases and may provide expected output alongside your output. Others may have more sophisticated test cases and actually provide you a percentage of test cases passed. It is your responsibility to read, understand and *address* all of the errors and/or warnings that the grader produces.
- The webgrader is a *black box* tester meaning you don't have access to its internal workings. You should properly and thoroughly test and debug your programs locally instead of relying on webgrader as a "blind tester"