CSO: Recitation 1

Goktug Saatcioglu

09.05.2019

1 Installing the correct libraries

1.1 OS X

1. Open a new terminal window.
2. Run the command `xcode-select --install`. This will install a C and C++ compiler (gcc and clang) along with other tools you will need for this course (such as git).
3. Install homebrew. Brew is a package manager for OS X.
4. In your terminal window execute the following commands: `brew install gdb` (to get the gdb debugger) and `brew install git` (to install a newer version of git).
5. If you don't have a GitHub account still create one. Follow Prof. Wies' instructions on this. Run the following commands

```
git config --global user.name "Your Name"
git config --global user.email "you@your-domain.com"
```

   to set your git user name and email. If you'd like colorful terminal UI run this command

```
git config --global color.ui auto
```

   (this is in general very useful).

1.2 Ubuntu/Debain (apt-get)

1. Open a new terminal window.
2. Run `sudo apt update` to grab updates.
3. Run the following command `sudo apt install build-essential` to install gcc and other build tools.
4. Run `sudo apt-get install gdb` to install gdb.
5. Run `sudo apt install git` to install git.
6. Run the git config commands.

```
git config --global user.name "Your Name"
git config --global user.email "youremail@domain.com"
```

1.3 yum

If your Linux OS uses yum contact me and I'll figure out the commands you need to install everything.

1.4 Windows 10

1. Install Ubuntu via the Windows Subsystem for Linux. Use e.g. the instructions in the following two tutorials to do this: 1 and 2.

2. If you notice blurry fonts when you open applications from within Ubuntu WSL that use the Xming X11 server (e.g. Visual Studio Code, Emacs, etc.), then do the following steps:

   (a) Open the directory `C:\Program Files (x86)\Xming`

   (b) Right-click the file Xming.exe and choose "Properties" from the context menu

   (c) Select the tab "Compatibility" and then click "Change high DPI settings"

   (d) Then under "High DPI scaling override" click the checkbox "Override high DPI scaling behavior" and select "Application" in the drop down menu labeled "Scaling performed by:".

   (e) Then close both pop-up windows by clicking "OK"

   After this, the fonts should be crisp the next time you start the X11 server. If you already have it running in the background, you'll have to manually kill the process before you restart it. Rebooting will also do the trick. Of course, if you do not have X11 also install that.

3. Run the commands given above for Ubuntu.

4. Install CLion (instructions given below) from inside the Ubuntu subsystem rather than as a Window app. Basically, use the Ubuntu subsytem for everything from this point on.

2 CLion

The recommended IDE for this class is CLion. In the next section we will also see how to compile and run C programs through the terminal so that you may use an IDE of your choice.

2.1 Download and Install

1. Go to JetBrains and first sign up for a JetBrains account. You should use your NYU email address so that you can get a student license to use the software.

2. Download the CLion installer from here. You can find the Try CLion section and then click Get Now. When prompted enter your newly created account's credentials.

3. Open the installer and follow the instructions for your system.

2.2 Setting Up

When first running CLion a setup process will run. Follow the instructions here.

1. Select Do not import settings and continue.

2. Choose a theme and continue.

3. Make sure you are using the bundled version of CMake. CLion should detect your Make, C and C++ compilers automatically. If not you will have to find them. Try running the `which` command in your terminal. For example, `which make` will show where your `make` is located. If nothing shows up then you probably haven't installed it correctly. For your debugger pick Bundled gdb or your system installed gdb. However, if you are on OS X you must pick Bundled LLDB due to a technical problem in OS X. Continue.

4. Keep the default plugins and continue.

5. Install Markdown support. If you are familiar with Vim you can also install IdeaVim.

6. Start using CLion!

2.3 Creating a Project

Now we will create a small C project to demonstrate how CLion works. After going through the CLion setup process the default CLion startup screen should show.

1. Click create a project.
2. Click on C Executable. We will use the C99 standard in this class. You can also change the name of the directory your project will be located in. Once you are done click create.
3. You should see a new screen with two file tabs. One is `CMakeLists.txt` which describes how `make` should build your C project. For now you can ignore this and most of the time CLion should be able to take care of this file for you. Secondly, there should be a `main.c` file. Open that one (if it already isn't).
4. Let's run this program. Hit the green button on the top right. You should see the string `"Hello world"` displayed at the bottom of the screen.

2.4 A Little Exercise

Let's try writing a small program as an exercise. We will create a new file named `fib.c` that computes the n-th Fibonnaci number recursively. You can try writing this program yourself sometime but we will use this solution for this exercise.

```c
#include "fib.h"

int fib_rec(int n) {
    if (n == 0 || n == 1) {
      return n;
    }
    return fib_rec(n-1) * fib_rec(n-2);
}
```

Listing 1: fib.c

```c
#ifndef HELLOWORLD_FIB_H
#define HELLOWORLD_FIB_H

int fib_rec(int);

#endif //HELLOWORLD_FIB_H
```

Listing 2: fib.h

We see two files here. The first one is `fib.c` which contains our implementations of recursive Fibonnaci calculation. The function `fib_rec` is fairly intuitive. The second file is `fib.h` which is referred to as the header file for `fib.c`. For now all you need to know is that a header file allows you to declare function signatures where the functions are implemented in your source file (`.c`) so that other C files can use your functions. Think of it as importing in Java but instead of importing your source code you import the declarations that you wish another file to be able to use. The compiler takes care of bringing everything together.

Let's create this file. Follow the steps below to create a new file and then run it.

1. Right-click on the root folder of your project in the file view screen. Select New and then C/C++ Source File.

2. Name your file `fib` and make its type `c`. Furthermore, make sure to tick Create an associated header and Add to targets. Add to targets will let CLion take care of configuring the `CMakeLists.txt` correctly for your new file. Create an associated header will create a header file `fib.h` to go with your new C file `fib.c`. Copy and paste the code above to the relevant files.

3. Go back to `main.c`. At the top of the file add the line `#include "fib.h"`. This tells the compiler that you wish to import the header file `fib.h` which is (for our purposes) equivalent to importing `fib.c`. Now our function can be used in `main.c`. Compute some n-th Fibonnaci number and print it on the screen by adding this line before the return statement `printf("Number : %d\n", fib_rec(n));`. `n` is a number of your choice.

4. Hit the green button. You should see the result of your computation on the screen.

2.5 Debugging

There is something wrong with the program above. Let's use the debugger to figure out the issue.

1. Navigate to `fib.c` and place a breakpoint at the line where the function `fib_rec` is declared. Breakpoints tell the debugger to stop execution at a certain point so that we may inspect the current state of the program at that point.

2. Go back to `main.c`. Call `fib_rec` with a negative number. Run the debugger by clicking on the green bug symbol next to the play symbol on the top right.

3. Keep executing the program by clicking on Resume program (the green play-pause symbol in the debugging screen). Observe the value of n. What is going on?

4. It seems we forgot to check for a corner case and our program does not stop executing. Exit the debugger by clicking the stop button (red square in the debugging screen).

We may learn more about the debugger later on in the course. It will come in handy for your assignments.

3 Terminal Compilation

We will now go over how to compile and run a C file via the terminal. You may need to do this in the course.

1. Review the terminal cheat sheet available here. You will need to use some of these commands.

2. Navigate to a folder you would like your project to be located in using `cd`.

3. Create a new folder using `mkdir` named `helloworld` (`mkdir helloworld`).

4. Navigate to this new folder using `cd` (`cd helloworld`).

5. Create a new file `main.c` using `touch` (`touch main.c`).

6. Edit the contents of this file using your favorite text editor. I prefer to use Vim but you are free to use whatever you like. Sublime Text is a popular option. You can copy and paste the contents of `main.c` CLion creates for you or create your own function. There are only two conditions for your compiler to recoginize the function. You must name at least one function `main` (this will be the entry point for your program) and, according to the C99 standard, this function should return an `int` type. Furthermore, exit code `0`, i.e. when `0` is returned from `main`, means that the program has successfully executed while any other execute code in the range `1` and `255` is up to the program creator to define what each code means.

7. Once you are done writing your program close your text editor and return to your terminal. In your terminal run the following command: `gcc main.c -o out` to compile your program. Here we are telling `gcc` (our C compiler) we would like to compile the source file `main.c` and have the output executable be named `out` which is what the `-o` flag does. If we had multiple files we could compile everything by doing `gcc main.c file1.c file2.c -o out`. Note that we never compile the header files. We can also change the name of the output file by changing `-o out` to say `-o hello`.

8. Execute the newly procuded executable by doing `./out`. You should see at least "Hello world!" printed on the screen.

At some point I may also go over running debugger via the command line (gdb). This is a little painful so it may not be worth spending time learning this.