Setup and First C++ Program (Lab 00)

Welcome to CSE 232, Summer 2018!

If you are intimidated by the length of this lab, please don’t stress. There are a large number of details that we need to take care of to make sure the lab runs as smoothly as possible.

In sequence, they are:

- Activating your engineering (EGR) account
- Starting and running the linux remote environment
- Activating Mimir so you can turn in your work
- Introducing the IDE
- Compiling and running your first C++ programs

While the lab as a whole seems daunting, each of these steps on its own should be straightforward and simple. Good luck!

The Lab

All of your laboratory sessions take place in the Engineering Building, room 3345. However, you may use any of the public computers in the Engineering Bldg as they have exactly the same configuration that will be described below.

The Machines

There are 24 workstations in 3345 for the laboratory sessions.

These are two monitors of Windows 10 machines. Though these are Windows 10 machines we will also be teaching you some Linux in this course!
Part 1: First Login

As of Fall 2017, the Department of Engineering and Computing Services (DECS) of the Engineering College will be managing all the public laboratory machines in the College. This is good news as it will give you access to more machines throughout the college.

- If you have never logged in to a EGR machine, never had a login, this section is for you.
- If you have a login but forgot your password, you are out of luck and will have to visit EB 1325. Bring your picture ID!
- If you have a working login, move on to Part 2.

First Login from the Labs

This is the screen you would typically start from for the EGR Windows 10 machines.

Simultaneously hold down the three keys Ctrl-Alt-Delete and you will get the login screen
Type your MSU netid as your user name. As you password type your pid, starting with a capital A and ending with the @ sign.

Activate Account

If you are in a lab, you'll be taken to your DECS Myaccount page https://www.egr.msu.edu/decs/myaccount/?page=activate.

If you want to activate from a web browser, you can visit the link from anywhere.

Pick "Activate Account" on the left and follow the directions. Make sure to select the button on the bottom that says you agree to following the EGR Acceptable Use Policy.

Since new accounts use your student PID for a password, accounts are vulnerable until this process is completed. At any point, you may receive an email reminding you of this.

Passwords

EGR requires a "strong" password. Your new password must have the following characteristics:

- Be at least 8 characters long (or longer). As a side note, the most useful thing about making a password secure is its length!.
- Must not include your NetID.
- Contain at least 3 of the following classes of characters.
  - Digits (numbers)
  - Uppercase letter.
  - lowercase letter.
  - Punctuation character.
How to change your password

Your EGR "my account" page is the place you can use to change your password, but it **does** require that you know your present password.

- Go to [https://www.egr.msu.edu/decs/myaccount/](https://www.egr.msu.edu/decs/myaccount/)
- Select Login (if it isn't already selected) and login with your present NetID and EGR password
- Select from the list of elements on the left column for what you would like to do.

Problems

If you cannot get it to work (try real hard) you can go to EB 1325. Bring your picture ID!

Lastly, log back in with your new password.
Part 2: Working with Unix

We are going to be working with Unix over the course of the semester, and through Unix a C++ compiler and other tools. You have a choice about how you interact with Unix and the compiler, which we will cover below. You can use any or all of the choices, but during the regular laboratory sessions we will be using x2go.

- **x2go**. You can x2go to login to a server which provides a full Unix desktop experience. You can do that from the labs or from you laptops at home.
- **Mimir**. The auto-grader/handin site we use is called mimir and provides a Unix IDE in a browser which can be used from anywhere.
- **Something Else**. There are lots of ways to get a compiler and/or unix on your own machine.

Part 2.1: Working with the Lab Machines

You probably know enough about Windows 10 to do work, but here's a little refresher to get you going. Everything runs off the "Start" button, the flag at the lower left of the screen.

When you click on start you get a scrollable list of all the programs you can run, such as a Mozilla Firefox browser.

Part 2.2: Setup for x2go

**x2go from the lab machines**

You can start x2go from the Start button as shown below
On the lab machines, you simply have to select what server you want (a number are preconfigured) and login. To access a different server, you have to:

- Open the program
- Select Session --> New session (this part is missing)

Note, you should not have to configure a new session on the lab machines.

**x2go from your laptop**

If working from your own laptop you need to download some software from [here](#). You need to download the x2go **client**. Make sure you download the correct software (for OSX or Windows) for your laptop.

For Macs, if you get a "application not trusted" or some such thing, do a Control-leftclick and open it. It will open by itself in the future.

**Setting up x2go**

DECS provides a detailed description for setting up your x2go environment. Follow the [directions here](#). These directions specify the server "rusty" but, at least from the lab machines, there are a whole list of servers you can use to run x2go. If one does not work, select another. List of x2go servers and their availability [here](#)

**Part 2.3: Using the XFCD desktop: x2go**

When you start x2go you get a window that contains a full Linux environment (CentOS is the distribution and XFCE is the GUI for the desktop). The window is a separate operating system, meaning that you have your native OS running, Windows 10 for the labs, and in the x2go...
window a full Linux. You can resize that window and, in the labs at least, use one monitor for your Windows env and expand the x2go window to occupy the other monitor.

When you get the x2go window, it should look something like the below. We've added a few labels so you can see the basic elements, but you are encouraged to look around a bit and see what you can find!
• Everything, all your applications, are somewhere off of the Applications menu at the top left. There are shortcuts, like for the bar at the bottom, but everything is accessible from here.
• At the top right, there are 4 boxes that are "workspaces". You can place a set of windows in each workspace then switch between them by clicking the workspace. There are keyboard shortcuts as well.
• Note: there may be a terminal window open on startup, so close that to get to the desktop

**Terminal**

Let's look at some applications. One that you will get more comfortable with is the Terminal. You can start it from the bar below or it is a featured element on the Applications menu

The terminal is a command line interface (CLI), which is a fancy way of saying that you don't use a mouse for commands, you type the commands. It is old fashioned and, once you get used to it, very efficient and very convenient. You'll get exposed to it over the course of the semester.

In the image below, I started the terminal then selected Edit → Preferences from the menu at the top of the terminal window. You can select lots of things, I changed the based color setup.

**File Browser**

Not unlike Windows or Mac OS, there is a file browser. I hovered over the file browser in the bottom bar, it provided some information, then I opened the file browser. When I open it I get to my home folder on the EGR servers.
Having opened my Home folder, we see a list of folders in the left column. Under the **Places** group are a series of shortcuts. For example, one of them is the Desktop. I can also make a shortcut by dragging a folder from the group of icons on the right to the **bottom** of the Places list. If you do not select at the bottom, you move the folder to the selected Places folder.

If you select the View menu option, you can change how the browser works. In particular view hidden files in your home folder. Lots of stuff there.

**Editor**

Editors are like religion: most people have one but don't like what anyone else has chosen. You will have the opportunity to pick the editor of your choice as you go along. The image below lists the 4 that are available in the XFCE/x2go server: atom, emacs, gedit, vim. Some basic info:

- Atom is a modern, programmer's editor. It is very flexible, has many options, but can be complicated to work with. If you want to be a programmer, this is a good one to learn eventually.
- Emacs is a very old, very configurable, editor. The joke is that emacs is an operating system disguised as an editor. This is what I use but I would not start here.
- gedit is a straight-forward, good place to start editor. It is not very configurable but does a good job at just being an editor. I would start here if you have no preferences.
- Vi/Vim is also very old, but comes as standard in many linux OS as a default editor. I wouldn't start here, but learning vi is something every programmer should do.

There is a brief tutorial on gedit [here](http://example.com).
Section 3: Mimir and Projects

For some labs, particularly this first lab, and all projects, we will be using the Mimir Platform. Mimir provides a convenient way to write, test, and submit your code. The assignments will have an associated Mimir page, including this one. Follow the directions below to submit your Lab 00 assignment for automatic grading.

Section 3.1: How to access Mimir

The Mimir access code is 7ec1027d66

First, Mimir is not free. It requires you to pay a $25 fee to join the course. Mimir is required software. You have to join Mimir as it is the only way you can turn in your work for the course. In the long run I think you will appreciate the ability to test your code before you turn it in, as well as having a simple interface via web browser requiring no software.

Mimir gives you a two-week grace period before you have to pay your fee. This gives you the opportunity to see the course before you pay your money.

You access Mimir by logging in to D2L, our course management environment. Select this course's page "CSE 232 Intro to Programming II in C++, Summer 2018". You should see the following:

In the Content Browser section, click on the folder called "Mimir".
Then click on the link called "CSE 232 FS17 Mimir link ". From that page, you should see a link called "Lab00". Go to it. This is the Project Submission page. On this page, you have see the project's description, and any project submissions you have turned in.

At the top right are three important buttons:

- "Submit": this is the alternate method for submitting code (where you need to zip and upload your code).
- "Download Starter Code": this gives you a zip file of the files we want you to edit.
- "Open IDE": this opens Mimir's terminal and editor

Click the "Open IDE" button to go to Mimir's IDE. It should open a separate tab that looks like this:
Mimir Workspace and IDE

On the left is a file browser. You should have a folder named "cse-232--introduction-to-programming-ii" for this class. The windows on the right can be open and closed. They are used to view and edit files. Please close the windows named "untitled" and "welcome.md". Then, using the file browser open the file inside Open cse-232--introduction-to-programming-ii → lab00 → hello_world → hello_world.cpp. Use the little arrows at the left of each name to open the directory below. Your tab should look like this:

To see how to compile and run the code on Mimir, see the addendum.
Part 4: Working with C++, introducing the Lab IDE

Section 4.1 About C++

From Wikipedia:

C++ is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation.

Languages come in various "levels", from "low" meaning very close to the CPU they might run on (assembler for example) to high, such as Python which abstracts much of that away. C++ is a kind of intermediate language, without all the help you might get from a Python but higher then an assembler.

It is important to remember the C++ is a compiled language. That means that you must pass your code through another piece of software called a compiler that translates your C++ into nearly machine-ready code. Note that Python has no requirement as there is always an interpreter available for running Python code.

The greatest strength of C++ is its potential for creating fast executables and robust libraries. C and C++ provide great flexibility in controlling many of the underlying mechanism used by an executing program. A programmer can control low-level aspects of how data is stored, how information is passed and how memory is managed. When used wisely, this control can lead to a more streamlined result. This is the point of the class. You are in charge of how your code runs because you (the programmer) are responsible for most aspects of how your code runs. This is a double-edged sword as being responsible gives you more opportunities to screw up.

One of the problems with C++ is that parts of the syntax have grown cryptic. More significantly, the flexibility given to a programmer for controlling low-level aspects comes with responsibility. Rather than one way to express something, there may be alternatives. An experienced and knowledgeable developer can use this flexibility to pick the best alternative and improve the result. Yet both novice and experienced programmers can easily choose the wrong alternative, leading to less-efficient, and possibly flawed, software. Be careful!

It turns out that C++ compilers are buggy software just like any other software. Thus depending on versions and vendors, C++ compilers can behave slightly differently.

The Rule of Compiling No matter how you develop your code, the only compiler that matters is the one on Mimir where you will turn in your code. If it compiles and runs there, great. If not, then it is up to you to get it to compile there. Only Mimir Matters
Section 4.2: The Lab IDE

Since the lab computers have all been upgraded to Windows 10, we are going to try a new IDE for the lab this summer – Visual Studio 2017. This comes preinstalled on all of the lab machines, and there is a free version available for any Windows machine (Visual Studio 2017 Community) as well as an open source version (VS Code) for non-Windows machines. Installing Visual Studio can be a lengthy process and takes up a fair bit of hard drive space – so be careful when installing on your own machines.

Opening Visual Studio 2017 should provide you with an opening screen that appears as so:

(You may need to log in with your EGR account first)

Section 4.3: Setting up a new project

First, create an empty directory named “Lab00” in your Documents or Desktop folder (Use either Windows Explorer or right click -> new folder on the Desktop). We will then create a project in this folder to create and test our code.

In Visual Studio, click File -> New -> Project from existing code. You should be presented with a pop-up like this:
Make sure that the project type is Visual C++ and then click next. You will then be presented with a screen like this:

Browse to the “Lab00” folder that you created, and then enter a project name in the corresponding text box (I recommend “Lab00”). On the next screen (see below) make sure that you are building the project as a **Console Application Project** through Visual Studio. At this point you can click “Finish” and a project will be created.

---

**Section 4.4: Writing and running code**

Once you have created a project, an empty window should pop up that looks fairly similar to this one:
This is your basic project page and is where you can organize information about your project. To start, we need to create a file for our code. Right click on the Source Files folder, select Add, and then select New Item.

A pop-up will appear prompting you for the type of file. You want to create a .cpp file called “hello_world.cpp”

Once you have created the file, copy and paste the following code into the window that opens up:

```cpp
#include <iostream>

int main() {
    std::cout << "Hello World" << std::endl;
}
```

To compile and run your code, press the green arrow in the task bar. A window should open up to show the results of the program.

If you are having issues keeping the window open, use the following guide:

- Start the project with Ctrl+F5 instead of just F5.
- The console window will now stay open with the Press any key to continue . . . message after the program exits.
- Note that this requires the Console (/SUBSYSTEM:CONSOLE) linker option, which you can enable as follows:
  - Open up your project, and go to the Solution Explorer. If you’re following along with me in K&R, your "Solution" will be 'hello' with 1 project under it, also 'hello' in bold.
  - Right click on the 'hello" (or whatever your project name is.)
  - Choose "Properties" from the context menu.
  - Choose Configuration Properties>Linker>System.
- For the "Subsystem" property in the right-hand pane, click the drop-down box in the right hand column.
- Choose "Console (/SUBSYSTEM:CONSOLE)"
- Click Apply, wait for it to finish doing whatever it does, then click OK. (If "Apply" is grayed out, choose some other subsystem option, click Apply, then go back and apply the console option. My experience is that OK by itself won't work.)
- CTRL-F5 and the subsystem hints work together; they are not separate options

(source: https://stackoverflow.com/questions/454681/how-to-keep-the-console-window-open-in-visual-c)
Part 5: Lab 00 (First Programs)

Lab 00 consists of two (hopefully simple) tasks. Correcting a typo in the file "hello_world.cpp" and creating a new file called "academic_dishonesty.cpp".

**hello_world.cpp**

I tried to write a "hello_world.cpp" file to show you how easy it is, but I made a typo. Take a look at the "hello_world.cpp" file; it should already be open when you open the lab on Mimir, but it is located in a folder with the class' name -> lab00__hello_world -> lab00.

Please correct my spelling of "Hello" then save the file.

Then right-click the folder named "lab00_hello_world" (sure you you select this one), and select "Submit dir to Mimir", be sure to select the correct assignment as well.

Now you can go to your submissions page (the other Mimir tab), and confirm that you are passing the tests regarding hello_world.cpp.

**academic_dishonesty.cpp**

The next part of the assignment involves making a new file called "academic_dishonesty.cpp".

Right-click on the folder named "lab00" and create a new file named "academic_dishonesty.cpp". Make sure the name is **exactly** what is indicated.

Copy the **contents** of hello_world.cpp into this new file.

Submit the assignment again like we did before. You should pass all but the last test. We need to fix that.

Modify the file academic_dishonesty.cpp to print out the following message **exactly** (instead of "Hello World").

My name is Bill Punch. I read the syllabus and I understand the consequences of academic dishonesty.

The file needs to have the exact message above, except Bill Punch should be changed to have your **actual name** instead (using only A-Z, a-z and space characters please). By submitting this assignment, you are confirming that you read the syllabus and understand the consequences of academic dishonesty.

Submit the assignment and you should pass all the tests and be done with Lab00, Congratulations!

To see how to compile and run the code on Mimir, see the addendum.
Addendum: Compiling C++ Code on Linux

Section x.1: Editing and Compiling Code

The best way by far to do your work is either through x2go or Mimir as we have discussed.

- x2go requires only the download of a single application which gives you access to a full unix environment for your everyday work from anywhere with an internet connection.
- mimir required only access to a web browser and the internet. While it does not provide a full unix environment, it does provide a unix terminal and a compiler, enough to do your work

However, since you are CS students perhaps you would like to do this on your own computer. Good news, C++ is available for any platform and has been for decades.

- For Linux, you already have it. Congratulations!
- For Mac OSX, you don't have it by default. However, you can get it if you follow these directions. They show you how to install Xcode, a large application developed by Apple for development, as well as the command line tools for your Terminal application.
- Windows, you don't have it, but two things:
  - You have available on the lab machines Visual Studio. You can try it out there if you like
  - You can download for free a copy of VS 2017 Community (go to Downloads → Software Catalog) and install it on your Windows laptop (Note there is a version of Visual Studio Code but it is just an editor, no compiler).
- Windows 10 also has a new feature (not yet enabled on the lab machines) called Windows Subsystem for Linux (WSL). Windows 10 does not come with a Linux command line but WSL provides one that looks very much like an Ubuntu 16.04 Linux terminal. You can set it up and run your code there, as you would on mimir. More adventurous, but take a look here. It really is quite nice!

Section x.2: Editing and Compiling Code

The first program you run in a new language is the Hello World program. This program does nothing put print the words “Hello World”. It is a tradition because it does very little except focus on the mechanics of writing your first program and running it. Look at the wikibooks page Hello world for more than 200 programming language examples of hello world.

In C++, Hello World is fairly easy, but clearly more work than something like Python! Pick an editor and save the following

```cpp
#include <iostream>
int main() {
    std::cout << "Hello World" << std::endl;
}
```
You can copy and paste the code into your text file (name it "lab00.cpp").

Next, the plain-text file needs to be compiled and executed. To compile with GNU's C++ compiler, type the following into the terminal:

```
g++ -std=c++14 -Wall lab00.cpp
```

`g++` is the GNU C++ compiler. In the above line, we have added a flag `-std=c++14` to ensure that the code is compiled using the newer C++14 standard (by default, most C++ compilers use the C++98 standard, which does not include all of the features/extensions that we will be using in this class). The `-Wall` is useful for finding errors. Although our "Hello World" program does not contain any C++14 extensions, it is a good idea to get into the habit of including this flag.

By default, `g++` will create an executable named `a.out`. If your compilation went correctly, you should see this newly created file in the current directory.

Finally, to execute/run the program type:

```bash
./a.out
```

The `./` simply means look into the current directory when attempting to find the `a.out` program. If all has gone according to plan, running the program should result in "Hello World" being printed to the command line.

Make sure you can run the Hello World program.

Note that to build a program consisting of multiple files in a directory, we will enter:

```
g++ -std=c++14 -Wall *.cpp
```

This will compile all the files that make up the source code of your program.

The typical workflow is something like:

- edit your code files using your favorite editor, then save (File -> Save) your code with a `.cpp` suffix
- In your command line terminal, enter, `g++ -std=c++14 -Wall *.cpp`
- If there errors (and there will be errors), read the error messages and re-edit your code. Repeat until it compiles
- Once your code compiles, enter `./a.out` in the terminal to run the compiled executable and observe the output, hopefully the correct output

You repeat the process of edit → compile → run until you get the result desired

**Section x.3: Reading and Producing Data**
Most programs running on a UNIX system read their data from **standard input**, write any data they produce to **standard output**, and send error reports to **standard error**. By default, standard input is taken from your terminal's keyboard, and standard output and standard error are both sent to your terminal's screen. You can redirect any of these three to read input from a file, write output to a file, or send data directly from the output of one program to the input of another.

To send the output of a program to a file, use the > (right angle bracket):

`./a.out > outputfile`

The following example uses **cat**, which is a standard Unix utility that reads files sequentially, writing them to standard output (the name is derived from its function to concatenate files). To redirect standard output and standard error to a file, use the command sequence (note, the dollar sign ($) is a marker for the prompt, hinting that you should run the command after it, you never actually type the $):

```
$ cat abcd >& errfile
$ cat errfile
```

In this example, we redirected the error message "No such file or directory" to the file "errfile". Therefore, no error message appears on the terminal until we list the contents of "errfile" with another cat command.

To take the input of a program from a file, use the < (left angle bracket):

```
$ ./a.out < input.txt
```

Again, **a.out** is the default name of output from g++, i.e. the executable of your program. If your program uses **cin** (the standard input stream, usually representing the keyboard), it takes its input directly from the terminal. However, you can place your input in a file and then send the file to **a.out** through standard input. The ability to read standard input from a file, rather than from the keyboard, is convenient.

You may often wish to redirect both standard input and standard output in and out of your programs. For example, to input input.txt to your program and send the output to output.txt, do the following: `$/a.out < input.txt > output.txt`

We'll be learning more about input and output, so hold on a bit for more detail.