EECS 211 Lab 2

Control Statements, Functions and Structures

Winter 2018

Today we are going to practice navigating in the shell and writing basic C++ code.

Getting Started

Let's get started by logging into a remote Northwestern server. We did this last week, but if you need help remembering the steps, they are included below.

The list of remote Northwestern servers can be found here: http://www.mccormick.northwestern.edu/eecs/documents/current-students/student-lab-hostnames.pdf

Windows

Open PuTTY. You'll need to enter a hostname to login to. The link on the right will take you to a list of student lab hostnames (such as *tlab-03.eecs.northwestern.edu* or *batman.eecs.northwestern.edu*). Ensure SSH is selected, then press Open. When prompted, enter your EECS username and password (not necessarily the same as your NetID password) and you're good to go.

Mac/Linux

Open up your terminal. At the prompt, use the ssh command of the form

```
$ ssh [eecs-id]@[eecs-host].eecs.northwestern.edu
```

where *[eecs-id]* is your EECS username (probably your NetID) and *[eecs-host]* is replaced by one of the EECS hostnames from the list of student lab hostnames (such as *tlab-03.eecs.northwestern.edu* or *batman.eecs.northwestern.edu*). When prompted, type in your EECS username and password (not necessarily your NetID password), press Enter again, and you should be logged in remotely!

Getting the code

Recall our basic Unix commands: *cd*, *ls*, *mkdir*, and *pwd*. What do they stand for and what do they do? Ask your TA if you don't remember. Use the following *wget* command to download the code into your directory of choice. It's easiest to just use your home directory, which is where you start out when you first log in to the server.

Or ask Google.

```
$ wget http://users.eecs.northwestern.edu/~jesse/course/eecs211/lab/eecs211-lab02.zip
```

Once we have our ZIP file, we will need to turn it in file directory using the *unzip* command.

```
$ unzip eecs211-lab02.zip
```

You should now have a directory called eecs211-lab02.

Setting up the build system

Type the `$ dev` command into the shell to ensure that you are using the correct developer toolset. You must do this every time you open a remote connection and plan on using CMake. Next, change your directory to eecs211-lab02. Make a new subdirectory called build and navigate into it. Then, the first time you set up the project, use the command *cmake* in order to setup the CMake build system.

Note the two periods, which tell *cmake* to run on the parent of the current working directory.

```
$ cmake ..
```

Writing the code

Navigate to your eecs211-lab02 directory, and open up lab2.cpp in Emacs using

```
$ emacs -nw lab2.cpp
```

Notice that there is already some skeletons of functions and some code in *main()* here.

Iteration

First, find the function called *sumNumbers*.We are going to use this function to sum up all of the numbers from 1 to num. If you remember from class, we have a few ways of iterating through numbers, most notably *for* and *while*. We will be using both, but first we will be using *while*.

Notice that *sumNumbers* is going to return an int.

While loops

As we learned in class, a *while* loop has the following syntax:

Note that in while loops we usually will use a boolean expression for <expr> (an expression which returns True or False)

```
while (<expr>) {
    // Looping through code here
    // Until <expr> is false
}
```

Use a while loop inside our *sumNumbers* in order to add the numbers from 1 to num together. Make sure to use a *return* statement to return the sum that we aggregated!

Remember that we have the ++ and += functions to help us.

Once you think that your function works as intended, save and exit emacs, then navigate to the build directory. If you remember from last week, we used the *make* command in order to turn our C++ file into machine code. Making sure you are in the build directory, type in

C-x C-s to save and C-x C-c to exit

```
$ make lab2
```

If everything works, if we list our files, we should now see a file called lab2. Enter the command

Remember, make works as follows: `$ make [target]`. Target is the name of the executable file that will be built by the make command.

```
$ ./lab2
```

See if your value looks right! If it doesn't, don't worry, Rome wasn't built in a day. Go back to your lab2 directory and try and see what went wrong. Play around with the value of numBound and see how it affects the result.

Error messages may look scary, but in reality, they're there to help you! Not intimidate you!

For loops

Once we have everything working with our *while* loop, let's work on using a *for* loop. To help you remember the syntax, here is an example of a for loop to print out the values from 1 to 5

```
for (int i = 1; i <= 5; ++i) {
    // Note that the code inside the curly braces is the code
    // That is ran each iteration of the for loop
    cout << i << '\n';
}
```

Go back to our *sumNumbers* function, and try to replace your *while* loop expression with a *for* loop to accomplish the same task.

Once you are done, again, navigate into your build directory, then make and run your file. See if everything looks the same! If not, no worries, go back and try again!

Structs

Structs are an important tool in C++ for grouping data together. In your lab2.cpp you will notice that we created an Apple structure for you. This is so that we can organize attributes of Apples together in a convenient way. If you look inside our Apple struct, we decided that we will want to know the weight, the variety, and the color of our apples. In our *main()*, we created an example of a Red Delicious Apple. Now, create your own type of apple (you'll need to define it as a type Apple), and give it those three attributes. Add in a print statement (we gave you an example one) to print out your new

How 'bout 'dem apples?

Apple. Navigate back to your build directory, and then make and run your lab2 file! Hopefully everything works as expected! If not, don't fret or get upset, go back and make changes!

Now that we know all about Apples, let's create our own structure. Define a structure of your favorite animal, and give your animal three attributes, with one of them being age. Don't forget to give your attributes types. You can create a new struct that looks very similar to the Apple we created.

Don't forget the semi-colon after the closing brace.

Once you have created your animal, go into *main()* and create an instance of your animal, assign it those three attributes, and then create a print statement to print out information about your animal. These print statements are getting annoying; we'll tackle that soon.

Navigate back to your build directory, make and run lab2, and see if your new animal shows up the way that you intended. Hopefully everything works! If not, as usual, go back and try and find what went wrong and update your code.

Creating your own function

So far, we've been filling in skeleton functions that were provided for you. Now it's time to write your own function from scratch. Remember how annoying it was to type out the *cout* lines each time you wanted to print out your animal? We're going to abstract that out and replace it with a simple call to a function!

Write a function called printAnimal that uses *cout* to print out your animal's three attributes. Note that this should take in one argument(of the same type as your animal struct). Think about what type your function should be!

Note that the void return type signifies that nothing is returned

Once you wrote your function, go to your *main* function and replace your print statement with a call to *printAnimal()*.

remember to pass your animal instance to the function

Go back to your build directory, make and run lab2, and see if everything still works!

Control statements

Now that we have gotten the hang of structs and functions, let's practice our control statements. Go back to our *printAnimal* function. Remember from hw1 and class that if statements have the basic following syntax:

```
if (<test-expression>) {
    // run if <test-expression> evaluates to true
} else {
    // run if <test-expression> evaluates to false
    // note that you don't necessarily need an else case
```

```
}
```

Using an if and else statement, check your animals age and add to your *printAnimal* function a line to print out "This animal is old!" if the animal is at least 10 and print out "This animal is not that old" if the animal is younger than 10.

Go back to the build directory, make and run lab2, and see if this feature is working!