EECS 211 Lab 3

Using CLion, and more practice with Vectors and Loops

Winter 2018

Today we will be installing an IDE (Integrated Development Environment) called CLion to make development of our C++ programs a little bit easier. In addition, we will be practicing the programming tools we've learned over the last few weeks!

Installing CLion

The first thing we will be doing today will be installing CLion in order to make writing C++ programs easier and more user friendly. First, go to JetBrains's website and apply for a student account:

JetBrains is the developer of CLion and lots of other developer tools

www.jetbrains.com/shop/eform/students

Once you submit your information, you will be receiving a confirmation email. Click confirm, then you will immediately receive a second email with a link to complete your JetBrains registration. For the installation of CLion, we will have slightly different steps for Windows and Mac laptops:

Windows

Before installing CLion on Windows, you need to install a C++ toolchain. Download MinGW from here:

A toolchain consists of a compiler and other programming tools.

http://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/installer/mingw-w64-install.exe/download

Follow the prompts to install MinGW—make sure to select the x86-64 architecture. Take note of where you install it, as you will have to configure CLion to find it.

Then go to this link to get CLion:

https://www.jetbrains.com/clion/

Press Download, then install CLion through their steps and login to your JetBrains account in order to activate your CLion editor. When prompted for the toolchain location, provide the path where you installed MinGW; this should be a directory that contains subdirectories with names like bin and lib. Check all of the "Create associations" boxes when they appear. Besides that, the default installation settings should work fine.

Don't worry about Plugins.

Mac

OS X automatically installs its toolchain when you attempt to use it from the command line for the first time. Thus, to install developer tools, run the *Terminal.app* program (from /Applications/Utilities) to get a command prompt. At the prompt, type

A toolchain consists of a compiler and other programming tools.

```
$ clang
```

and press return. If it prints "`clang: error: no input files`" then you have it installed already. Otherwise, a dialog box will pop up and offer to install the command-line developer tools for you. Say yes.

Then go to this link to get CLion:

https://www.jetbrains.com/clion/

Press Download, then install CLion through their steps and login to your JetBrains account in order to activate your CLion editor. The default installation settings should work fine.

Don't worry about Plugins.

Using our first CLion Project

Once you have CLion installed, download the zip file from the course site:

http://users.eecs.northwestern.edu/~jesse/course/eecs211/lab/eecs211-lab03.zip

Once you have downloaded the zip file onto your laptop, extract the zip file into its own folder. Make sure you keep track of which folder it's in! Next, open up CLion and Click on File –> Open Project, and click on the Lab 3 project that you just unzipped.

Once you open the project, test out the output from the program already loaded on your screen. In order to do this, look into the top right corner of your CLion Window. Press the button that has a down arrow with 1s and 0s. This is the Build button. This will essentially set up our CMake environment we previously would have to build on the command line. Then, press the arrow that looks like a play button just to the right of that, called the Run button. The first time you press it, in the "Executable" line, click the dropdown menu down to "lab3." Next hit Apply, then you can hit Run. Notice in the subwindow on the bottom of your CLion window. You can now see our nice message in the output! This should remind you of DrRacket from EECS 111. You have a top window where you edit your code, and a bottom window which displays the output of your code when it runs.

One big advantage of IDEs are their ability to abstract out the command line.

Continuing with Vectors and structs

If you remember from class, a Vector is a way of storing a group of items together of the same type. The basic syntax of a vector for a declaration is:

```
vector<type> varName;
```

You can initialize the vector with a set of values using the following notation:

```
vector<type> varName = {var1, var2, varN};
```

Notice in the lab that we created a struct called `Dog` which is located in our `Dog.h` which has a few member variables. In this lab, you will be the owner of a dog sanctuary, where you want to make sure that all the dogs are happy and getting along well together. Back in `lab3.cpp`, in your main function, we created a few instances of Dogs and gave them names, ages, and happiness ratings. We also created a vector of the type Dog called `dogs`, initialized to contain all of the Dogs that we created. Feel free to create one or two of your own dogs and add them to the vector!

Note that the happiness rating is at the discretion of your assistant and may be wrong, but they're doing their best so cut them some slack!

The first thing that you will be doing is trying to find the youngest dog at your sanctuary, so that you can warn guests to the sanctuary about them teething on the guests' fingers. Go to the function skeleton we gave you for *youngestDog* in `Dog.cpp`, and try to write a function that will return the Dog who is the youngest. You will probably want to use a *for* loop, which you may recall has the form:

That could stop donations from coming in to help out all the dogs!

```
for (size_t i = 0; i < vectorName.size(); ++i) {
    // Do stuff here with i, for example:
    cout << vectorName[i] << '\n';
}
```

We put an example *foreach* loop in the main function to help you out.

Call the function you just wrote in the main function, and print out the name of the youngest dog from your function's result using *cout*.

Remember that you should be returning the youngest Dog, not the youngest Dog's age.

This time you just have to press the run button, and it will automatically build your program for you and run it!

Now, once you have written *youngestDog*, write a similar function called *happiestDog* where you have to find the happiest Dog in the vector, and return that dog.

Again, go back to the main function and call your new function on the `dogs` vector, and print out the name of the happiest dog.

Once you have found your happiest and youngest dogs, let's create a function that can help you find a dog's owner given only

the dog's name. Fill in the function *findOwner* which takes in both a vector<Dog> and a string, where you have to return the name of the Dog's owner which you found from the vector.

Once you have this, try it out by calling it in your main function to ensure that you are able to find the name of a Dog's owner, and test out a few examples, printing out the names of the dog owners.

Dealing with Errors

Now, let's consider the case where the dog name inputted to the *findOwner* function is not in the dogs vector. In this case, let's try and print an error to the output, so we know that we have a weird solution. We can use *cerr* to print out this to the output window for our special error output.

Hopefully the dog sanctuary can find the dog soon though!

Call *findOwner* with arguments which should create this error to occur. Notice the color of the text in your output window for the error!

Now, remember from class that while *cerr* will output an error in your output window, it will not cause the program to fail/exit. Now, let's assume that if you can't find the dog name in the dogs vector that you want to stop running the program. We can throw a runtime error using the following syntax:

```
throw runtime_error("Message for the error");
```

Now, notice when you run the program again, in the same call you made before to *findOwner*, when you hit the error, instead of silently outputting an error, your code will stop running entirely. You can test this out by adding a *cout* line after your call to *findOwner*, and see that it never runs!

Testing

For your homework, you will be writing your own tests to test the code. We have provided you with tests already for this lab, but it is important that you understand how to run unit tests using CLion, so we will go over that now.

Click on the dog_test.cpp file that we have provided in the project. Notice how we have already written your tests. For your homework, you will have to write your own! Now let's run the tests to see if your code passes. In order to do so, go to the top right corner of your CLion window, and click on the dropdown menu in between the build and run buttons. You'll notice that your dropdown menu currently says "lab3". Now, go down to "dog_test" and select that. Press run, and you will see the results of the tests running!

If you want to go back and run your main code, you can easily go back to the dropdown menu and select "lab3" instead of "dog_test."

Please feel free to post on Piazza with any questions, or ask your TA!