Lab 1 - Intro to CLion and C++

Lab Topics

1. Creating a simple C++ program with CLion.
2. Explore compile time errors.
3. Configure code formatting.
4. Accessing Course Management System and submitting code.

Directions

• The labs are marked based on attendance and effort.
• It is your responsibility to ensure the TA records your progress by the end of the lab.
• While completing these labs, you are encouraged to help your classmates and receive as much help as you like. Assignments, however, are individual work.
  You may not work on assignments during your lab time.
• If you complete the lab early, you should experiment with C++; however, you may leave if you prefer.
• If you do not finish the lab exercises during your lab time, you are encouraged to complete them later to finish learning the material. You will still receive full marks if you arrived on-time and put in your best effort to complete the lab.

1. Setup

If you are working in the lab then:

1. Boot the computer to Linux.
   ◦ If it’s currently in Windows, restart it; during the early boot it will give you a text menu (“GRUB”). Use the keyboard’s arrow keys to select “Ubuntu” (Linux).

2. Log into the computer.
   ◦ If you are unable to log in (rejects your password), then:
     1. Retry your password (ensure you typed it correctly)
     2. Reboot the computer back into Linux (some networking issues are resolved by restarting the machine).
     3. Submit a help ticket by emailing helpdesk@cs.sfu.ca. Include:
        Course: CMPT 130 with Dr. Fraser
        Computer ID: <fill in the name of the computer you are at; should be displayed on screen>
        Explain what you are trying to do, and what happened.
     4. Try another computer in the lab.

If you are working from your own PC, see the course website (Resources page) for directions on installing CLion.

• When installing, if double clicking on clion.sh shows you the contents, then right-click
and select Open With --> Other Applications... and click View All Applications and select Run Software.

2. Hello World

This section shows how to use the CLion IDE (Integrated Development Environment) to create and run a very simple program.

1. Launch the CLion IDE
   - Click the Ubuntu icon in the very top-left hand corner of the screen. Type in clion, and select CLion.

2. If asked to import previous settings or not, select no settings to import.

3. If shown, accept the privacy policy.

4. In the lab, if asked for “CLion Licence Activation” then:
   - Select the “License server” option.
   - Enter the server: http://cs-sc-licensing.cs.surrey.sfu.ca:8023

5. Select any UI Theme you like, accept the default tool chain, accept the default plug-ins, and it does not matter if you select to create the desktop icon or not.

6. If the “Welcome to CLion” window comes up, then select New Project. Otherwise go to the menu File → New Project

7. Enter the project name and location:
   - Project name: Lab1
   - Project location: /home/YOURIDHERE/sfuhome/cmpt130/Lab1
     (Replace YOURIDHERE with your SFU id, such as bfraser. Case sensitive)
     The sfuhome/ directory is the same folder which is mounted on all (most) computers across campus. See this guide for suggestions for accessing your files from home. If you are following this guide at home, you can create the project anywhere (the default location should be fine).
   - It should look as follows:

     ![New CMake Project Dialog](image)

     - Click OK.
   - If asked about open project in a new window or this, select either This Window or New Window.
8. You will now see the CLion IDE open to your project, complete with a default little C++ program to get you started:

![CLion IDE window]

- OK to disable the tips at startup.

9. Display the Project window: View → Tool Windows → Project
Or press **Alt-1**.
Expand the “Lab1” folder:

![Project window]

10. `main.cpp` should already be displayed in the IDE’s code editing section. You can open files by double clicking on them in the Project view.

11. Replace all the text in `main.cpp` with the following code (you may copy-and-paste this code to speed up the process, but you may have to correct the spacing and tabs to make it look like the code below).

```
// Lab 1 Program
#include <iostream>
using namespace std;

int main()
{
    // Some simple output to the screen.
    cout << "Hello lab world!\n";
    return 0;
}
```

© Dr. B. Fraser
12. **Save** your file.

13. **Compile** the program by using the menus and selecting **Run → Build**. Alternatively, try pressing **Ctrl+F9** will work. This will save you a lot of time!
   - If it succeeds in compiling, it will say “[100%] Built target Lab1”
   - Make sure there are no errors in the Output window. If there are, double click on the error and correct the problem; see next section for how to fix them.
   - A successful compile will look as follows:
14. Run your program by selecting **Run → Run ‘Build All’**. Or, press **Shift + F10** as a shortcut. You should see the “Edit Configuration” window.

15. Once per project you may have to select what you want to actually run. In the “Edit configuration” window do:
   - Select the **“Executable”** of **Lab 1** from the drop-down.
   - Click **Run**.
   - In the output section at the bottom of the IDE you should see the output:

16. Congratulations! You’ve got your C++ program running!

17. Add another output statement (**cout**) or two. Perhaps have it print out your favourite joke, a meaningful saying, or a greeting message.
3. Exploring Compile Time Errors

1. Add a syntax error to your code (for example, remove a ';' from the end of a statement).
   - Notice the editor shows an underline on the statement(s) in the file. If you mouse over the red-underline, it will show a description of the error.

2. Try to run the program (Shift-F10). Notice the error message in the output pane (shown in blue). Click on the error message to jump straight to the line of code which has the problem. (You may need to scroll up in the output window to see the error, which is blue and underlined).

3. Correct the error. Recompile (or run) your program and the error should disappear.

4. Sometimes error messages are hard to figure out. Change the “<<” (stream insertion operator) to “>>” (stream extraction operator) in a cout statement.
   - Again build the project and notice that an error is displayed in the editor.
   - Look at the build output in the “Output” panel. Notice that there are multiple build errors! Scroll up to the first error. It may resemble the following:

   ```
   /home/brian/sfuhome/cmpt130/Lab1/main.cpp:8:10: error: no match for 'operator>>' (operand types are 'std::ostream {aka std::basic_ostream<char>}' and 'const char [18]')
   cout >> "Hello lab world!\n";
   ^
   
   This error means that cout does not have a “>>” operator that will work with a string (perhaps not perfectly clear!).
   - If you click on this error, it will take you to the line of code in main() which is using the incorrect stream operator.
   - However, if you click on any of the other errors (which have even less useful descriptions) you'll be taken to other files. Here are a couple rules to figuring out build errors:
     1. Always look at the first error (only). Track that one down, then rebuild (i.e., recompile/run). One problem in your code can cause many build errors.
     2. The problem is (almost) always going to be in code you write, not in code provided with the compiler. Therefore, if you follow an error and it's showing you code you didn't write, look elsewhere for the error.
     3. The actual problem in your code may be a line or two above where the compile tells you it thinks the problem is.
   5. Correct all compile time errors and re-run your code to prove it is working.
4. Code Formatting

“There is always time to correctly format your code. Always.”

- Correct code formatting means having the start of each line of code correctly aligned using tabs, plus (later we’ll learn) where to put the { and }’s.

- Always format your code correctly as you write your code.

- Students sometimes ask me about a problem in their code when their code is also incorrectly formatted. Often this incorrect formatting can make it hard to find the bug! So, always format your code correctly as you write your code.

- If you ask for help on incorrectly formatted code, I’ll say, “I can’t read your code; it’s incorrectly formatted.” However, I am very happy to help you reformat your code!!

- Hint: Always format your code correctly as you write your code.

- As you code, CLion will generally give you pretty good indentation! Each time you hit enter, it will usually start the next line at a good place.

- Nothing need be done for this part of the lab, other than to have a look at how your code is formatted.

- In summary:

Always format your code correctly as you write your code.
5. Exploring cout

Get used to exploring and self-discovery with these labs. Here's an easy start:

1. Experiment by adding additional cout statements in your code. Have it print out the name of your favorite colour.

2. Add statements to your program to, in addition to the above behaviour, print out the following text, including the two lines of ten *'s:

**********
A quote by Sir John A. Macdonald:
Let us be French, let us be English, but most importantly let us be Canadian!
**********

Note to Python programmers: C++ does not have an operator to repeat a character; you will have to type the all the *'s you want directly.

6. Submission Server

Assignments are submitted using CourSys. This tool also shows you your grades for this course. Get in the habit of submitting carefully! For assignments, you can only get marks if you submit your work correctly.

1. Create a .ZIP file which contains your .cpp file from this lab as follows:
   a. Open a file browser window to your home directory. You'll find an icon for your home directory on in the top of Linux's application-bar, far left side of the screen.
   b. Brows to the sfuhome/cmpt130/Lab1 director.
   c. Right click the file main.cpp, and select Compress. Note: You can hold control and then click on multiple files before selecting Compress if you need to submit multiple files in a ZIP file.
   d. Name it lab1Submission
   e. Select the file type to be .zip.
   f. It should look like this:

   ![Compress Window]

   g. Click Create.

2. Double click on the lab1Submission.zip file. You should see a window showing the contents of the zip file. Ensure that the main.cpp file is listed in your zip file correctly.
Before submitting any ZIP files to CourSys, always double check the correct files are included.

3. In a web browser, go to CourSys:

https://courses.cs.sfu.ca/

4. Login using your standard SFU ("Unix") user ID and password (the same one you use for connect.sfu.ca). This is not your student number (like 209443517), it is the same as your email (like bfraser).

5. Select CMPT 130 --> Lab 1.

6. In the Actions box (top left), select to submit an assignment. Upload your ZIP file.

7. You should now be able to view the file you submitted.

   If you resubmit a file, only the last version you submitted will be marked.

   Note that we are not marking the correctness of this lab. What you submit here is just to show completion of the lab. It may include code from the Extra Challenge section if you like.

### 7. Extra Challenge

Try these tasks for an extra challenge. It's OK to add to your existing code for this lab.

- **Using cout statements, draw a rectangle on the screen out of *'s:**
  
  ********
  *      *
  ********

- Write some code which prints out a Haiku or a (clean) limerick. Format it so that it looks good on screen.

- Write some code which outputs the following text on the screen:

  He'll say, "WOW! She\he did great!"

### 8. Skills and Understanding

You should now know how to:

1. Create a new C++ project, compile and run it.
2. Use cout to put output on the screen.
3. Submit via the CourSys.

When you finish the lab, **LOGOUT OF THE LAB COMPUTERS!** If you leave yourself logged-in, the computer will remain locked and our IT staff will have to manually log you out. Doing this too many times will cause an account to be blocked.