Lab Exercises wk02 – Lab Basics
First Lab of the course

Optional Reading

Problem Solving with C++ - Chapters 1 to 4 (Review)

Instructions – PLEASE READ *(notice **bold** and underlined phrases)*

Lab Exercise has three parts:

A. Lab Demo – Watch Demo, reproduce it, show your TA
B. Exercises – To be started during the Lab, and completed by deadline
C. Submission – Submit specified exercise files by deadline

1. You are encouraged to work on these lab exercises in teams of two, however each student must reproduce the demo and show the TA their own running C++ Project. Team members should take turns at the keyboard typing in the code, or use two terminals side-by-side. You will start working during the lab time, and it is expected that you will complete the demo during the lab, but may need additional time to continue with the exercises later.
2. **Submission deadline: 10:30am Friday January 15th**. You should be able to complete the work if you have completed the required reading for the lab. Some topics discussed in this lab and needed for the exercises to be submitted may be seen later in this week's class! Please submit before the deadline. No late submissions are permitted. You may resubmit without penalty provided you resubmit before the deadline. Be sure to download and test all submissions.
3. The exercises are roughly presented in sequence so that you gradually advance with the material. It is highly recommended, for your own benefit, that you do ALL the exercises, and in the order provided. You will get lab participation points even if you do not finish all the exercises (but you do have to work on some of the exercises besides the one to submit in order to get full participation points).
4. For this lab, ONLY ONE EXERCISE SHOULD BE SUBMITTED for marking: the last exercise. If working in pairs you should submit only one version of the file and submit it for the group that you define. If you are working individually, you will still need to define a group with you as the only group member.
5. Keep a copy of everything you submit in your files.
6. Before leaving CSIL, make sure the TA looks at your work and records your attendance in order to receive your lab attendance and participation marks.

A. Lab Demo – Presented by TA, and repeated by Students

Students must observe demo presented by TA, then reproduce it.

Student must have the TA check off demo program completion by end of tutorial for full marks. Marks will be awarded for attendance but only partial marks for incomplete work at the discretion of the TA.

Even though students may work in teams, <u>each team member</u> must create a running project based on the demo for full marks.

Using Computer Systems in the CSIL

To work effectively in the Computing Science Instructional Laboratory (CSIL), it's helpful to understand that several organisations work together to provide computing services to students at SFU.

SFU IT Services[1] (ITS) provides general computing support to all of SFU. This includes a **`sfuhome`** directory that is accessible from any of the ITS Computer Labs[2] across the campus.

Individual departments often have their own computing facilities to serve the specialised needs of the courses offered by the department. In Computing Science, this is CSIL. Each CSIL Linux workstation has its own local home directory.

The home directory provided by SFU ITS is distinct from the home directory provided by each Linux workstation in CSIL, and each home directory provided by Linux workstations in the CSIL is local to that workstation. Keep this in mind as you work this semester. You should use your **`~/sfuhome`** directory to hold course work so that you can access it from any workstation in CSIL and from different computing labs around the campus.

Logging on to Ubuntu and Using NetBeans

Log in to an Ubuntu Workstation

If the workstation is displaying the Windows login screen, reboot the workstation. At the GRUB boot prompt, select Ubuntu linux from the menu. Log on to the workstation from the Ubuntu login screen. Use your normal user id and password.

When you first log on to an Ubuntu workstation in CSIL, you are placed in the local home directory for that workstation. Programs that expect to place data and configuration information in your home directory will use the workstation home directory. Your ITS home directory is mounted under sfuhome.

Getting Started with Ubuntu

If you have never used Ubuntu 14.04, it can be a little confusing at the start. The first thing to know, is how to find and start various applications. On the left hand side of the screen, there is the Launcher window where you can place your favorite applications. To search for an application, Click on the top most icon in the launcher or hit the "Super" button on the keyboard (the key that looks like the Windows Logo, between the Ctrl and Alt buttons in the lower left of most keyboards). This will open a prompt to search your computer and online sources for applications. You can enter the names of various applications here like "terminal" to find the Terminal application. Double-Click on the application icon to launch it.

Another way to launch a terminal in Ubuntu is to hold down the <Ctrl><Alt> keys simultaneously, then press the "t" key. This will bring up a terminal command line window. For a list of other keyboard short cuts, press and hold the Super key.

Before starting NetBeans for the first time, create a linux directory for your Cmpt 135 course work under **sfuhome**. You can do this using the Files or Nemo icon from the launcher or via a command line terminal.

Starting NetBeans

NetBeans uses a workspace to hold your programming projects, one directory per project. The workspace itself is simply the top-level directory that holds the directories dedicated to each programming project. You can have more than one NetBeans workspace. A common approach is to have a separate workspace for each course that requires programming. Each assignment can be a separate NetBeans project within a workspace.

To create a NetBeans workspace in your sfuhome directory, execute the following commands from a Linux command shell:

```
cd ~/sfuhome
mkdir cmpt135
mkdir cmpt135/labWorkspace
```

The first command will change directory ('cd') to your ITS home directory **sfuhome**. Tilde ('~') is the Linux abbreviation for your home directory. On CSIL linux systems, '~' is the home directory that's local to each workstation, whereas ~/sfuhome will change your working directory to sfuhome. The second command will make a directory ('mkdir') named **cmpt135**, while the third command creates a directory called **labWorkspace** within the cmpt135 directory.

Remember that unix directory names are case-sensitive. Also keep in mind that directory and file names with embedded spaces are inconvenient in a command line environment. They should be avoided because they can also cause problems for our course grading scripts.

To start NetBeans, press the Super button, then type "netbeans" at the prompt (without the quotes). Double-click on the NetBeans icon. If NetBeans is unavailable in CSIL, then use Eclipse but realize that some changes to the steps are required. The Menus for creating a new project are different between NetBeans and Eclipse, but both are similar. In addition, when creating a new project, NetBeans automatically adds the line "`using namespace std;`" to main.cpp, but this must be done manually by the student when creating a new project with Eclipse. Ask your TA or instructor for help as needed.

---

[1] http://www.sfu.ca/itservices.html

[2] http://www.sfu.ca/itservices/technical/computerlabs.html

Creating a HelloWorld Project

Once NetBeans is started, you will see the NetBeans "Start Page" in the first tab of the main window. This page contains useful Tours, Demos, and Tutorials for using NetBeans. You may want to visit this later if you want more information about how to use NetBeans.

New Project: To create your HelloWorld Project, click on the NetBeans File menu and select "New Project". Select C/C++ under Categories, then C/C++ Application under Projects, and hit Next.

Project Name and Location: For the Project Name, enter "Lab02Hello". To set the Project Location, hit the Browse button to find and select your **`labWorkspace`** directory. Make sure that "Create Main File" is checked, and then click the Finish button. This will create a new project in the folder **`~/sfuhome/cmpt135/labWorkspace/Lab02Hello/`**

Write your Lab02Hello Program: Double-click on the Lab02Hello project in the projects menu, then double-click on "Source Files", and finally double-click on main.cpp. The file main.cpp will be opened in the Source edit window.

Add the following two lines after the #include on line 8:

```
#include <iostream>
#include <cstdio>
```

Add the following two lines after line 23 that contains "main":

```
cout << "Hello World using cout!" << endl;
printf( "Hello World using printf()\n");
```

Run your Lab02Hello Program: From the Run menu, select "Run Project". If there are no errors in your code, this will cause NetBeans to invoke the gnu C++ compiler g++, compile and link your application, and run it. You should see the following in the Output window at the bottom of your NetBeans screen:

```
Hello World using cout!
Hello World using printf()
```

When you submit your course assignments to CourSys, you will submit only the specified files with the specified file names, not the entire project directory. To find the files, open a Terminal window using <ctrl><alt>t, and change your working directory to the location of your project folder. You should see the file `main.cpp` there. Use the cat command from the terminal window to verify that this is the correct file as follows:

```
cd ~/sfuhome/cmpt135/labWorkspace/Lab02Hello/
ls
cat main.cpp
```

The **`ls`** command will list the files in your project folder. You should see some directories plus a Makefile and your `main.cpp` file. The linux **`cat`** command copies the contents of the specified file to your screen, and this output should match the **`main.cpp`** file you wrote in NetBeans.

Congrats! You have just run your first C++ program of Cmpt 135! When you leave NetBeans your files will still be saved under your project folder. When you re-enter NetBeans, it will return you to your last project by default.

B. Lab Exercises – To be completed by Students

Students are responsible for having the TA look at your work by end of tutorial for full marks. Marks will be awarded for attendance and for making progress on the Lab Exercises. The exercises do not need to be completed by the end of the lab for full marks, but some progress needs to be shown.

Students may work in teams of two and complete the exercises as a team.

Learning Objectives: By the end of this lab, students should be able to:

- **Use NetBeans on Ubuntu to create a C++ Project**
- **Create basic C++ input and output using cout and cin**
- **Use boolean expressions and IF statements to control program flow**
- **Use variables as parameters when invoking functions**
- **Define and use simple programmer defined void functions**

Lab Instructions:

1. Create a new C++ project called Lab02
2. Edit your main.cpp and insert the following statements into main() which will print out a message. Copy/paste the statement below directly to NetBeans into the ***main()*** method body.

```
cout << "Lab02 is working!" << endl << "Amazing eh?!";
```

3. Save your changes and Compile your Lab02 project by hitting F6
   a. The F6 button (at the top of your keyboard) is a short cut for telling NetBeans to Save your changes, compile your code, and run the code if it compiles successfully. If there is a syntax error, it will report any errors and stop.
   b. Did F6 find any compile errors? If a compile error was detected, you will not be able to run your code. The line where the compile error was encountered should now be highlighted in the main.cpp tab.
   c. The object **cout** used in **main()** has not been defined yet. This is a compile error. Identifiers need to be declared before they are used in C++.
   d. Can you spot the error? What is missing? Hint: it is described on page 25 of the text. Did you read the text in advance before you came to the lab? If not, try to keep up with the required reading. Doing so will help you learn and save you time.
   e. Fix the error by adding the missing #include statement after line 8:

```
#include <iostream>
```

   f. Now save the changes and hit the Run Project button again. Does your program run now? If not, compare your main method with the one from Lab02Hello and fix any errors. It should compile and run when correct.

7. Once you are able to get the code to compile, it will run and produce output in the console window at the bottom. Did it print what you expected? What does the token "**endl**" do?
8. Our program uses the object **cout** to produce output, but a program that takes

no input can do nothing interesting. Let's use object **cin** to accept some input. Add the following code to declare an int variable called **num** and accept an int input from the terminal via **cin**.

```
int num = 0;
cout << "Enter a positive number from 1 to 10: ";
cin  >> num;
cout << "You entered the number " << num << endl;
```

9. When you are done, the body of main should look like the following:

```
int main(int argc, char ** argv) {
   int num = 0;

   cout << "Lab02 is working!" << endl << "Amazing eh?!";

   cout << "Enter a positive number from 1 to 10: ";
   cin  >> num;
   cout << "You entered the number " << num << endl;

   return 0;
}
```

10. Hit F6 to save and compile and run your project. If it does not compile, look at the error messages and fix any errors. Does the program accept an integer and display it properly. Run it several times with different numbers as input. Does it work as expected? What happens when you enter a negative number? What happens when you enter a large floating point number like 110.6? What happens when you enter a string like the name "Bob"?

11. Good code should validate input and react to bad input. The simplest reaction is to print an error and exit. Let's add code to check if the number entered is less than one or more than 10, and report an error and exit if it is out of range. Refer to Chapter 2.4 "Simple Flow of Control" of the text if you need a refresher on the syntax of IF statements. Add the following code before the **return** statement in main():

```
if ((num < 1) || (num > 10)) {
    cout << "Input " << num;
    cout << " is out of range. Exiting" << endl;
    return -1;
}
```

12. Run the new code and verify that inputs between 1 and 10 report no error and have an exit value of 0. Check that it handles bad input by reporting the error and exiting.

13. Notice that the exit value is 255, not -1. For now, just know that any non-zero exit value indicates an error, while zero means success. Why is the value 255 instead of -1? It is because exit values are 8-bit unsigned integers and under these conditions, a value of -1 looks like 255. We will talk about that in class.

14. Our program still does not do anything useful with valid input. Let's use the principles of **mañana** programming to get our program to do something without actually doing the work. Let's call a procedure to do all the work called DisplayStars. Add a call to this routine at the end of main() before the "return 0" statement:

```
DisplayStars(num);
```

15. Notice that NetBeans has underlined the function name DisplayStars. Why?

16. Trying compiling your project. Do you get an error? We seem to be missing something and we cannot wait until **mañana** to fix it! Add the following code after the "using" statement and before main() to create a procedure to display a line of stars. Refer to Chapter 4.3 "Programmer Defined Functions" if you need to review how functions are declared. For now, enter the following code after the "using" statement in main.cpp:

```
void DisplayStars(int numStars) {
}
```

17. Hit F6 to rebuild your project. It should now compile and run. DisplayStars does nothing (this is called a stub) but at least the program runs. Stubs are a good tool for starting a coding project without getting lost in all the implementation details. It lets you focus on the big picture today, and worry about the low level details tomorrow (**mañana** in Spanish).

18. When writing code, especially with stubs, remember to comment what the stub ***should*** be doing so you don't forget. Here is an example for DisplayStars:

```
// ****************************************************
// Procedure DisplayStars will up to 5 stars on a line
//
// Inputs:
//     numStars the number of stars to print.
//        Should be no more than 5.
// Returns:
//     Nothing
//
// ****************************************************
void DisplayStars(int numStars) {
   /* A Stub for now, later print numStars stars */
}
```

19. To complete the program, we need to add the C++ code required to implement DisplayStars. The code has a parameter "numStars" that specifies how many stars to print, so it will need a loop. Chapter 3.3 "More about C++ Loop Statements" describes how one creates loops in C++ if you need a review. For this example, we will create a FOR loop. Add the following code inside DisplayStars:

```
#define MAX_STARS 5

if (numStars > MAX_STARS) {
   cout << numStars << " stars is too many, ";
   cout << MAX_STARS << " is the maximum!" << endl;
   numStars = MAX_STARS;
}
for(int i=0; i<numStars; i++)
   {cout << "*";}
cout << endl;
```

20. Run your project and use various input values from 0 to 11 and see what happens. Does it reject 0 and 11 as expected? Does it print the correct number of stars for values 1 to 5? What happens with input values 6 through 10? Can you follow the Flow of Control through the program that makes this happen?

21. When the input value 10 is used, main() accepts this value but DisplayStars reports that it is too many stars and imposes a limit of only 5 stars by over-writing numStars and setting it to 5. We know that DisplayStars changes the value of numStars from 5 in this case, but does this also change the value of num in the calling procedure? To check, add the following statement to main() just after the call to DisplayStars:

```
cout << num << " stars requested." << endl;
```

22. Run your program with the input value of 10. How many stars are printed and was the value of num changed when DisplayStars over-wrote the value of numStars? We will talk about this in class.

23. Your program is now working but it is not complete. All C++ files submitted to CourSys for Cmpt135 should have a block comment which includes your name, student number, email, the course number, and lab week. It should also contain a brief overview of what the program does. Here is an example of what your block comment at the top of main.cpp should look something like:

```
// ************************************************
// ***
// *** Name  : Scott Kristjanson
// *** St.Num: 123456789
// *** email : skristja@sfu.ca
// *** Course: Cmpt135
// *** Lab   : wk02
// ***
// *** File  : main.cpp
// ***
// *** This program demonstrates using NetBeans to
// *** create a simple program that serves to review
// *** such C++ concepts as variables, parameters,
// *** conditional statements, loops, and functions.
// ***
// ************************************************
```

C. Lab Exercise Submission – To be completed by Students

Student are responsible for submitting the requested work files by the stated deadline for full marks. Since Lab Exercise solutions will be discussed in class following the submission deadline, late submissions will NOT be accepted. It is the student's responsibility to submit on time.

Students may work in teams of two and submit a single set of files on behalf of the group in Canvas.

24. You must submit your final version of file main.cpp before the deadline. For students working in the lab with a partner, only one submission is required for the group of two students. Instructions on how to submit your code will be sent out in a separate email to the class. Submit early to avoid problems.