LAB 01 Name: _____

Before you start these exercises, you should read the tutorial named l**ab01_tutorial.pdf**, located in the class directory,

/data/biocs/b/student.accounts/cs135_sw/tutorials

It explains everything you need to know to so this assignment. It is also on the course website http://www.compsci.hunter.cuny.edu/~sweiss/course_materials/csci135/csci135_36_fall12.php for your convenience.

Getting Around the File System

Let's make sure that you know how to get around the Unix/Linux file system before you start writing code. All of the exercises below are supposed to be done in the command line, unless otherwise stated. When I write a command, I always put the dollar sign **$** to represent the shell prompt. You do not type a dollar sign!!

The easy way to figure out "where you are" in the file system is to ask what your present working directory is: type the **pwd** command:

```
$ pwd
```

In the space below, write the output of that command.

Before continuing, try running the following commands to see what they output: try typing

```
$ who
```

and

```
$ who am I.
```

Now type the following *compound command*

```
$ date ; hostname
```

and write the output in the space below:

If you are not in your home directory, you can get there by simply typing **cd** followed by enter (with no additional arguments) – this will always bring you home.

Now, you will create several directories and files. Use the **mkdir** command to create the following

directories in your home directory:
cs135, **cs136**, **old_files**, **scratch**.

If you type **ls**, you should see the list of all the files in your home directory, including the directories that you just created. If you see a long list of files zooming by, then you probably need to do some cleanup in your directory: it is not the best idea to keep all of your files at the top level of your directory; after this class you should be able to organize your files better.
Change your PWD to **cs136** and create two more directories there: **labs** and **testing**. Within **labs**, create a subdirectory named **lab01**.

Change your current working directory to **lab01** and type

```
$ ls -l
```

to see its contents. Write its output below.

What single command will change your PWD to **testing**?

Make sure that your PWD is now **testing**. You can create an empty text file by typing **touch** followed by the file name. Enter

```
$ touch test1
```

and then use **ls** to see if a file named **test1** was created. You can edit the content of this file in many ways. **gedit** is one of the friendlier simple editors on Linux and I suggest you use that for editing your text files, including your C++ source code files. Type

```
$ gedit test1 &
```

and a window with an empty file should pop-up (the **&** at the end of the last command runs the command in the background, i.e. it allows you to continue using the terminal window while the **gedit** window is open – if you omit **&** your terminal will be blocked until you close the **gedit** window). You will see some output in the terminal such as

```
[1] 1276
```

Ignore it for now. Later I will explain it.

Enter some text into the body of the file (at this point it does not matter what text you type), save it and close it. This is exactly what you will do to write your code. After you close **gedit** and enter another command, you will see a message like you might see a message appear on the screen like

```
[1]+  Done            gedit test1
```

Ignore this as well. It says that **gedit** exited.
What command will remove the file **test1** from the directory **testing**? Write it in the space below and delete **test1** with it.

Now, create a new, empty text file called **test1** inside the directory **testing** again (yes, the one that I just told you to delete). Change your PWD to **lab01** (parent of **testing**) and try to remove the directory **testing** itself. What command should do this? What happens when you try? What do you need to do first to remove a directory? Write the answer in the space below.

You can now move back to your home directory and remove some other directories that you created before if you want, or you can leave them there.

Using the g++ Compiler

Let's make sure you can use the **g++** compiler. The class directory

```
/data/biocs/b/student.accounts/cs135_sw/cs136demos/lab01/
```

contains a C++ source file named **johnnybgoode.cp**p. Assuming that you read the tutorial and when doing so, you created a link to the class directory in your home directory that is called **cs135link** (if you did not, now is the time to do it), you can copy this file to the directory **cs136/labs/lab01/** in your home directory using the **cp** command:

```
$ cp ~/cs135link/cs136demos/lab01/johnnybgoode.cpp  ~/cs136/labs/lab01
```

(~ is another shortcut notation that stands for your home directory: whatever you type after is relative to your home directory.) There are other ways to copy the file, but this is relatively simple. Once the file is in your directory you can compile it, build executable code and run it (make sure that at the end of this lab you understand the difference between these three terms).

Change your PWD to the directory **cs136/labs/lab01** and make sure that the file is really there. A simple way to compile and build this program is to run

```
$ g++ -o johnnybgoode johnnybgoode.cpp
```

The **-o** option to **g++** tells **g++** that the string that follows it is what it should name the executable file that it creates for your program. If you omit this option, **g++** will use the default name which is **a.out**. Run **g++** again, but without specifying the name of the executable file. Look at the list of the files in your PWD, what do you see? Write it here.

To run the program type **./johnnybgoode** (it is the name of the executable file preceded by dot and forward slash) or **./a.out**. These programs are identical since they were created from the same source code.

Now edit the file **johnnybgoode.cpp** and save it as **johnnybgoode2.cpp**. Remove the line from the file (the line that contains **using namespace std;**), save and compile this new code. Write what happens when you run **g++**?

The compiler performs multiple steps when it is taking your source code and creating the executable program. Let's look at these steps:

1) ***Preprocessing***. Remember all those lines in the program that start with #? These are called ***preprocessor directives***. Their purpose is to insert the contents of the included file into the program at that point. When you type **#include <iostream>** in your code, the compiler replaces this line with the contents of the **iostream** library header file **iostream.h**. **g++** runs only the preprocessor[1] if you use **-E** option:

   ```
   g++ -E johnnybgoode.cpp > johnnybgoode.i
   ```

   By default the output is written to the screen; the bash redirection

   > **johnnybgoode.i**

   redirects the output to a file named **johnnybgoode.i**. You can use **gedit** to look at this file (it is much larger than the original **johnnybgoode.cpp**). Look at last sixty lines of this file. What do you find there? Describe what is there in the space below.

1 The preprocessor is actually a program named **cpp**. You can read about it by typing **man cpp**.

2) ***Compilation***. After preprocessing, the compiler actually reads the code and makes sure that it is syntactically correct. Compilation itself has a few sub-steps: parsing your code to validate it syntactically, creating assembly code, optimizing it, and then creating the executable code. **`g++`** stops after the creating the assembly code if you use **`-S`** option:

```
g++ -S johnnybgoode.cpp
```

By default the output is written to a file with **`.s`** suffix, so you should have a file called **`johnnybgoode.s`**. (Yes, it is a bit confusing that the output of **`g++ -E`** goes to the screen but the output of **`g++ -S`** goes to a file.) Look at this file. This is the assembler code for your program. The command **`wc`** (word count) followed by the name of the file tells you how many newlines, words and bytes the file has, in that order. Use this command to find out how many newlines are in your file **`johnnybgoode.s`** . How many lines does it have?

The second part of compilation is creation of the executable code of your program alone (this code will rarely be all that is needed to actually run the program). **`g++`** stops after this part if you use **`-c`** option:

```
g++ -c johnnybgoode.cpp
```

By default the output is written to a file with **`.o`** suffix, so you should have a file called **`johnnybgoode.o`**. This is not a text file anymore, so you cannot look at it with a text editor such as **`gedit`**. It is not an executable program yet; it does not have definitions for the symbols defined outside of itself such as **`cout`** and **`cin`**. They are in the **`iostream`** library. Files produced by this stage are called ***object code***, or ***object files***. The next step produces the final runnable program.

3) ***Linking***. At this point the object code for your source code exists, but the missing definitions need to be linked to it. These definitions usually come from the C++ libraries that the program uses; for now we are only using **`iostream`**, but soon we will use other ones as well. The last

step that is performed is called linking[2]. The command

```
g++ -o johnnybgoode  johnnybgoode.cpp
```

runs all the steps including the linking.

Display the list of all the files in your directory now:

Finally, write your own small C++ program: it should ask user for his or her age and then print a greeting using that information. The sample output of such program is:

```
Enter your age: 75

++++++++++++++++++++++++++++++++++
 Hi Stranger, You are 75 years old.
   Welcome to CSci 136, section 2
++++++++++++++++++++++++++++++++++
```

Name this file ***username*_welcome.cpp**, where ***username*** is replaced by your username on our system, i.e., the name with which you login. Compile and run your program.

How To Submit Your Work

The exercise has two parts, this document and your program. To submit this lab document, with everything you filled in, use Acrobat Reader's Print to File feature. Select Print, and in the dialog box, check the Print To File check-box. Then click the Browse button and name the file ***username*_lab01.ps**. where ***username*** is replaced by your username on our system.

2 The linker program used by g++ is called **ld**. You can read about it by typing **man ld**.

To submit both this file and the file ***username*_welcome.cpp** that you created above,

1. Create a directory in
   /data/biocs/b/student.accounts/cs135_sw/cs136labs/lab01/submissions
   whose name is your username. For example, I would create the directory **sweiss**.
2. Copy the two files, ***username*_lab01.ps** and ***username*_welcome.cpp**, to this directory.
3. Change your `PWD` to
   /data/biocs/b/student.accounts/cs135_sw/cs136labs/lab01/submissions
4. Execute the command
   $ chmod 700 *username*
   where ***username*** is the name of the directory you created. I would type "**chmod 700 sweiss**" for example. This prevents everyone else from reading your files.

You must do all of this by the end of the class on Thursday, August 30. If you do not complete all the work, you have until the end of the day on Friday, August 31, to receive full credit.