PHY480/905 Activities 1

*Online handouts:* Recommended C++ options; "Useful Unix Commands"; GSL intro

*Your goals for today (note: the goals are usually over-ambitious!):*

* Get used to the GNU/Linux environment (using Linux directly or via Cygwin or Windows Linux Subsystem (WLS) on Windows)
* Download, unzip, compile and run some simple C++ programs, then modify and repeat
* Try out Python
* Understand how underflow, overflow, and machine precision limits can be determined "experimentally"
* Try out a makefile: using g++ compiler warnings
* Try out a program that uses a function from the Gnu Scientific Library (GSL)

Please work in pairs with comparable computing and physics experience. The instructors will bounce around the room and answer questions (ask about anything!). *Fill out and hand in this sheet at the end of class.*

GNU-Unix Environment with Cygwin (Windows users)

1. Tie up any loose ends from the Cygwin [installation](https://people.nscl.msu.edu/~bogner/PHY480_905/doc/cygwin_install.php). *Ask questions if you are having trouble!*
2. Start up the Windows X-server by clicking XLaunch under the VcXsrv item in the Windows start menu. Accept all the default options.
3. Start up the Cygwin64 Terminal in the Windows start menu. Create a PHY480 subdirectory by typing "mkdir PHY480" at the command line.
4. Start up a web browser. Go to the course [webpage](https://people.nscl.msu.edu/~bogner/PHY480_905/index.php) and navigate down to the class schedule and click on the session01.zip link under Codes (choose "Open with Windows Explorer" and then select it and "Extract all files" to your PHY480 folder). You will also be able to unzip it from the Cygwin command line if you save it to your PHY480 folder. (NOTE: you could have used the "trick" outlined below for the WSL users of using symbolic links.)
5. Take a look at the unix command summary handout if you are unfamiliar with basic commands. Type pwd ("present working directory") to see where you are. Change to the PHY480 directory with cd PHY480 and then use cd to go to the session\_01 subdirectory. Use the tab key for "name completion" (try it!).
6. Can you figure out how to "see" your Windows directories inside the cygwin terminal? Once you figure this out, try (say) to copy a file from your Windows Documents folder to your working directory in Cygwin.

GNU-Unix Environment with Windows Subsystem for Linux (Windows users)

1. Tie up any loose ends from the WLS [installation](https://people.nscl.msu.edu/~bogner/PHY480_905/compphys_setup.php). *Ask questions if you are having trouble!*
2. Start up the Windows X-server by clicking XLaunch under the VcXsrv item in the Windows start menu. Accept all the default options.
3. Create a Windows PHY480 folder in Windows Explorer (e.g., I put mine in c:\Users\bogner\Documents\PHY480).
4. Start up a web browser. Go to the course [webpage](https://people.nscl.msu.edu/~bogner/PHY480_905/index.php) and navigate down to the class schedule and click on the session01.zip link under Codes (choose "Open with Windows Explorer" and then select it and "Extract all files" to your Windows PHY480 folder).
5. Start an Ubuntu terminal from the Windows start menu. At the prompt, type ln -s /mnt/c/Users/bogner/Documents/PHY480/ ./ This creates a symbolic link PHY480 in your Ubuntu home directory that points to the Windows PHY480 folder you set up previously. You should then be able to access the session 01 files from Ubuntu by typing cd PHY480/session\_01/ . There's probably a way to directly download files to the Ubuntu working directory as we did with Cygwin, but I lost patience after searching for 30 seconds and used this as an opportunity to introduce you to symbolic links ;).
6. Can you figure out how to "see" your Windows directory inside the Ubuntu terminal ? Once you figure this out, try (say) to copy a file from your Windows Documents folder to your working Ubuntu directory.

GNU-Unix Environment (If you have a Mac or a Linux laptop)

1. Your life is a bit easier ;).
2. Create a PHY480 working directory.
3. Start up a web browser. Go to the course [webpage](https://people.nscl.msu.edu/~bogner/PHY480_905/index.php) and navigate down to the class schedule and click on the session01.zip link under Codes. Save it to your PHY480 directory
4. Use unzip to extract the files.

A Simple C++ Program (look at the file first in an editor!)

1. *Make a note below of anything you don't understand about the code for the simple program area.cpp.* (If you are new to C++ or just rusty, you might not understand much; see the discussion in the background notes for some details.) Editor: use nedit or emacs or vi or ... (I suggest using nedit if you don't know any of these) to look at area.cpp. E.g., type nedit area.cpp &. (The "&" runs the editor in the "background" so the terminal is still available.) *Why use "area" and "radius" for variables rather than "A" and "R", which are quicker to type?*

**This is in order to be more specific**

1. Compile and link area.cpp to create area.x (or area.exe on Windows) using g++ -o area.x area.cpp (the executable would be called a.out if you left out "-o area.x"). Run it by typing "area.x" (or "./area.x" if you get an error message about not finding the file; why does this work?). Add "<< endl" to the end of the output line (starting "cout ...") and recompile and rerun. *What did this do?*

**It did not do anything different**

1. To get used to error messages, modify the program to create some errors and see what the compiler says (try these then invent your own):
   1. remove a semi-colon (leaving out a ; is the most common error you will make!)
   2. use // to comment out using namespace std;    [fix this by adding std::]
   3. remove a } (or change it from "}" to ")" or "]")

Each time, try compiling again. The compiler should give an error message with the line number it thinks is a problem and an explanation (which is often not helpful!). Fix the error and make sure the program still works. Note that "not declared" errors are reported as being at different lines than where the error is made. *Why aren't they noticed immediately?*

**In the editor, there were filter criteria set**

1. *List two ways to****verify****that the program is giving correct answers using known special cases and scaling arguments (e.g., multiply radius by factors of 10):*

**Get rid of filter criteria in editor and see if there are any red underlines in editor**  
  
  
  
Later (in PS#1) you will be asked to carry out some of the "to do" list in the comments (with assistance, as needed). *Which, if any, of them do you know how to do now?*

**Honestly,none**

1. Copy, rename, and modify area.cpp to make a program volume.cpp, which calculates the volume of a sphere using V = (4/3)πr3. Test it. *[Warning: Many people get this wrong the first time. Test carefully!]*
2. Let's try Python! Look at area0.py in the editor, and run it (python area0.py is one way). Compare with the C++ version. *Do they have the same pseudocode? List three differences between the C++ and Python implementations.*

* **Import in Python is Include in C++**
* **C++ uses “using”, Python does not need it**
* **C++ uses “<<” and “>>” as operators, Python does not**

Overflows, Underflows, and Machine Precision

Refer to the background notes on number representation as you do this section.

1. Look at the file for the code "flows.cpp". *Where does the output go?*

**Creates a file named “flows.out:**

A single-precision number in C++ is a "float" and a double-precision number in C++ is a "double". You are to empirically determine (in base 10, not powers of 2) *(put your answers in the blank spaces and compare to the notes)*

* 1. where underflow and overflow occur for single-precision floating-point numbers;  
       
     **Under: 0, Over: inf** (incorrect)
  2. where underflow and overflow occur for double-precision floating-point numbers.  
     **Under: e-45 , Over = e+38** (incorrect)

1. Now for the machine precision, which is calculated crudely in "precision.cpp". *[Note: This code has an intentional bug. Compare to flows.cpp to track it down.]* Determine empirically (in base 10)
   1. the machine precision for single-precision floating-point numbers;  
        
      **e-25** (incorrect)
   2. the machine precision for double-precision floating-point numbers. *[Note: you have to change the code for this to work correctly. Hint: the correct answer is smaller than 1e-12.]*

**e-08** (incorrect)

Explain briefly what the machine precision is and how it is found (note the output file):

**It is the digits after the decimal point and found by “e\*\*”** (incorrect— look in the ch1 notes for a formal definition of what it is. You should also be able to figure this out just from studying the code and the output.)

Using a Makefile

Look at the makefile "make\_area" in an editor. Makefiles are useful tools to organize the code for computational physics projects. They contain instructions for how to build an executable (what files have the code, what options to use in compiling, what libraries to link to). Here we have only one source (.cpp) file, but the makefile keeps track of all the g++ options.

1. Look at make\_area in an editor. We'll go through the details of a makefile later; for now note the following:
   * Comments start with "#" and continuation lines are indicated by "\" (so "SRCS= \" and "area.cpp" is all on one line). [Warning: There can't be any spaces after the "\".]
   * The list of options defined by "WARNFLAGS" are options to g++ (see C++ options handout for details).
2. Run the makefile using "make -f make\_area". It will try to compile area.cpp and then link to create area.x. *Did it work?* **Yes**
3. Make a copy of make\_area ("cp make\_area make\_flows") and edit it so that it compiles and links flows.cpp. [Follow the instructions in the makefile.] *Success?* **Yes**

Using the GSL Scientific Library

A link to the full GSL documentation can be found on the 6810 web page. Here we look at our first example program, which has been adapted from the documentation.

1. Examine the file "J0\_test.cpp" in an editor. This test program calculates the cylindrical Bessel function J0(x) at x=5 using the GSL library function "gsl\_sf\_bessel\_J0" (you would find out the name of the function by looking at the web page with the GSL reference manual). *Look up "Bessel function Wikipedia" using Google. How can you use a graph on the webpage to check you are calculating the right thing?*

**Check if the y-value at x=3 of the Bessel Function graph is the same as the code output**

1. Compile J0\_test.cpp *according to the directions in the file* and verify that the correct answer is given. *Is it?* **Yes**
2. Now calculate for x=3 (modify the code for this value or add code to read in any x). *Answer:* **-0.260051954901933446**
3. Verify your answer using Mathematica or MATLAB or Python. If Mathematica, under the Help menu, start the Help Browser, choose "Master Index", and look under Bessel function). If MATLAB, under the Help menu, select "MATLAB Help", choose "Search Results", and search for "Bessel Functions".  
   *Did you succeed?* **Yes, I got -0.260051954901933446 in both Python and C++ for J0(3)**
4. *Discuss with your partner how a more general program could be structured, and make at least two suggestions here.*

* **Use Boolean features**
* **Raise errors in code to detect mistake**

Some More Python (if time permits)

1. Create volume0.py.
2. Look at and run the Python versions of the flows and precision codes (included in session01.zip).
3. Try running Python interactively. At the prompt, just type python and return. You will get a >>> prompt. Try import area1 and then help(area1). Use quit() to exit. (On Linux you will need to type /usr/bin/python.)
4. Tkinter demo: Look at hello\_world\_gui.py and run it. Try to change the text and font.

Sharing Your Session Using GitHub

We're going to try GitHub to share files (also your homework with the instructors!).  
*\*\*\*\*\*\**

Grade: Check-minus. To upgrade to check-plus do the following

1. always include all your codes, output files, plots, etc. in the GitHub repo
2. correct answers in overflow, underflow, machine precision section