Lab 1 Activity Sheet

Purpose for Lab 1

To make sure you have the basic skills you will need through the rest of the semester.

By the end of this lab, you will be able to …

[1] Connect to the Engineering servers using PUTTY to perform basic functions in UNIX (`mkdir`, `cd`, `zip`) and to compile your C++ program.

[2] Do a multi-file compilation.

[3] Debug a program

[4] Learn how to use an FTP tool to transfer files from the Engineering servers to a Windows/Mac computer and from a Windows/Mac computer to the Engineering servers.

[5] Upload your work to Canvas.

You must have a School of Engineering Login and password for both the Windows environment and the UNIX environment. NOTE: the engineering login credentials are different from your SMU login. If you cannot login today, temporary IDs are available. Send an email to help@lyle.smu.edu to get your login information. The temporary IDs will be disabled sometime during the 2nd week of class.

Prepare Code Files [1]

1. In the lab, log into Windows using your SMU credentials.
2. If you have not already done the UNIX tutorials, do so now before going any further. Click on this link: **http://s2.smu.edu/~etchison/cse1342/unixTutorial.doc**
3. Log into one of the UNIX machines using PUTTY. Ask your TA for help with this step if needed. Enter genuse30.lyle.smu.edu for the host name and click login. Then enter your School of Engineering credentials.
4. At the Unix prompt, create a directory named `<lastName>Lab1` (`mkdir is the Unix command you will use`)
   E.g. `EtchisonLab1`
5. Move into this directory using the `cd` command.
6. You need to create the three files shown in Appendix 1 of this document. You may download them to windows by clicking on the link below and then ftp them to your new Unix directory using the `winscp` tool. The files can be obtained at
   http://s2.smu.edu/~etchison/cse1342/Lab1.zip

   They are stored in a '.zip' file (`Lab1.zip`). In your directory, you can extract the files from this archive by typing:
   `unzip Lab1.zip`. (See the Unix Cheat Sheet in Appendix B for more info.)

Multi-file Compilation [2]

7. In the past, you may have only compiled one source file for each of your programs. All the programs in this class will require you to compile more than one source file. **So you need to pay close attention to this step.**

8. At the UNIX prompt, type: `g++ TestClass.cpp MainDriver.cpp` Here are some things you need to notice about this:

   1. You don't include .h (header) files in the g++ compile line. The linker will handle this automatically. Only include files with .cpp extensions. You'll learn more about what goes in these different files later.

   2. If you want to compile all the .cpp files in the current directory (and they are all part of the same program), type: `g++ *.cpp` – try this with the files you created.

   3. Your executable file will be a.out. If you want your .out file to be called something else (which you will) you should use the –o flag on your g++ command.

      The following command will compile all the cpp files in the current directory and produce an output file called Lab1.out

      ```
      g++ *.cpp –o Lab1.out
      ```

   4. When there is an error in your program. You will need to open the .cpp file in a text editor. There are many you can use. A simple one is pico. Type the following line at your Unix prompt:

      pico TestClass.cpp

   5. Your program is opened in the text editor and you can now correct it. To save the changes, use the <ctrl> key and the letter O. To exit, press the <ctrl> key and the letter X.

Debug Your Program [3]

9. Here are some tips to detect syntax errors:

   - ☒ If it is a short program, scan through to detect syntax errors like missing semicolons. Sometimes the compiler will help you by giving you line numbers. But often an error on one line may be due to a mistake on a previous line, so scan the surrounding lines.
   - ☒ Solve one problem at a time (starting with the first error found by the compiler or several that you recognize and are easy. Recompile your program. Often one simple error at the top of your program will lead to many errors later on.
   - ☒ **Learn the messages your compiler returns for different errors.** Each message means something – you will probably encounter the same messages over and over again throughout the semester.
   - ☒ A common error is when the compiler does not recognize a variable or datatype you use in your program. Ask yourself, "Where do I tell the compiler about this variable?" Go back to that declaration, prototype declaration or #include and look for syntax errors.

- ☒ Try to isolate the error by commenting out portions of code and recompiling. If you get no errors or **get a different error**, go back and look for syntax errors in the code you just commented out.

10. Here are some tips to logically mistakes(bugs in your solution):
    - ☒ The most powerful tool for detecting bugs is the `cout` statement. Temporarily insert cout statements to test the value of variables. Calculate by hand what you think the variables should be.
    - ☒ Sometimes, a block of the code (e.g a loop) never gets executed because the conditions for entering the block are never met. Use an appropriate cout statement that says something like "I got to the xxx function" to verify that certain parts of code are being executed.
    - ☒ '==' is different from '='. Often the compiler will not pick this up, but your program will not work correctly.
    - ☒ **Make sure the program does what it is supposed to do**. This may seem trivial, but companies have lost millions of dollars because of this bug.
    - ☒ Your user interface is important. Technically, a user interface that does not make sense to the user, or that has spelling mistakes is a bug. Use newline characters to make your interface presentable. Mistakes in the documentation are also bugs because it can potentially mislead someone who uses your code later.
    - ☒ Sometimes the compiler issues a warning but compiles successfully. Usually, you should pay attention to these warnings and try to find the root cause. This may require digging into reference books!
    - ☒ Above all, do not lose hope! No error is impossible to detect. No bug is impossible to find. It may take a while but keep using some of the steps above and you are certain to find the misdeed!

Upload Your Program [4]

11. If you are satisfied that everything is in working order, you need to compress your files into one file before you transfer them to the windows environment. (You can alternatively compress your files in the windows environment if you know how to use WinZip or some other compression utility). Follow these steps to compress in UNIX:
    a. Move up one directory (`cd ..`)
    b. Type:
    `zip -r <Lastname><firstInit>Lab1.zip <directory>`
    e.g. `zip -r EtchisonJLab1.zip EtchisonLab1`
    (See the course outline for specifics on the naming convention that is required for your zip files for regular assignments)
12. Using WinScp, transfer the compressed file to your Windows Machine.

13. Upload that compressed file to Canvas.

Appendix 1: The 1342 Guess the number game (3 source files)

```
/*************************************************************
 * MainDriver.cpp
 *
 * This file is used for CSE 1342 - Lab 1. It will use the
 * TestClass class to create an object that
 * can play "guess a number" game
 *
 *************************************************************/

#include <iostream>
#include "TestClass.h"
using namespace std;

int main (void) {
       // create an object of type TestClass
       TestClass anObject;
       bool flag = false;
       while (flag == false)
       {
              anObject.getguess();
              flag = anObject.checkgues()
       }
    cout >> "Thanks for playing \n << endl;
    return 0;}



/*************************************************************
 *
 * TestClass.h
 *
 * This file contains the interface for a TestClass that
 * plays a "guess the number" game.
 * The implementation is found in TestClass.cpp.
 *
 *************************************************************/



#ifndef Something
#define Something
class TestClass
{  private:
     int guess;
   public:
        void getguess();
        bool checkguess();
};
#endif
```

```
/*********************************************************
 * TestClass.cpp
 *
 * This file contains the implementation of the guessing game
 *
 *********************************************************/


#include "TestClass.h"
#include <iostream>
using namespace std;
/*************************************************************
 * This is the implementation for the TestClass
 *
 *************************************************************/

void TestClass::getguess()
{

	cout << "I have a number between 1 and 100" << endl;
	cout << "Can you guess the number; Type your guess";
	cin >> guess;
}
bool TestClass::checkguess()
{
	if (guess == 56)
	{
		cout << "You are right" << endl;
		return true;
	}
	else
	{
		cout << "you are wrong" << endl;
		if (guess < 56)
			cout << "Your number is too low, try again\n";
		else
			cout << "Your number is too high, try again\n";
		return false;
	}
```

}Appendix 2: Unix Cheat Sheet

These are the most common UNIX commands – you should be familiar with them.

`tcsh`	This command will change the shell that you are currently running. If you execute this command, you should be able to use the backspace key with most of the servers.
`g++`	This is the GNU c++ compiler
`dos2unix <fname>`	Removes ^M that might appear in a document that you copy over from DOS (a MS Windows machine) to one of the UNIX servers.
`lpr -P<prnt> <fname>`	Prints from UNIX on the School of Engineering Machines. ☒ `<prnt>` is the name of the printer. E.g. in Patterson 214, it is `pathp`. In the open lab in SIC, the printer name is `sic_si`. ☒ <fname> - the name(s) of the file(s) that you care to print. Don't forget to include the file extension. You can list more than one file for printing.

Directory and File Manipulation Commands

`ls`	Lists the contents of the current directory.
`mkdir <newDirectory>`	Create a subdirectory of the directory that you are currently in. It is highly recommended that you put all the files pertaining to one program or assignment in its own directory. This will make your life much easier when it comes to compiling programs that have many files.
`cp <source> <dest.>`	Copy a file from <source> to <dest.>
`mv <source> <dest.>`	Move a file from <source> to <dest.>
`rm <filename>`	Delete a file
`rmdir <directory>`	Delete a directory
`pwd`	Display the path of the directory you are currently in.
`man <command>`	Shows the UNIX manual page for `<command>`
`less <filename>`	Prints the file to the screen and allows you to move up and down to view it. NO EDITING ALLOWED here.
`zip -r <directory>`	Archive a directory as a zip file.
`unzip <zipfile>`	extract the files in an archive into the current directory
`quota`	This command will tell you how much of your disk quota you have used and how much you have remaining.
`pquota`	This will tell you how much print quota you have remaining.

Editors

TELNET & X Windows Session (Exceed session)	
`pico`	A simple single file text editor.
`vi`	A more robust editor (see the VI tutorial here or on Canvas if you want to know more or you want a review of how to use VI).
X Windows Session only	
`emacs`	A robust editor. It allows the user to have multiple files open for editing at the same time. It is possible to use it in a TELNET session, but this is much harder.
`xcoral`	Another editor; more graphics-based; not available on all servers.