**Lab 1: Review of Emacs and Unix**

# Overview

In this lab you will:

1. Review how labs are graded
2. Review basic Emacs and Unix commands
3. Create and compile a simple C++ program
4. Create a makefile
5. Submitting your work

# Basic Unix Commands

There are several flavors of Unix: IRIX, Solaris, Linux, etc. For this class, your programs must compile and run on Linux. When you log onto GL, be sure that you connect to the host named gl.umbc.edu or linux.gl.umbc.edu. Connecting to these host names ensures that you are actually working on one of the three Linux servers,linux1.gl.umbc.edu, linux2.gl.umbc.edu, or linux3.gl.umbc.edu; which of the three Linux servers you are connected to depends on the current load for each machine.

### Simple UNIX Commands

**ls**

Lists the files in the current directory

ls -l gives more information about files. -l stands for "long" output.

**cp**

Copies a file

cp sample.cpp example.cpp makes a copy of sample.cpp and names the new copy example.cpp.  
The file sample.cpp is unchanged.

**mv**

Renames (“moves”) a file

mv average.cpp mean.cpp changes the name of the file from average.cpp to mean.cpp.  
The file average.cpp no longer exists.

**rm**

Removes or deletes a file

rm olddata.dat deletes the file olddata.dat.

**more**

Displays the contents of a file on the screen one page at a time

more example.txt shows the contents of the file example.txt one screenful at a time.  
You must press the spacebar to advance to the next page.  
You may type q to quit or b to go back to the beginning of the file.

**mkdir**

Makes a new subdirectory in the current directory.

mkdir cmsc202 will create a new directory called cmsc202 in the current directory.

**rmdir**

Removes an empty subdirectory from the current directory

You must delete all of the files from a subdirectory before you are allowed to delete it with this command.

**cd**

Change the working directory

The command cd with no arguments will return you to your home directory.  
cd followed by a subdirectory name found in the current directory, such as cd cmsc202,  
will change from the current directory to the subdirectory called cmsc202, if that subdirectory exists.  
cd .. changes to the directory one level up in the directory tree.

**man**

Gives a description of a UNIX command.

man cat will describe the cat command.  
If you don't know the name of a command, but you do know what you want to do, use man -k.  
For example, if you've forgotten the command for copy, type man -k copy and you wil be shown  
a list of commands with the word "copy" in their description.  
The man command uses the more command to present the information, so you must press the  
spacebar to advance to the next page. You may type q to quit or b to go back to the beginning of the file.

## Starting Emacs

Emacs is flexible and widely-used text editor which is supported on the GL systems. To start the text editor, type the command emacs at the Linux command prompt.

You may also specify a file to open or create on the command line. For example, the command emacs example.cpp starts Emacs and loads the file example.cpp, if it exists; if the file does not exist, it will be created when the buffer is saved (e.g. with the Emacs command C-x C-s; see below).

## Emacs Commands

Emacs commands are entered using key combinations that include the Control or Escape keys. A quick reference to major commands is provided below. The letter Crepresents the Control key; the command C-n means "hold the Control key and pressn." The letter M stands for "Meta," which is the Escape key on most systems. For example, the command M-a means "press the Escape key, then press the letter a." Donot hold down the Escape key for Meta commands.

### File Operations

C-x C-f Retrieve or open a file

C-x C-s Save the current file and continue editing

C-x C-c Save the current file and exit

C-x C-w Write the buffer contents to a file

### Cursor Movement

C-n Move to the next line

C-p Move to the previous line

C-b Move backward one character

C-f Move forward one character

C-u n C-f Move forward `n' characters (n should be a number)

C-a Move to the begining of the line

C-e Move to the end of the line

M-f Move forward one word

M-b Move backward one word

M-a Move to the beginning of the sentence

M-e Move to the end of the sentence

C-v Move forward one screenful

M-v Move backward one screenful

M-< Move to the begining of the file

M-> Move to the end of the file

M-g Goto line (prompted for line number)

### Editing

C-d Delete one character

C-k Kill (cut) from the cursor position to end of line

M-k Kill (cut) to the end of the current sentence

M-d Kill (cut) the next word after the cursor

C-y Yank (paste) the last killed text back

### Miscellaneous

C-g Cancel current command

C-l Clear screen and redisplay everything, centering the screen

on the current cursor position

C-x u Undoes one command's worth of changes

# Creating a C++ Program

You are going to create your first C++ program. At the command line type emacs proj0.cpp to start Emacs. The file proj0.cpp should not exist yet, so it will be created the first time you save your file.

The program must read two integers from the keyboard (standard input), sum them, and write the sum to the screen (standard output).

Here is how the program should look when it is run:

linux2[1]% ./proj0

Enter the first number: 17

Enter the second number: 23

The sum is 40

linux2[2]%

In this example, the user provided the input (17 and 23); the program printed the prompts, computed the sum, and output the sum to the screen. Note that linux2[1]%is the Linux shell prompt, not part of the program.

Here is boilerplate code for a basic C++ program:

#include <iostream>

using namespace std;

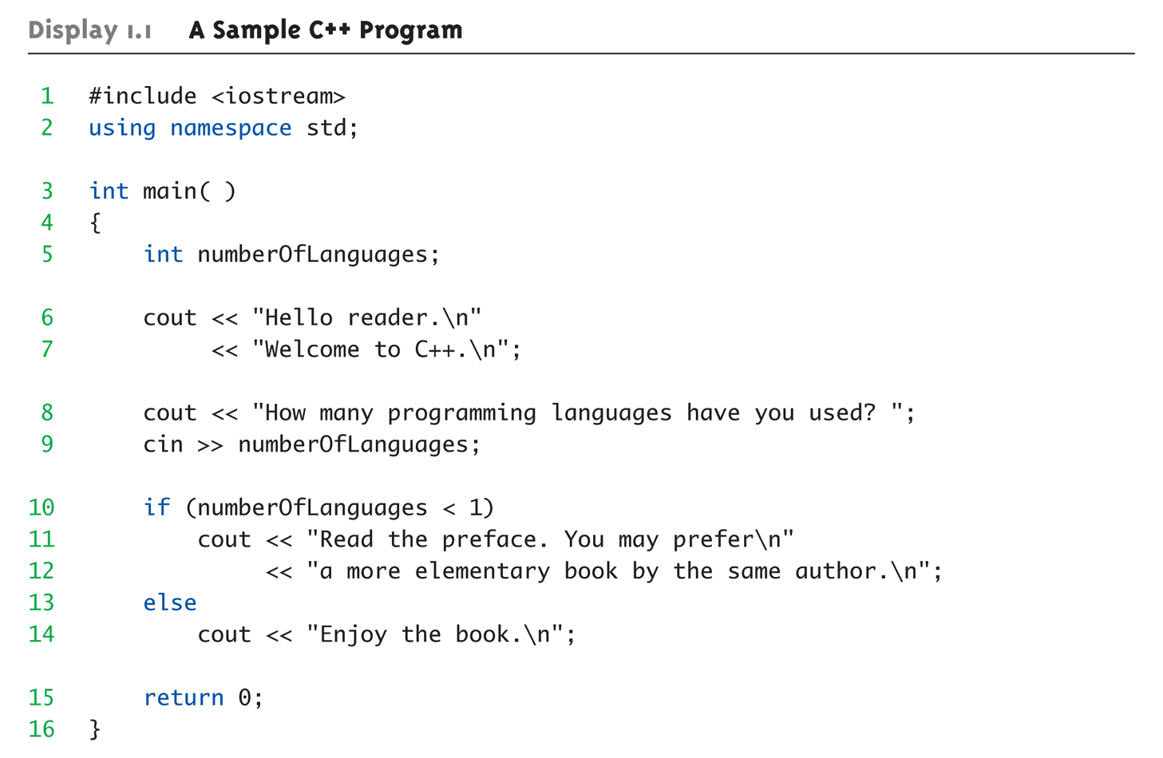
int main() {

// Insert your code here

return 0;

}

You will need to enter these lines and add code to complete the given task. The following code sample demonstrates how to declare an integer variable, read a number from standard input, and write to standard output:



The sample code does not demonstrate how to include the value of a variable in output, but this is easy to do. Suppose in the sample I had wanted to print a special message to any user knowing five or more languages. Here's a code snippet that demonstrates how to do it:

if (numberofLanguages > 4)

cout << "Wow! " << numberOfLanguages << " is a lot of languages.\n"

If the user provides the input value "7", this snippet would produce the message "Wow! 7 is a lot of languages."

Once you have typed your code in Emacs, be sure to save (C-x C-s) or save and quit (C-x C-c).

## Compiling and Running

To compile your program, enter the following command at the Linux prompt:

g++ -Wall proj0.cpp -o proj0

This command runs the GNU C++ compiler (g++). The option -Wall instructs the compiler to be verbose in its production of warning messages; the option -o proj0 (hyphen followed by the letter "o", not the digit zero), instructs the compiler to give the executable program the name proj0. If the program compiles corectly, the executable file proj0 will be created in the current directory.

At the Linux prompt, enter the command ./proj0 to run your program. It should look like the sample output provided above (small variations in prompt and output text are okay).

Makefiles  
The Unix make utility can be used to automate the compilation and linking of programs. It may be difficult to see how this could be useful at this point, but you will appreciatemake's functionality when you work on complex, multi-file projects later in the semester. For now, we will learn how to use make for simple program builds.

make requires that the programmer provide instructions to build the program in a text file called a *makefile*. The makefile file will typically named Makefile with a capital "M"; the make program looks for this file by default.

A makefile consists of multiple sections, called *rules*, that specify how the program is built. Each rule consists of a header line, listing a specific file to build (called the *target file*) and the files the target depends upon (the *prerequisites*), followed by zero or more lines specifying the commands necessary to construct the target file (the *build actions*or *recipe*). make compares the last-modified timestamp of the target file with those of the prerequisites it depends on, and if any of the prerequisites is newer, it will execute the recipe to bring the target up-to-date.

Without further ado, here are the contents of a very simple makefile:

program: program.cpp

g++ -Wall program.cpp -o program

In this example, program is the target, program.cpp is the only prerequisite, and the recipe is g++ -Wall program.cpp -o program. When make is run using this Makefile, it will check to see if program.cpp has been modified more recently thanprogram and, if so, will execute the recipe to recompile the program.

**Note:** the whitespace at the beginning of a recipe *must* be a tab. Emacs will insert a tab correctly so long as your makefile is named Makefile – Emacs knows not to replace tabs with spaces when working on makefiles.

Now, create a makefile (named Makefile) to build your C++ program proj0.cpp. Create the makefile in Emacs, entering the lines from the sample, modified appropriately. Once you have created the makefile, save and quit Emacs. Test the makefile by entering the following commands at the Linux prompt:

touch proj0.cpp

make

The touch command changes the last-modified time of proj0.cpp to the current time, ensuring that proj0.cpp is "newer" than the executable proj0; thus the make command will execute the recipe to rebuild the executable.