Introductory Concepts and Tools

Computer Science 240

Laboratory 1

In CS 240 this semester, you will be using programming languages, operating system, editor, and project management applications that you may not be familiar with. This first lab is intended to introduce you to these, which you will then continue to use throughout the semester to help you investigate and learn about computer systems and how computers work. In lab today, you will complete tutorials on the following:

1. Linux (operating system)
2. Emacs (editor for creating programs)
3. C (programming language)
4. Bitbucket and Mercurial (source control applications to manage and share your work).

At this point in your career, you have a working knowledge of at least one high-level programming language, such as Java or Python. However, as a computer scientist, it is quite useful to understand how the code you write is translated to and executed by the underlying computer hardware, because software design decisions made at the high level can significantly affect the performance of the program when it is run.

The C programming language was created specifically to write the UNIX operating system (an operating system is the software which manages and controls all other applications and resources on a computer). C was designed for high level programming, but also to provide access to the hardware level. So, you can (much more easily than Java or Python, for example) examine the direct impact of your program on the hardware. You will be writing some programs in C this semester for that purpose, in the Linux environment.

The computers we have available in lab are dual-bootable to either Linux or Windows, and we will use Linux in the first lab. Linux is a version of the UNIX operating system, and includes a graphical user interface, X Windows system, the Emacs editor, and C compiler.

Some of your work will be done by entering UNIX commands directly at a prompt in a terminal window, rather than selecting options through a graphical interface (which is how you usually have used a computer). A **terminal** is a session that can receive and send input and output for command-line programs, and a **shell** is a program that is used for controlling and running programs interactively via a terminal (sometimes the two terms are used interchangeably).

Since the underlying operating system on Macs is UNIX, you will be able to open a terminal on your Mac and have a very similar operating environment to what we will have on the lab computers. If your own computer is a PC, you can open a remote connection to a Linux machine for a similar environment. You can always also use the lab machines for assignments and outside work.

Linux

Exercise 1: As a laboratory assignment today, you were asked to complete Module 1 and Module 2 of the Linux Survival Tutorial, available at the following URL, which introduced you to the directory structure and command-line entry that you use in Linux:

http://linuxsurvival.com

Now, try some of these commands on a real machine. Log in to your lab machine (which should be booted to Linux, not Windows) using your Domain password, and open a Terminal from the **Applications->System Tools** menu. You will see a window with a command prompt (similar to what you saw in the tutorial).

For each of the following steps, record **both** the command used and the answer to the stated questions:

1. What is your working directory when you initially log in to the system (the directory you are currently in)?

2. List the files and subdirectories in your working directory.

3. How do you know whether the items listed are files or subdirectories?

4. Are there any files which begin with a '.'? If you are not sure how to list all files, what command can you use to get help in figuring out how to do it?

5. Display the contents of the **.bash_profile** file in the Terminal window (do not use an editor to do this):

6. Make a new directory called **cs240**.

7. Change your working directory to **cs240**.

8. Echo your name to the screen (if you are not sure how to do this, either Google or use the **man** command to figure out how to do it).

9. Echo your name again, but this time redirect the echo into a new file named **test1** (again, if you're not sure how to do this, try to figure it out).

10. Find out whether the new file was created or not. Then, display its contents to the screen (without an editor).

11. Do the same thing as step 9, but echo your partner's name to a second file called **test2**.

12. Concatenate the two files and put them into another file called **test3**.

13. Display the difference between **test1** and **test2.**

14. Search for your name in **test3** (if you're not sure how to do this, try to figure it out).

15. Copy **test3** to a new file **test3.c** (do not use an editor and do not remove **test3**).

16. Change the name of **test3** to **test4** by using a Linux command (do not use an editor). The **test3** should no longer be listed.

17. Remove **test4**.

18. Change your working directory back to its initial location (what are two different ways to do this?).

19. Display the contents of your **test3.c** file without changing to the **cs240** subdirectory.

20. Change back to the **cs240** directory.

Emacs

Exercise 2: When you are in the Linux environment, the **Emacs** editor is commonly used to create and edit files. A little later in lab today, you will explore the features of Emacs in more detail, but for now, try some very simple editing and save the result.

In the steps below, the '$' is used to indicate the command prompt.

1. Open your **test3.c** file using **emacs** from the command line:

    $ **emacs test3.c**

2. The .c suffix denotes a file containing a C program. Make the text in your file into a C comment by enclosing with the comment delimiters (assuming Jean and Ben are the partner names), and add a main function.

```
/* Jean
 Ben */

int main( ) {
   return 0;
}
```

3. Save the change by entering a **CNTRL-x**, and then a **CNTRL-s** (hold down the control key while entering the specified character). You should see a a message at the bottom of the screen :

    Wrote /home/yourbane/cs240/test3.c

4. Exit Emacs by entering a **CNTRL-x**, and then a **CNTRL-c.**

You have now created a valid C program (albeit one that does not actually do anything)! You will practice doing more later in lab today.

5. It is also desirable to open Emacs as a background process, so that your Terminal is available for other commands while you are editing. You do this by simply invoking the command followed by an ampersand:

    $ **emacs test3.c &**

which pops up a separate window where you can do your editing, leaving your original Terminal available for entering other commands (such as compiling your newly edited and saved file).

NOTE: it is also possible to use X Windows to open Emacs from the top menu (look in **Applications->Accessories).** You can then use a graphical interface to do your editing. However, it is much slower than learning the key sequences for commands. If you spend the time becoming proficient at using the command-level emacs, it will be a valuable skill for future courses and possibly jobs, as well.

Compiling

Exercise 3: In the Linux environment, the GNU C (**gcc**) compiler is used to compile C programs.

1. Enter the following command to compile the **test3.c** program. The **–g** option produces debugging information, which we will soon learn more about). The **–o filename** option puts the executable result of the compile into the specified filename :

    $ **gcc –g –o test3 test3.c**

2. Execute the program by doing the following (you will not see anything happen, but that is okay for now):

    $ **./test3**

NOTE: You can also compile by using a **Makefile**, basically an automated recipe for compilation. Your assignments

will provide Makefiles that do exactly the right tasks for compilation, plus instructions on how to use them.

Although you may not be familiar with the C programming language, you could probably grasp the meaning of the simple program you just wrote. The basic syntax of C is quite similar to Java and Python, with some key differences. Here are the differences you need to know about immediately:

- C is an imperative language: there are no objects (Java is, and Python can be, object-oriented)
- Everything in C is written as a function (called methods in Java, but Python also uses functions)
- The program begins execution at function **main** (similar to Java, unlike Python).
- C does not have a **boolean** data type (unlike Java and Python). So, you represent a boolean with an an integer, where a value of 0 is interpreted as false and a non-zero value is interpreted as true.

In the next exercise, you are given a simple C program for Hello World, and you are then asked in exercise 6 to write some simple programs of your own.

Bitbucket/Mercurial

Exercise 4: This semester, we will be using Mercurial and Bitbucket to manage all course materials and student work. You should have created your Bitbucket account by now, but if neither you nor your partner has done so, follow those steps before proceeding.

Go to the tutorial at the link below and follow all the steps through the section on Mercurial Solo Basics (the last section on Team Basics will be relevant later when you begin working on team projects for class).

https://cs.wellesley.edu/~cs240/common/hg/hg.html

At the end of lab today:
Submit results by saving this file as your lab report, with your files from the previous Linux section **(test1, etc.)** and also any C files you create in the final C section of lab today (**program1.c,program2.c,**etc.) to your **cs240-lab1** repository. Then, push your local **cs240-lab1** back to your Bitbucket fork.

More on Emacs

Exercise 5: Open Emacs from the **Applications->Accessories** menu, and run through the Emacs tutorial that you find there.

More on C

Exercise 6: Refer to the following guide as needed to write and test some short C programs :

http://www.tutorialspoint.com/cprogramming/c_quick_guide.htm

Save the programs as **program1.c, program2.c, program3.c,** etc..

Program 1: Add two integers entered by the user and display the sum

```
Sample output
Enter two integers: 12
11
Sum: 23
```

Program 2: Find the size of int, float, double, and char of your system.

```
Sample output
Size of int: 4 bytes
Size of float: 4 bytes
Size of double: 8 bytes
Size of char: 1 byte
```

Program 3: Count and display the number of digits in an integer entered by the user.

```
Sample output
Enter an integer: 34523
Number of digits: 5
```

Program 4: Check whether a number entered by the user is prime or not.

```
Sample output
Enter a positive integer: 29
29 is a prime number.
```

Program 5: Find the highest common factor of two integers entered by user.

```
Sample output
Enter two integers: 14
35
HCF of 14 and 35 is 7
```