Lab 1: The Ropes

Due date: Friday, September 26, 11:59pm.

Lab Assignment

Assignment Preparation

Lab type. This is an **individual lab**. Each student will submit his/her set of deliverables.

Collaboration. Students are allowed to consult their peers[1] in completing the lab. Any other collaboration activities will violate the *non-collaboration* agreement.

Purpose. The purpose of this lab is to get everyone started. You will be introduced to the computing environment used by the CSC/CPE/SE programs, will perform activities designed to give you some acquaintance with Linux, will peruse the course web page, compile a small C program and submit the results of your activities to the instructor using the course submission program handin.

The Task

Note: Please consult the instructor if any of the steps are unclear, or if the reality does not match the instructions below.

1. Login. Sit in front of the computer in the lab. You shlould see the Linux login screen. All department of Computer Science workstations (CSL

[1] A peer for the purpose of CPE 101 is defined as *"student taking the same section of CPE 101"*.

machines) come with the same installation of Linux. Some machines, including the ones in our lab, 14-302, are dual-boot, and allow one to boot into Windows XP, but **in CPE 101 we will always be using Linux**. If your machine is displaying Windows XP login, it needs to be rebooted.

You all should receive a CSC loginId, which allows you access to CSL machines. The loginId is *typically* (although not always) the same as your CalPoly loginId. **However**, your CSL account is **different** from your CalPoly account. In particular, it has a different password.

If you never logged onto your CSC account, your password will be the default password set for you by the system administrators. This password needs to be replaced. While password change is not part of the lab (because some of you might have already done so), instructions for doing so are provided at the end of the lab.

Assuming you are looking at the Linux prompt, **perform the following actions**.

1. From the pull-down list of possible login options, select "Other..." and click on it.
2. At the Username prompt that appears, enter your CSL loginId (username).
3. At the Password prompt enter your current password.
4. Wait until the login is complete. As a result of your login, you should see a desktop with three — four icons on it, a system menu at the top, and a status bar at the bottom. The environment you are in is generally called XWindows. Congratulations, you are in!

2. Set Up CPE 101 directory. Your next task is to prepare a working directory for your CPE 101 coursework.

Perform the following tasks.

1. In the system menu at the top of the desktop, select Applications ⟶ System Tools ⟶ Terminal. A command window will open. The window will give you a command line in the *home directory of your account*. It will also have a menu, which you can explore later.
2. In the command window, at the prompt type

   ```
   > ls
   ```

   and hit `<Enter>`. All Linux commands are finished by hitting the `<Enter>` key, so we will omit mentioning this in the future.

   Note: The ">" symbol in the command about represents the Linux command-line *prompt* and **should not be typed by you!**. The actual Linux prompt is customizable and may look differently on different machines. For example, Linux prompt on my office machine looks like this:

```
8:07pm londo ~$
```

From now on, ">" will indicate any Linux prompt in your terminal window.

3. In the command window, at the prompt type

   ```
   > ls -al
   ```

   The `ls` command lists the contents of the current directory (it should be your home directory here). First, you asked for a brief listing, and received very little information - only the names of the files and subdirectories in the current directory. (In fact, it is quite possible that if this is your first login, `ls` command will return nothing. The ls -al command asks Linux to display everything that resides in the current directory and display detailed information about each item. Comparing the results of the two commands, you can see, that there are some "hidden" files in your home directory: their names did not show up after the first ls command was given, but did show up after the second command. The names of these files start with a ".", i.e., these files DO NOT have a file name proper, only an extension! These are the so-called *"dot-files"*. Typically they contain some setup and application-specific information and help various applications including XWindows to run properly on your system.

4. Create a directory for CPE 101 coursework. You can choose any name you want for this directory. I will use `cpe101` as the default name. The command is:

   ```
   > mkdir cpe101
   ```

   To verify that your directory has been created, type `ls` again.

5. Change to the CPE 101 directory:

   ```
   > cd cpe101
   ```

6. Create a new directory for Lab 1 files, and change to that directory.

You are now ready for the real work!

3. Get the files! Your next task is to put a few files into your Lab 1 directory. Perform the following tasks.

- Start a web browser. Firefox is the most popular browser in the Linux world. You can start it by typing

  ```
  > firefox&
  ```

  in the terminal window (*notice the "&", it will let you continue using your terminal window for further commands!*), by clicking on the web browser icon on the system menu, or by selecting Applications ⟶ Internet ⟶ Firefox Web Browser from the system menu.

- Download the web page for the course. The url is

  http://www.csc.calpoly.edu/ dekhtyar/101-Fall2008/

- On the course web page locate Lab data section. You should see two files available there for Lab 1, `hello.c` and `foo.c`. Save both of these files in your Lab 1 directory.

- You need to get one more file, but this time, you will be using Linux `cp` command to copy the file to your Lab 1 directory. The file name is `aboutme.alex` and the file is located in the ∼`dekhtyar/www` directory. Use the following command to copy it:

  ```
  > cp ~dekhtyar/www/aboutme.alex .
  ```

  Note: "." in this command refers to current directory. The command reads *"Copy `aboutme.alex` file from the ∼`dekhtyar/www/` directory into the current directory.*

- Type `ls` to confirm that all three files are now located in the your Lab 1 directory.

- Let us now make sure you can peruse both PDF and Postscript files. Find the handout for Lab 1 on the course web page. You will be offered two download formats: PDF and Postscript. Select download in PDF format. Select `"Open with ...Document Viewer"`. The application that opens your PDF file is called `evince`.

- Now, download the file in Postscript format. In the dialog window, select "Save to disk" and save the file to your `cpe101` directory.

- Change to `cpe101` directory. Assuming you are currently in the Lab 1 directory, the appropriate command is

  ```
  > cd ..
  ```

  An alternative method is

```
> cd ~/cpe101
```

Here, `..` is used to indicate the parent directory. ∼ indicates your some directory. Also, a pair of commands:

```
> cd
> cd cpe101
```

will achieve the same effect.

- Use the `ls` command to make sure that your directory contains the Postscript file you downloaded. Type

  ```
  > gv <FileName>
  ```

  replacing `<FileName>` with the name of the downloaded PostScript file. The program that opens the file, `gv`, of `GhostView` is a PostScript file viewer. Make sure you can navigate the document (move from one page to another), and study other features of `gv` that are immediately available to you. Exit `gv`.

- Now, type

  ```
  > evince <FileName>
  ```

  again, replacing `<FileName>` with the name of the downloaded PostScript file. Note, that `evince` can be used to view PostScript files as well as PDF files.

- Change back to your Lab 1 directory.

You are now ready to view/peruse all course materials.

4. Compile and run the programs. Time to put the files you downloaded to use.

- Let's view the contents of each file. The command that prints the contents of a file to a terminal is `more`. The format of the command is

  ```
  more <filename>
  ```

  where `<filename>` is the name of the file you want to view. Using the `more` command, output the contents of `hello.c` and then - `foo.c` to the terminal.

- Study the contents of the `hello.c` file. This is the famous Hello, World! program written in C.

- Let us compile this program. You will be using GNU C++ compiler `gcc` to compile your C programs in this course. `gcc` recognizes pure C programs and compiles them accordingly. Type

  ```
  > gcc hello.c
  ```

- Nothing happened. You get a new Linux command-line prompt. What gives? Well, let's see if anything has changed in your Lab 1 directory. Type

  ```
  > ls -al
  ```

  If you did everything correctly, the output will look (roughly) as follows:

  ```
  total 15
  drwx------ 2 dekhtyar Faculty_Users  512 2008-09-22 20:53 .
  drwx------ 3 dekhtyar Faculty_Users  512 2008-09-22 20:52 ..
  -rwx------ 1 dekhtyar Faculty_Users 5424 2008-09-22 20:53 aboutme.alex
  -rwx------ 1 dekhtyar Faculty_Users 4803 2008-09-22 20:51 a.out
  -rw------- 1 dekhtyar Faculty_Users  625 2008-09-22 20:34 foo.c
  -rw------- 1 dekhtyar Faculty_Users  364 2008-09-22 14:15 hello.c
  ```

  There is a new file, `a.out`, which was not there before. And (if your terminal if set up right, this file name will be `green`). This is the result of compiling `hello.c` file. (It's green because this is how Linux terminal marks executable files).

- Let's run this program. Type

  ```
  > a.out
  ```

  What's the result? Congratulations, you just ran your first C program (albeit, you did not write it).

- Let us now examine the second program. Use `more` command to view its text again. What do you think it does? Try compiling it, the way you compiled `hello.c`. What is the result?

- We will be compiling `foo.c` using a more complex command. In particular, this command will ensure that `gcc` does all of the following things:

  – uses ANSI C standard for syntax checking (`-ansi` flag);

  – reports all compilation errors and warnings (`-Wall` flag);

  – compiles using *strict* rules - all warnings are treated as errors (`-Werror` flag);

  – tells `gcc` to actually correctly process the `sqrt()` reference (`-lm` flag);

- changes the name of the output executable file from the default `a.out` (`-o` option);

Type

```
> gcc -ansi -Wall -Werror -lm -o foo foo.c
> ls
```

If you typed the `gcc` command correctly, you now should have an executable file `foo` in your directory.

- Run the `foo` program. Follow program instruction. Run it a few more times. Can you tell now, what this program is doing?

  Note: For most, if not on all labs and programs, we will be using the `gcc -ansi -Wall -Werror -lm` command to compile your submissions. This will force you to write code in purse ANSI C, and will force you to identify, debug and remove all and any compile-time errors and warnings.

5. Create your first program. Since you already have the Hello, World! program, the first program you create will be more complex.

- You will be modifying the `hello.c` program. First let us make a copy of it, that you will be using. Using the `cp` command make a copy of `hello.c`. Name your new file `aboutme.c`.

- Edit `aboutme.c` file. See **Appendix B** for commentary concerning editing C programs and editors available on CSL Linux workstations. You have to change the contents of the `aboutme.c` in the following manner:

  1. All lines in a C program that start with `"/*"` and end with `"*/"`. C compiler ignores all text in these lines. This text is for the programmers to specify what program this is, who created it, etc. . . .

     Modify the comments found in `aboutme.c` to include the following:

     - mention that Alexander Dekhtyar is the instructor of CPE 101.
     - the section you are in.
     - your name.
     - change the title of the program to indicate that this is an `"About me"` program.

     Note: All changes above are to be done ONLY in the comment lines.

  2. Let us now, edit some code. Examine the code of the current program. It should be clear by now, that the actual printing of the `"Hello, World!"` text is accomplished by the

```
printf("Hello, World!\n");
```

line. Note the following about this line:

- C programs consist of statements. printf("Hello, World!\n") is a statement of the C program.
- printf is a *special* C function whose purpose is to output information to screen. While we will learn much more about printf in near future, for now, we will use its simplest version as above: the text to be put on screen is provided in the parentheses.
- The line ends with a semicolumn (";"). In C, semicolumns are used to separate different statements. When you edit your program below, each printf statement must end with a semicolumn. Forgetting to include one for each statement is the single most common compilation error you will encounter during this course.
- The text to be printed, Hello, World! appears in double quotation marks ("). C uses double quotes to represent strings, i.e., textual information the program deals with.
- The text inside the double quotation marks ends with "\n but the output (as observed in previous runs) does not contain "\n". What is it? To determine what "\n" does, delete it from the line, (leave the line to be printf("Hello, World!"); save the program, compile it (use

  ```
  > gcc -ansi -Wall -Werror -lm -o aboutme aboutme.c
  ```

  command), and run it. What is the difference between running aboutme and a.out program that you got by compiling hello.c? Can you figure out what "\n" means and does?

3. Restore the printf statement to its original form. Add another printf statement immediately below the first one. Type

   ```
   printf("\n My name is <Name>.\n");
   ```

   replacing <Name> with your name (in my case, it'd be Alexander Dekhtyar). Save the program, compile and run it. What do you see now?

4. Edit the program to add more printf statements. The text in the statements should contain some brief information about you - your major at Cal Poly, when you joined Cal Poly, why you want to pursue Computer Science/Computer Engineering/Software Engineering (or, if not one of these majors, why you are interested in programming), as well as some of your hobbies.

   A short version of such text for your isntructor is available by running the aboutme.alex program (remember — you copied it some time ago). This file contains the executable of the instructor's version of the aboutme.c program.

   Please note, that the text you put should be readable (i.e., well-formatted, including line breaks in reasonable places), and should be truthful. I may use the outputs generated by your program in determining group and pair designations for future labs and/or program assignments.

6. Submit your program. You are at the finishing line for this lab assignment. All that is left is to submit your work. You will be submitting just one file, your `aboutme.c` program.

In all your coursework, you will be using a program called `handin` to submit your work. The typical submission format for `handin` is

```
> handin <instructor> <assignment> <fileName1> <fileName2> ...
```

Here,

- `<instructor>` is the CSL LoginId of your course instructor. My loginId is `dekhtyar`.
- `<assignment>` is the designation of the assignment, for which you are submitting. Current lab assignment designation is `lab01`.
- `<fileName1>, <filename2, ...` are the names of all the files you wish to submit. For this lab you are submitting just one file, `aboutme.c`.

Thus, to submit, you should issue the following command from your Lab 1 directory:

```
> handin dekhtyar lab01 aboutme.c
```

Note: You can resubmit your files as many times as you want until the deadline. Past the deadline, submission will be kept open for 24 hours to collect any late submissions (you can submit as many times in that period, but you run the risk of earning progressively larger late penalties), after which, the submission will be closed. Once the submission is closed, all subsequent attempts to use `handin` for the particular assignment will be rejected.

Good Luck!

Appendix A: Changing the password.

Upon successful login, start up an Xterm or a terminal. In the terminal window, type

```
 > passwd
```

You will receive the message:

```
Changing password for user <LoginId>
```

where `<LoginID>` is your CSC loginId.

The next line will prompt you to enter the current password. Type in your current password. You should not see any symbols on the terminal. After you type the password, hit the `<Enter>` key.

You will be prompted to enter new password. Enter it, hit `<Enter>`.

You will be prompted to reenter your new password. Enter your new password for the second time, making sure you enter the same sequence of characters. Hit `<Enter>`.

If you were, successful, a message saying that the password has been changed will be displayed. Congratulations, you are done!

If you were unsuccessful (your current password was typed incorrectly or you mistyped the new password when re-entering it), a message stating that the password was not changed will be displayed. You will need to try again.

Note: It takes some time for the password change to propagate throughout the entire CSL system. Because of this, it may take up to 20–30 minutes until you can log in with your new password on *any* CSL machine. Because of this, do not change password in the middle of the lab period, wait to do it until later in the afternoon.

Appendix B: Notes on the use of editors for C programs.

C programs are **plain text files**. Therefore, you are not allowed to use MS Office or OpenOffice[2] to prepare/edit your C programs.

For the most part in this course you will be using Linux plain text editors to create and edit C programs. There are four viable options available to you: `vi/vim`, `nano`[3], `emacs/xemacs` and `gedit`.

Two of these four editors, `vim` and `nano` work inside a terminal window; the other two, `xemacs` and `gedit` open windows of their own to load files. When selecting an editor to work, remember the following:

- `xemacs` and `gedit` are easier to use, as they look similar to the word processing software you may be familiar with from high school. Both feature menus, toolbars, and provide relatively easy access to some degree of text editing and formatting functionality. Remember that since these are plain text editors, some familiar word processing functionality (e.g., font selection, text color) will be unavailable to you - at least not immediately available.

  The other benefit of these two editors is that they allow you to continue using your terminal window for Linux commands - in particular, you

[2] OpenOffice is a Linux office suite which contains a word processor, a spreadsheet and a presentation manager.

[3] On some workstations known as `pico`.

can edit the C program in the editor window and compile/run it from the terminal window.

- `vim` and `nano` are text-based editors, and they run inside the terminal window itself. Both have some degree of editing functionality relatively easily available. The main drawbacks of these two editors are lack of GUI support (your mouse is useless, you cannot click on anything, must use the keyboard for all navigation and activities) and the fact that it occupies the terminal window, meaning, that in order to keep editing, while simultaneously compiling/running your program you would need to open a second terminal window.

  These editors, however, have one major advantage as well. Because they do not require XWindows for their work, they can be used to edit text files if you are logged on your machine remotely using an `ssh` session (please make sure you learn how to do this soon enough).

  Because all (or almost all) programs we will be writing in this course work inside a single terminal window, you can work on your code remotely, logging onto CSL machines from your home desktops or personal laptops. If you do so, using `vim` or `nano` will allow you to edit your programs.

Because of these considerations, you should really familiarize yourself with at least one editor from each category.

Here is a brief overview:

vim: for a text-based editor, `vim` is surprisingly quite powerful. It is also going to be the most unusual of all the editors for you. Because `vim` lacks GUI, its designers chose to implement editing functionality by introducing two modes (states) in which `vim` can be: *command* and *edit*. In the *edit* state, the user can navigate through the text of the current file and type in new text and/or replace old text. In the *command mode* all keypresses are interpreted by `vim` as commands which facilitate various editing functionality: from text deletion to search and replace, etc.... `Vim` has a large number of commands, entered as keystroke combinations on the keyboard. Learn just a handful of basic commands and be able to use `vim` to edit your small C programs. Or, become a `vim` guru and use it as your primary editor for all your tasks!

  `Vim` is a newer version of `vi`: one of the oldest Unix text editors. A lot of your professors, who went to school when `vi` was the only text editor available on their Unix systems still successfully and productively use `vim`, `vi`, or the most recent addition to this family of editors, `elvis` (not available currently on our system). Knowing `vim` may earn you respect of your peers and instructors, but the process of learning how to edit within `vim` can be daunting, error-prone (one too many "d" keystrokes entered at an inopportune time, and you are missing half of your C program...), and different from your experiences with other text editors. On the bright side, `vim` is very fast, and always works.

nano is the simplest editor of the bunch. It does not boast `vim`'s vast array of functionality, **but** all functionality that `nano` has is easy to discover and easy to use. All special commands are done using the `Ctrl` keyboard key, and the list of these commands is found at the bottom of the `nano` screen. You will not be able to do complex text editing in `nano`, but the learning curve for it is almost non-existent.

`nano` is a modified version of `pico` (available on some of the CSL machines, but for the most part, depricated in CSL), which, in turn is the default editor for the `pine` mail client program, which is available on the Linux machines. It was built to be simple, and easy-to-use.

xemacs is the most powerful editor of the four. It is an XWindows wrapper around `emacs`, the famous Unix editor. Learn some LISP programming, and you can make `emacs` (and `xemacs`) do virtually anything, as `emacs` comes with its own programming environment, and a large set of macros for things like programming language syntax highlighting, and much, much more. An an editor, `emacs` is very non-user-friendly. `Xemacs` somewhat alleviates this problem, providing menus and toolbars for the most important editing commands, but still leaving the possibility of using the vast array of `emacs` functionality via keystrokes.

A lot of your professors, especially those who went to school in late 1980s - early-to-mid 1990s will have used quite a lot of `emacs` in their carreers and continue using it (or `xemacs` today). Working knowledge of `emacs` commands may earn you some degree of respect from them.

gedit is essentially a prototypical text editor, without any specific backstory (like LISP-powered `emacs` has). Its use should be relatively straightforward for anyone, who'd worked with Wordpad under MS Windows, or with any other simple text editor before. It has all the necessary features (cut-and-paste, search-and-replace, print, etc...), uses traditional Windows shortcuts for cut-and-paste functionality (something, `xemacs` does not do), and provides convenient access to all the functionality via the menu system and toolbars. It also allows tabbed editing - you can edit muiltiple files in different tabs at the same time.

Overall, you will find that a lot of professors suggest (or insist) that you use `vim` and/or `xemacs` in your coursework. Your instructor firmly believes in "teach-what-you-do" principle. Since your instructor tries his best to **avoid** using either `vim` or `emacs`-derivative editors in his work, his advice to you is to start by using `gedit` when running XWindows, and `nano`, when using remote login. If interested, you can later take it upon youself to learn the other two editors, and become a real guru. CPE 101, however, is not about text editors. For more on the editors, see the enclosed comic strip.