Lab 0: Surveys and Setup

Due date: Wednesday, January 4 end of lab period.

Surveys

With this lab description, you will receieve a survey form and an index card. Please take time to fill out both the survey and the index card according to the instructions below.

Survey. Your survey answers will be used to form pairs and teams for a variety of assignments throughout the quarter. Because of this, I require that you include your name on the survey.

Pages 1, 3 and 4 of the survey shall be completed by everyone.

Page 2 asks questions about CSC 123. If you are not a CSC/CPE/SE major or if you are a trasfer student who has not taken this course, answer "No" to the first question and skip the rest of the questions on the page.

Index card. Fill both sides of the index card as follows.

On **one side** of the card, write your name, major, and the reason(s) why you chose your major.

On the **other side** of the case, please write one interesting fact about yourself. This can be anything you'd like to share with me.

Submit both the survey and the index card at the end of the lab period.

Setup

We need to make sure you are ready for the labs to come. The tasks below verify that you have a working CSL (Computer Science Labs) account.

Those of you who took CSC 123 may need to perform fewer activities than those who have not. The list of tasks below applies to the situation when you have not yet used your CSL account or have not used it extensively.

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 machines) come with the same installation of Linux (CentOS). The machines, including the ones in our lab, 14-302, are dual-boot, and allow one to boot into Windows 7. **In CPE 101 we will always be using Linux**. If your machine is displaying Windows 7 login, reboot it.

Your CSL login Id and password are your Cal Poly login ID and your MyCalPoly password.

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

1. At the Username prompt that appears, enter your CSL loginId (username).
2. At the Password prompt enter your current password.

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 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:

   ```
   dekhtyar@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 0 files, and change to that directory.

7. Start a text editor of your choice. You can use any plain text editor you like. If you are not certain what editors are available to you, read **Appendix A** for some options that are available on the CSL machines.

8. Create a text file that contains the following information:

   - Your name
   - Your Cal Poly email address
   - Your section of CPE 101 (01 or 09)
   - Your major
   - A short paragraph describing what you expect from this course.

   Save the file under the name `lab00.txt`.

9. Submit your file to the instructor.

   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. You will be submitting your assignments to one of two different LoginIds. My loginId is dekhtyar. The loginId for the grader account for this course is dekhtyar-grader. Each time you are given an assignment which includes an electronic submission component, I will specify the name of the account to which you have to submit.
   - <assignment> is the designation of the assignment, for which you are submitting. Each assignment has two designations (representing the actual physical directories where the submissions will be placed): one for Section 01 and one for Section 09. This lab uses lab0-01 amd lab0-09 as the designations to be used by students in Sections 01 and 09 respectively.
   - <fileName1>, <filename2, ... are the names of all the files you wish to submit. For this lab you are submitting just one file, lab00.txt.

   Thus, to submit, you should issue the following command from your Lab 0 directory:

```
> handin dekhtyar lab00 lab00.txt
```

   Note: You can resubmit your file(s) as many times as you want. New submissions overwrite the previously submitted files with the same name.

Good Luck!

Appendix A: 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[1] 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 multiple viable options available to you: `vi/vim`, `nano`[2], `emacs/xemacs`, `gedit`, `kwrite` as well as the editors in various Interactive Development Environments (IDEs), such as `Eclipe`.

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

- `xemacs`, `gedit` and `kwrite` 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 editors is that they allow you to continue using your terminal window for Linux commands - in particular, you 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.

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

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

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. `nano` and `pico` can be used interchangebly.

emacs is the most powerful editor. Learn some LISP programming, and you can make `emacs` 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. `Emacs` provides menus and toolbars for the most important editing commands, but still leaves the possibility of using the vast array of 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. Working knowledge of `emacs` commands may earn you some degree of respect from them.

gedit and **kwrite** are essentially prototypical text editors, without any specific backstory (like LISP-powered `emacs` has). Their use should be relatively straightforward for anyone, who'd worked with Wordpad under MS Windows, or with any other simple text editor before. Both have all the necessary features (cut-and-paste, search-and-replace, print, etc...), use traditional Windows shortcuts for cut-and-paste functionality (something, `emacs` does not do), and provide convenient access to all the functionality via the menu system and toolbars.

`Gedit` also allows tabbed editing - you can edit muiltiple files in different tabs at the same time, while `kwrite` always opens files in new windows.

Overall, you will find that a lot of professors suggest (or insist) that you use `vim` and/or `emacs` 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.