Assignment #0: Getting familiar with Unix, vi, compiling, and debugging

In this assignment, you will make certain that you can (details are provided below):

- find the main CS lab at **NPB 2.118**.
- login to your user id.
- learn some essential **Unix** commands (pwd, mkdir, ls -al, cd, man, rm, gcc, cat, vi) and get familiar with the Unix cheat sheet.
- learn to use **vi** and get familiar with the vi cheat sheet
- modify the provided C source file to complete documentation and coding (see 3.4)
- learn to use **gcc** to compile your code
- learn to use **ddd** to debug your code
- optionally, learn to use Microsoft Visual Studio

Some useful stuff:

viCheatSheet	This word document provides a brief list of the most important commands for using **vi**. It is available on my webpage under "setup".
unixCheatSheet	This word document provides a brief list of the most important **Unix** commands. It is available on my webpage under "setup".
dddTips	This word document gives some tips on using the **ddd** debugger. It also has a reference to a detailed tutorial. It is available on my webpage under "setup".
vimtutor	This provides a tutorial that allows you to practice using the Unix vi editor. Change to the ~/tmp directory (**cd ~/tmp)** and execute vimtutor. Follow the directions provided in the file.
usingMicrosoftVisualStudio	(optional) This word document describes how to use Microsoft Visual Studio. It is available on my webpage under "setup".
ssh download	(optional) This will download a copy of ssh suitable for windows. It is available on my webpage under "setup".

Steps:

1. Login

1.1 Go to the CS lab NPB 2.118 (assuming it is available). If working remotely, see how to login remotely on my setUp and skip to step 1.5.

1.2 There should be two types of workstation configurations in the CS lab:
Dell Thin Client - (left side of lab) which access VDI. See how to login to VDI on my setUp.
Linux Workstations - (right side of lab) these run Linux 14.04 and have a good desktop.

1.3 Use a Linux workstation. Login locally using your id (3 letters and 3 numbers). Your password will be defaulted to your UTSA banner ID without the @.

1.4 Open a terminal window. If your Linux login has a menu, use Applications>Accessories > Terminal. If not, use
- Click on the bottom left icon which will show an applications menu.
- Click on Accessories
- Click on LXTerminal
- Most of the discussion below uses the Terminal window.

1.5 Since your password was defaulted, change your password using the **passwd** command. (Note that the "$" is being used to indicate that the Linux shell is prompting you for input. The actual prompt may include your user id.)In the terminal window type:

```
$ passwd
```

Notes: If you are logging in remotely (from home):

If using linux:

- Start a terminal window:
  `$ ssh -X userName@foxii.cs.utsa.edu` (where *ii* is one of 01 through 06)

If using windows from home:

- You should install **ssh** (see the setup web page).
- You can transfer files, use **vi,** and the **gcc** compiler, but you can't use **ddd** without an x-window.
- When you use ssh specify *userName*@fox*ii*.cs.utsa.edu (where *ii* is one of 01 through 06)
- (In the future, consider using ubuntu as a dual boot or use Cygwin.)

2. Check out some Unix directory-related commands. (You may want to look at the unixCheatSheet.)

2.1 After logging in, check your current directory using **p**rint **w**orking **d**irectory:

```
$ pwd
```

2.2 See what files are in your directory:

```
$ ls -al
```

This will list all files with long details). You should notice some dot files (begin with ".") such as .vimrc

2.3 Create a directory for cs2123:

```
$ mkdir cs2123
```

2.4 Change to your cs1713 directory:

```
$ cd cs2123
```

Verify that you are in that directory by using the **pwd** command.

2.5 Change back to your home directory:

```
$ cd ~
```

Verify your current directory using **pwd**.

2.6 The following are done to practice creating directories and subdirectories:

2.6.1 Create a subdirectory called **Fruit** and another called **Veggie** under your home directory.

```
$ mkdir Fruit
$ mkdir Veggie
```

2.6.2 Create two subdirectories under **Fruit** called **Apple** and **pineApple.**

```
$ cd Fruit
$ mkdir Apple
$ mkdir pineApple
```

2.6.3 Change directory to the **Apple** directory and then use **pwd** to check where you are:

```
$ cd Apple
$ pwd
```

2.6.4 Create a subdirectory under **Apple** called **Green**

```
$ mkdir  Green
```

2.6.5 Go up to the **Fruit** directory by using

```
$ cd ..
$ pwd
```

Note that **`cd ..`** took you up one directory.

2.6.5 You can specify multiple directories in a path when using cd

```
$ cd Apple/Green
$ pwd
```

You should be in Fruit/Apple/Green

2.6.6 Go to the **Veggie** directory using ~ for the home directory.

```
$ cd  ~/Veggie
$ pwd
```

2.6.7 See what files exist now in your account.

```
$ cd ~
$ ls -alR
```

This lists files recursively following subdirectories.
If you have more than a page worth of files, use **ls -alR | more**. To exit out of **more**, type **q**.

2.7 Use the **man** command to see how it gives you manual pages for most Unix commands by typing **man *command***. You can scroll through the manual pages by pressing the space bar to scroll a page or **enter** key to scroll fewer lines. To quit out of man, type **q.**

```
$ man  ls
```

2.8 Practice using the **find** and **rm** commands.

2.8.1 Use the **find** command to find all files containing the word "Apple".

```
$ find  .  -name  "*Apple*"
```

2.8.2 Use the **rm** command to remove the Fruit folder and all its contents.

```
$ rm -r Fruit
```

2.8.3 Use the up arrow several times to get Linux to show the find command you did. If you go up too far, use the down arrow. Execute that find again and notice that it doesn't find the Apple and pineApple directories.

2.8.4 Remove the Veggie directory.

3. Use the **vi** editor.

3.1 Look at the viCheatSheet.

3.2 Try the vi tutorial:

3.2.1 `$ cd  ~/tmp`

3.2.2 `$ vimtutor` (follow the instructions in that file)

3.3 Create a **.vimrc** file to help you with indentation and showing line numbers.

3.3.1 `$ cd ~`

3.3.2 Use the Unix **cat** command to create that file (input is terminated by pressing the **CTRL-D key**). Note that this only has to be done once.

```
$ cat  >.vimrc
:set ai sw=4
:set number
:set expandtab
:set softtabstop=4
:set smartindent
CTRL-D
```

3.4 Create your first program for this class. (Create the cs2123 folder if you haven't already.)

3.4.1 `$ cd  ~/cs2123`

3.4.2 Copy the code (details of how to do this are shown in 3.4.2.1), include file and data from my web page**:**

p0abc123.c - contains most of the code for program #0. You will have to make several changes:

getCourseData - replace the function documentation header with a good one based on my programming standards

printCourseData - change the code to print the course data as shown in the existing function documentation header.

In the program documentation header, specify your name.

cs2123p0.h - include file for this assignment
p0Input.txt - data file to be used with this assignment

3.4.2.1 Invoke the firefox browser:

```
$ firefox
```

3.4.2.2 The firefox internet browser will open. Within firefox, go to Blackboard and login with your abc123 and pass phrase:

http://utsa.blackboard.com

3.4.2.3 Go to our class **CS2123** (not the recitation). Then go to Content -> Programming Projects and download p0abc123.c, cs2123p0.h, and p0Input.txt. Save them to your cs2123 directory**.**

3.4.3 You may want to check to see if the files contain a combination of carriage returns (\r) and line feeds (\n) which happens when some file are copied from Microsoft Windows. See the setUp page.

3.4.4 Use **vi** to make your changes (as described in 3.4.2)

```
$ vi  cs2123p0.c
```

4. Compile your program, naming the executable p0:

```
$ gcc  -g  -o p0  cs2123p0.c
```

where `-g` tells the compiler to generate information suitable for the debugger,

`-o` specifies the name of the result (in this case an executable)

This compiles `cs2123p0.c`, creating an intermediate compiled object file, and generates an executable named p0. Until we have more complex assignments, we will use that form of the gcc.

If you have any compilation errors, the compiler will tell you the line number. Correct your mistakes using **vi** and repeat step 4.

5. Execute your program using the provided input file.

```
$ ./p0 < p0Input.txt
```

You can look at the output by piping the result to **more** (use **q** to quit, **Enter** to advance 1 line, **SpaceBar** to advance many lines)**:**

```
$ ./p0 < p0Input.txt | more
```

You can save the output in a file by redirecting stdout to a file using ">":

```
$ ./p0 < p0Input.txt > p0Out.txt
```

6. Use the debugger.

6.1 Look at **dddTips.docx** under my setup web page. You will find it important to refer to that information while using **ddd**.

6.2 Invoke the **ddd** debugger. (Note that if you didn't use the -g compiler switch, the debugger will not work. Also, if you are not executing from a Unix terminal window, ddd will probably not work.)

```
$ ddd  ./p0
```

You may want to show the line numbers for each statement. See dddTips.

6.3 You will see the ddd screen which has the following areas:

Menu Bar	This is at the menu at the top of the window and includes File, Edit, View, etc.
Tool Bar	This is immediately below the menu bar and contains some important tools like Find and Break.
Data Area	At the beginning, this shows an empty grid area. It is used to show values of variables which you wish to closely examine.
Source File Area	This displays your source code.
Command Tool Area	This appears on the right side of the Source File Area and includes buttons:

Run	run the program.
Step	execute the current step. It will step into a called function.
Next	execute the current step, but do not step into a called function.
Cont	Continue execution until the next break or stdin.
Finish	continue execution until the finish of the current function.
Kill	Kill the execution of your program.

Console Area — This is at the bottom of the screen:

- debugger commands are shown/entered
- standard input is entered (depending on preferences)
- standard output is shown (depending on preferences)

6.4 Tell **ddd** to show line numbers. This is done via the **Source** menu item.

Source > Display Line Numbers

6.5 Run your code using the **Program** menu item.

- Click on **Run in Execution Window** to give you an execution window when you run. You only have to do this once.
- Run your code using Program > Run. Tell the Run dialogue window to use the input file.

  For arguments, specify:

  `< p0Input.txt`

- Click the **Run** button in that Run Program dialogue window.
- Your program should run to completion unless there is a bug. If it didn't work and you wish to terminate the program press the **Kill** button in the Command Tool Area.

6.6 Set a Breakpoint on the first **printf** statement in getCustomerData(). Click on its line number, right click to get a submenu, and then select **Set Breakpoint**. A stop sign will appear on that line. This will cause the code to break execution so that you can examine what is happening at a point in the code. You can set many breakpoints.

6.7 Run your code using Program > Run. Again tell the Run dialogue window to use the input file.

For arguments, specify:

< p0Input.txt

6.8 Execution should have stopped at the first break point. Notice that an arrow will appear in the Source File Area indicating the current statement to execute. In this section, you will step through your code using the **Next** button and examine variable values. You can hover the cursor over a variable to see its value.

Try the **Cont** (continue) button. It will continue executing until it hits a breakpoint.

Eventually hit the **Finish** button. It will ignore breakpoints in getCustomerData() until it returns from getCustomerData().

Continue executing until the program completes.

6.9 Execute your program again. It is often necessary to see what is happening with your variables. You can **display** variables in the Data Area. Examining arrays is a little more difficult. Arrays and complex structures are usually shown in the Display Area. Right click on the customer array and select **Display.** If your variable is a pointer, you might prefer to display what it references by selecting **Display***. Sometimes, it it easier to display a slice of the array. Please see the **dddTips** for more information.

Step through your code using **Next** and see how the values in the Display Area change.

When you get to the statement which invokes the getToken function:

- If you use **Next**, **ddd** will execute that function and then return.
- If you use **Step** (which means step into), **ddd** will **step** into that function, allowing you to use Next to within that function.

If you want to stop debugging, use the **Kill** button in the Command Tool Area.

6.10 Another useful feature is to tell **ddd** to interrupt execution when a particular variable's value changes.

Clear all your breakpoints by clicking on each breakpoint and then pressing **Delete Breakpoint**.

Insert a new breakpoint at the call to getCustomerData().

Run your program using the input file. It will stop at getCustomerData().

Insert a **Watch point** to watch for when the `iNumberOfCustomers in main()` changes. (This is done by clicking on that variable and then selecting the watchpoint button, which looks like two eyeballs, on the main tool bar.) Press **Cont** to continue execution. Notice that execution stops when that variable changes even though it is the corresponding parameter.

When you are finished using ddd, type **quit** in the Debugger Console.

7. We want to save the output of your program by redirecting the output to **p0Out.txt**

```
$ ./p0  < p0Input.txt >p0Out.txt
```

That redirected the stdin (standard input) to come from **p0Input.txt** and stdout (standard output) to **p0Out.txt**.

To save the output when using Microsoft VS, see the instructions in setup.

8. Upload your results to blackboard.

In the Programming Projects folder on Blackboard, there will be a dropbox for Programming Project 0. Follow the directions on the screen to upload mutiple files**. Upload p0abc123.c** and **p0Out.txt**