Lab 5

Get checked off incomplete work from the previous lab within the first 10 minutes of lab.

This part of the lab cannot be completed in pairs!!!

(5 pts) 50 minutes Practice Proficiency Demo

1. First, open a terminal on the host computer:
   - Once setup, type `vim` to use vim, not just vi, and create `test.cpp`
   - Compile your program by typing `g++ test.cpp -o test`
   - To run your program, use `./test`
2. Get 2 questions from the TA. Choose one question you want to solve. All questions include variables, user input, conditional statements, and repetition. You should be able to finish the code within 50 minutes.
3. Scoring:
   - 5 pts if fully coding the solution
   - 3 pts if getting pretty far, but not finishing the if/else or loop logic
   - 1 pt for no clue but showing up
4. When you finish, get a TA to check you off

You can complete this section individually or in pairs.

(1 pt) VIM Exercises

These **exercises on the computer need to be repeated by each student** in the pair. This is to ensure that both students understand how to get around in Linux!!!

Now, let's practice using the vim editor following the instructions below. (If needed, refer to Lab #1 for a reference guide to the basic commands)

1. Open a test file for practice – **vim_test.cpp**
2. Save the file using – **:w** <Press Return>
3. Go into insert mode by pressing **i**
4. Type the standard input/output header and main function in your test.cpp file.
5. Get out of insert mode by pressing <Esc>.
6. Show the line numbers in vim by typing **:set number**
7. Go to the third line (or whichever line has main) in your code by typing **:3**
8. Delete this line of code by typing **dd**
9. Put the line of code back underneath the first line by moving your cursor to the top line and typing **p**
10. Redo the delete by typing **u** to undo the paste.
11. Paste the line back using **p**

12. Now, copy the line of code by typing **yy** (If you want to copy >1line, you type one less than the number of lines you want to copy after the y, i.e. y4 copies 5 lines)
13. Paste the line of code by moving your cursor to a line and typing **p**
14. Undo the paste by typing **u**.
15. Try using the **j**, **k**, **h**, and **l** keys to move around the screen. You can also use the arrow keys, but these don't always work.
16. Try searching for "main" in vim with **/main**.
17. Go back into insert mode, **i**, and type some random letters, dkfjdsl, then press <Esc> to go back into command mode.
18. Use the **x** to delete all the characters from this random string you typed.
19. Press **i** to begin typing text again.
20. Now, write a small program:
    - Ask the user if she/he likes vi as an editor.
    - Read the 0 or 1 integer value from the user
    - If the user says true, then display a message, "You love vi!"
    - If the user says false, then display a message, "You hate vi!"
21. Now, press <Esc> to get back into command mode, and go to the first line in your program by typing **:0**.
22. Then, type **=G** to auto indent your program. You can set the number of spaces you prefer by typing **:set sw=3**, then you want to auto indent again, **=G**. (If you want to auto indent while you type, then use **:set cindent**)
23. You can change your color scheme or background by using one of these commands:
    :colorscheme evening
    :set background=dark
24. Now, write and quit the file by typing **:wq**

For this part of the lab, you will create a .vimrc file that will help you develop your C++ programs using Vim. First, we need to create a simple .vimrc file in your home directory on the engr server.

vim .vimrc

In this file, you can insert the following lines so that it makes working with vim easier. Comments are prefaced with a quotation mark, ".

```
filetype on
filetype plugin on
filetype indent on
autocmd FileType c, cpp
set cindent "This allows for c-like indentation
set sw=3 "Set the sw for the auto indent to 3 spaces
set number "Show the line numbers on the left
```

```
“Change the color of the text and turn on syntax highlighting
“color desert
color torte
colorscheme evening
syntax on “Turn on syntax highlighting
set showmatch “Show matching braces
set showmode “Show current mode
“When one of the chars is typed, the matching is typed and the cursor moves left
“The line below is single quotes
inoremap ' ''<Left>
“The line below is double quotes
inoremap " ""<Left>
inoremap { {}<Left>
inoremap ( ()<Left>
```

There are many more commands you can insert in this file, and here is a reference guide to some of these: http://vimdoc.sourceforge.net/htmldoc/starting.html.

You can complete this section individually or in pairs.

(4 pts) Practice Value-Returning Functions and Passing Arguments

Write a function called **get_sentence()** that gets the initial input string/sentence. Create a string object to read the string from the user, **#include <string>**. Since, a user can enter a single word or a string with spaces, you will need to use the getline() function to read the string/sentence from the user, i.e. **getline(cin, s).** Use this function to get the string/sentence from the user and print the contents of the string after making the call to this function. For example:

```
int main() {
  string sentence;

  sentence = get_sentence();
  cout << sentence << endl;

  return 0;
}
```

Based on the function call above, notice that get_sentence() is a value-returning function because the value being returned is stored in sentence, and it doesn’t have any arguments in the function call (between the parenthesis is blank). Therefore, the prototype for this function is as follows.

string get_sentence();

(2 pts) You must define the value-returning get_sentence() function based on this information.

Now, let's make get_sentence() a void function and pass the string in main to the function to get the sentence from the user. This means you cannot assign the value returned from the function call to the sentence variable in main! Since the function doesn't return the value back to the call, it has to modify the string argument directly. Our main function will change to the follow:

```
int main() {
   string sentence;

   get_sentence(sentence);
   cout << sentence << endl;

   return 0;
}
```

Based on the function call above, notice that get_sentence() is a void function because we don't do anything with the function call (it is on a line by itself). It has a string as an argument in the function call (between the parenthesis is sentence). Therefore, the prototype for this function is as follows.

void get_sentence(string s);

Note the parameter name does not need to match the argument name. These are indeed different places in memory!

(1 pt) Write the definition for the void get_sentence() function, and note what is printed by the cout after the function call, regardless of what the user enters.

Notice we can use the string inside the function, but we cannot change the value of the string inside the function. Let's change our parameter to contain an ampersand in front of the variable.

void get_sentence(string &s);

(1 pt) Make this change to the parameter in your void function and note what is printed by the cout after the function call.

Show your completed work and answers to the TAs for credit. You will not get points if you do not get checked off!