Lab 4
Strings and vectors

19 September 2019

The first program in this week's lab will be on repl.it, and it will read a single string—specifically, it will read one string that doesn't have spaces in it—and print out information about it. Remember to test your program after each step!

1. First, write the program to read the string and print out the message

   ```
   The word ______ has _ characters.
   ```

   except with the blanks filled in correctly for that string: the string itself and its length (how many characters it has).

2. Add second line to the output that says

   ```
   Its first is '_' and its last is '_'.
   ```

   filling in the blanks with the first and last characters of the string (you can assume it will not be empty).

3. In cases where the string is at least three characters long, add a third line

   ```
   If you trim the first and last characters it leaves ____.
   ```

   The blank here should have the whole string *except* for its first and last characters.

4. Finally, print one line that says

   ```
   The characters are:
   ```

   and then lines with the characters of the word, in order (each on its own line).

Don't forget to include documentation comments at the top of your program.

I'll be circulating around the lab to answer questions. If you're stuck on some part of the drill, ask me about that (and while you're waiting for me to get to you, look at the next section about dotfiles). If you're not stuck but haven't finished the drill, work on that now. If you're done with the drill, continue on to the next section.

Vim FOTD: Cut and paste

Open a new file called `dummy.txt` in vim. Enter insert mode (by pressing '`i`'), type two lines of text, and then escape back to command mode.

Somewhere in the middle of the first line, press '`x`' a few times. (This removes characters.) Now press '`p`' a few times.

Somewhere in the middle of the second line, press '`x`' again. Now press '`P`', that is, Shift-P—notice what it did differently?

Go back to the first line, and type '`dd`' (i.e. press the '`d`' key twice). Press '`p`' a few times.

Go to the new first line, and type '`dd`' again. This time, press '`P`' (Shift-P) a few times.

What's happening here is that every time you use a vim command to delete something,[1] it's stored in a clipboard (as if you'd selected "Cut" in a GUI word processor). Then, you can use one of the two paste commands to put the text back—where it was, somewhere else, or any number of times.

There are two kinds of cut commands: those that remove some number of characters and those that remove some number of lines. You've now seen one of each ('`x`' deletes the character under the cursor, and '`dd`' deletes the line under the cursor). The paste commands differ in that '`p`' means "paste clipboard contents *after* this spot" while '`P`' means "paste clipboard contents *before* this spot". If the contents of the clipboard were character-based (like if you hit '`x`'), then "this spot" means "the character under the cursor", and if the contents were line-based (like if you hit '`dd`'), then "this spot" means "the line under the cursor".

A few more occasionally-useful delete commands:

`dw`	Delete to end of current "word" (char-based)
`db`	Delete back to beginning of current "word" (char-based)
`d$`	Delete to end of current line (char-based)
`d}`	Delete to next blank line (line-based)
`7dd`	Delete seven lines (line-based) (similarly for other numbers)
`dG`	Delete to end of file (line-based)

For each of these, replacing the `d` with a `y` makes Vim do a copy instead of a cut. (The mnemonic for `y` is "yank".)

[1]NB: this doesn't include using the Backspace key while in Insert Mode.

Part 2: Print initials

The purpose of this whole lab is to give practice with strings and vectors, so we don't have interesting word problems to work through—all the parts of this lab have a sort of "drill" flavour about them (sorry).

In this part, you'll write a program that processes a vector of strings and prints out the initial character in each string. For instance, if the vector contained the strings `"Longwood"` and `"University"`, the program's output would be "`LU`".

You may assume none of the strings are the empty string. (You could skip it if you encounter it, but I'm not checking for this.) I do expect that you'll try running your program with different values in the vector, but since the only way we have (for now) of making vectors with values is to put them in when the vector is defined in `main`, there's no separate place to define test cases; you're on your own for making sure it's well-tested and correct.

Do this problem on repl.it, in the Lab 4 Part 2 assignment.

Part 3: Skip lowercase

You'll do this part on the department server, so create a directory for this lab if you haven't already; don't forget to put a readme file in there, and remember that you'll have to write some test cases (.in and .expect) and show in the readme how to use them to test your code.

In this part, you'll write a program that reads in a single word and then prints back only those characters that aren't lowercase letters. That's *not* to say that you *change* letters to uppercase, just that you omit letters that are lowercase (but print the uppercase letters and the characters that are not letters at all).

There are a few possible ways you might identify lowercase letters. To nudge you towards (what I think is) the easiest of them, I'll point out that `char` values, like the other basic types we've seen, can be compared using the relational operators. I'll also remind you that `char` literals are indicated using single quotes, like this: `'Q'`. (Using double quotes, `"Q"` gives you a single-character string, not a `char`.)

Handing in

It's due as usual on Wednesday at 4pm. The drill and Part 2 are already on repl.it and should be submitted via the "submit" button there; Part 3 needs to be handed in on the server. Hand in using the usual command, this time with assignment name `lab4`.

Rubric

RUBRIC (Tentative)

1	Attendance at lab with drill done or question written down
1	Appropriate documentation in each part
Drill (string info)	
1	Program compiles and runs, reads string, prints it out
1	Uses `if`, `length` correctly
1	Uses `substr` correctly
Print initials	
1	Define and init a vector of string (with valid strings)
1	`for` loop prints initials
Skip lowercase	
1	Program compiles and runs, reads string, prints something
1	`for` loop prints excluding lowercase
1	Test cases, good coverage *

* To get the full point for good test case coverage, you have to EITHER pass all test cases in the group OR indicate in the readme which ones aren't passing. This is your way of showing me that you've actually run your tests.