Lab 5: Released Wednesday, week 6 and due Tuesday, week 7

Note: This lab gives you all of the instructions for writing your code in Step 2, and then the instructions for compiling in Step 3. You are not required to write all three programs before compiling, feel free to write one, compile and test it, and then move on to the next.

Fun with Strings and Files

Introduction

The assignment for this week will utilize concepts of string manipulations and sorting algorithms.

We will also be grading for meeting requirements, using “class legal” code, and plagiarism. So, it is not enough for your lab to just pass the Gradescope autograder! Please read the instructions herein carefully. Remember that copying solutions from online (or adapting them, with minimal changes) is plagiarism. **If you plagiarize this assignment you will get a 0 with no chance to regain points.**

It is highly recommended that you develop the algorithms for each program first and then develop the C++ code for it.

Step 1: Getting Ready

Recall what environment you set up in lab 1. Navigate there, and then clone the Github repository. Go to the course Github (https://github.com/ucsb-cs16-1-u22) and make sure you can see a repository named Lab05-yourgithubusername. If you can see that repository, open it up in your browser and go to the green “Code” button near the upper right. Copy the SSH link for this repository (for more instructions on this step, and future steps involving Github, check out https://sites.cs.ucsb.edu/~maradowning/cs16/refs/gitbasics.html).

From here, run the following command in the folder you’d like your Lab05 folder to end up in:

```
git clone <link you just copied>
```

Check afterwards with `ls` to make sure the repository has been cloned properly, you should see a new folder named Lab05-yourgithubusername. You can run the following command to enter this directory and see the starter code files:

```
cd Lab05-yourgithubusername
```

Step 2: Edit your C++ Programs

This week, you will need to create two (2) C++ programs called sentence.cpp and stats.cpp. They are worth 50 points each and have to be submitted properly for full assignment credit.

IMPORTANT NOTE: We will take major points off if you use C++ instructions/libraries/code that either (a) was not covered in class, or (b) was found to be copied from outside sources (or each other).

Program 1: sentence.cpp
The program will ask the user for a sentence and will then, re-arrange the letters in the sentence by

- Converting all alphabetical characters to lowercase,
- Sorting them in alphabetical order (per ASCII codes), and
- Removing all non-alphabetical characters.

The program will then show the frequency of every letter in the string (but just the alphabetical characters). You have to convert all upper-case letters to lower-case. Additionally, you should print at the end of the frequency list how many characters you removed.

For example, see the 2 runs below:

```
$ ./sentence
Enter sentence: How now brown cow?
Sorted and cleaned-up sentence: bchnnoooorwwww
b: 1
c: 1
h: 1
n: 2
o: 4
r: 1
w: 4
removed: 4

$ ./sentence
Enter sentence: $1,234.00
Sorted and cleaned-up sentence:
removed: 9
```

Requirements

- Your program must utilize the Bubble Sort algorithm (not another sorting algorithm) that we reviewed in lecture. You will need to adapt the algorithm to a string, rather than to an array of integers.
- Be sure to utilize only techniques we've covered in lecture. Do not use "special" arrays, do not use vectors, do not use built-in sorting functions, etc. You will get zero points if you do.
- Start your program using the sentence.cpp program that I've provided you with. The skeleton program shows 2 called functions, which, of course, you must use. You can create more functions in your program if you need to (you likely will). You are free to change the arguments to these functions as you see fit.

Hint: - Realize that, in C++, assigning char types their ASCII code (which are integers) is legal. For example char letter = 65; assigns A to letter. See https://simple.wikipedia.org/wiki/ASCII for a list of ASCII character codes.

Extra Credit (5 points): Before printing the sorted and cleaned up sentence, also print out how many swaps you performed in the bubble sort as compared to the total swaps in the worst case scenario (fully reverse order). Use the following as an example:

```
$ ./sentence
Enter sentence: How now brown cow?
Ratio of swaps to worst case: 35/91
Sorted and cleaned-up sentence: bchnnoooorwwww
b: 1
c: 1
h: 1
n: 2
o: 4
r: 1
w: 4
removed: 4
```

Program 2: stats.cpp

This program takes its inputs from a file that contains an indeterminate number of floating-point numbers. The program reads them in as type `double`. The program outputs to the screen the mean (average), the median, the standard deviation of the numbers in the file. Additionally, the program should print out the proportion of numbers that are negative. The file contains nothing but numbers of type double separated by blanks and/or line breaks. For this exercise, the average is the sum of all the numbers divided by the count of these numbers. The median is the number that is in the middle of the list, where 1/2 of the numbers in the list are smaller than the median and the other 1/2 of the numbers is larger than the median. If there are an even number of numbers, then the median is the average of the 2 middle numbers. The standard deviation of a list of numbers x_1, x_2, x_3, . . . x_n is defined as:

$$\sqrt{((x_1 - a)^2 + (x_2 - a)^2 + ... + (x_n - a)^2)/(n - 1)}$$

Where the number a is the average of x_1, x_2, x_3, . . . x_n.

Requirements

1. Your program should take filename as an input from the user. If the file does not exist, it should print the error "File could not be opened." to cerr.
2. You can safely assume that the number files will never have more than 10000 numbers in them.
3. Your program should define at least 4 functions that only do the calculations for the mean, median, standard deviation, and proportion of negatives (they must not print to standard out) and return the values back to the function caller.
    a. You can have additional functions, if you like.
    b. If your program does not have these 4 functions, you will not get credit for this part of the assignment, even if your program passes the Gradescope autograder.
    c. You can pass the output of the mean function into the standard deviation function, if you want (would be helpful).
4. Be sure to utilize only techniques we've covered in lecture. Do not use “special” arrays, do not use vectors, do not use built-in sorting functions, etc. You will get zero points if you do. You have to calculate mean, median, and standard deviation using the “classical” methods talked about here—no shortcuts.

Let’s say the input file, nums1.txt, contains this (note the separation by whitespaces):

```
6 9
7
8
```

Then, a session should look exactly like the following example: (different numbers of decimals are OK). Do not add any formatting to your numbers, just print the doubles and let the compiler decide how many decimal places each one should have.

```
Enter filename: nums1.txt
Mean = 7.5
Median = 7.5
Stddev = 1.29099
Negatives = 0%
```

Let’s say that another input file, nums2.txt, contains this (note the separation by whitespaces):

```
-1.2 -4.2 0 -10 -5
```

Then, a session should look exactly like the following:

```
Enter filename: nums2.txt
Mean = -4.08
Median = -4.2
Stddev = 3.90026
Negatives = 80%
```

One more example: consider input file, nums3.txt, which contains:

```
55.8 86.9 2.4 16.8 5.9 10.2 11.2 25 96.1 4 12.7 81.7 60.4 -72.1 52.1 39.6
24.2 2.6 -52.3 49.9 14.4
90.9 48.1 88.6 44.3 -51.1 55.1 77 33.4 8.4 66.2 89.2 95.3 68.6 41.2 36.5 14
52.4 96.7 10.2 91.7 44.6
27.1 52.1 -16.7 79.2 91.7 76.1 81.8 79.3
```

Then, a session should look like this:

```
Enter filename: nums3.txt
Mean = 41.988
Median = 46.35
Stddev = 40.7124
Negatives = 8%
```

Final example (with file error):

```
Enter filename: nums4.txt
File could not be opened.
```

You will only be submitting the C++ file, stats.cpp. We will test it with several different number files.

Extra credit (5 points): Give one final statistic of the number of whole numbers in the file as a percentage of the total. This should count any number (negative, positive, or zero) that does not contain any decimals.

As an example, given the same nums3.txt file as above, the output would instead be:

```
Enter filename: nums3.txt
Mean = 41.988
Median = 46
Stddev = 40.7124
Negatives = 8%
Whole = 8%
```

Step 3: Compile your Code

This time you'll use a Makefile to compile your code. A Makefile already exists in this folder, with contents as follows:

```
all: sentence stats

sentence: sentence.cpp
	g++ -std=c++11 -Wall sentence.cpp -o sentence

stats: stats.cpp
	g++ -std=c++11 -Wall stats.cpp -o stats

clean:
	rm sentence stats
```

To compile your programs, you can run
`make sentence`
and
`make stats`

You can also just run
`make`
to compile both at the same time.

If you need to delete your executable files, running
`make clean`
will delete them

You can run the executable files as normal:

```
./sentence
./stats
```

If you encounter an error, use the compiler hints and examine the line in question. If the compiler message is not sufficient to identify the error, you can search online to see when the error occurs in general.

Remember to re-compile after you make any changes to your C++ source code.

Step 4: Submit your Code

There is a more detailed (and more general) set of instructions on the course webpage, under Quick References: https://sites.cs.ucsb.edu/~maradowning/cs16/refs/gitbasics.html. If you are new to git/github, I suggest reading through this guide as well when you submit your assignment.

Once you have finished your code (or any time you want to save your code to Github and take a break), first run `git status` to see which files you have modified/created. From there, add any files you want to end up on Github by running the following command:

```
git add <filename>
```

For this lab, most likely you will run two add commands:

```
git add sentence.cpp
git add stats.cpp
```

Please do not add any executable files you created by compiling your code. However, if you do end up accidentally adding them, don't worry about it for now.

Once you've added all the files you want, you will need to commit them. Run:

```
git commit -m "Your commit message here"
```

Please write a descriptive commit message within the quotes, something like "Finished Lab05" if you are done, or something more specific if you're pushing code partway through.

Finally, run:

```
git push
```

to push your changes to Github. Once you've run this command, make sure to go to your repository in your web browser and check that the updated files are there (you might have to reload).

Finally, once you're completely finished with the lab, fill out the Lab05 submission on Gradescope. It only asks you for a list of people you discussed the lab with (remember, discussion of the lab is ok but do not give classmates answers), the number of hours you spent, and has a checkbox to indicate that you're done. I will use this as an indication that your code is pushed to Github and I can begin grading.