ICS-53 Assignment-1

Spring 2022

Problem - 1

Write a program which prompts the user to enter a string and returns the count of all the characters in the string in increasing order of occurrence and lastly, prints the string with the characters sorted in ascending order. The program should first print a “>” symbol as a prompt to the user. The user then enters the string and the program prints the count results. Punctuations, whitespaces and non-printable characters are ignored while processing the input string. Assume that the length of the string is no longer than 256 characters including the null terminating character. (‘\0’). The frequency of characters having the same frequency can be printed in any order. You can assume that the entered string will not have uppercase characters but it still can have punctuations, whitespaces.

The following shows an example where the user enters the string “hello world”.

```
> hello world
h: 1
e: 1
w: 1
r: 1
d: 1
o: 2
l: 3
dehlllloorw
```

Problem - 2

You will need to present your software development process for the solution to Problem 1 by submitting a sequence of screenshots showing each step. You will need to perform the following steps.

- Write a program on your machine
    - You may use any tool on your machine to write the program. I use Notepad++ on a Windows machine, but the tool you use is up to you.
    - Take a screenshot of the window in which you wrote your code.
    - Save the screenshot image with the name “p2_1.XXX”. The XXX will depend on the image format that you use.

- Transfer the program from your machine to openlab

    - You may do this using any secure ftp program that you want to. I use WinSCP on a Windows machine, and sftp (from the terminal) on a Mac.
    - Take a screenshot of the window showing the tool that you are using to transfer your program to openlab.
    - Save the screenshot image with the name “p2_2.XXX”. The XXX will depend on the image format that you use.

- Check the compiler version on openlab (ssh UCNetID@openlab.ics.uci.edu)
    - Do this by typing “gcc -v” at the prompt on openlab.
    - If your version is not 4.8.5 then you will need to fix that.
    - Take a screenshot of the output produced by “gcc -v”
    - Save the screenshot image with the name “p2_3.XXX”. The XXX will depend on the image format that you use.

- Compile your program on openlab
    - Do this by typing “gcc p1.c -o p1”.
    - Take a screenshot of the output produced by “gcc p1.c -o p1”.
    - Save the screenshot image with the name “p2_4.XXX”. The XXX will depend on the image format that you use.

- Execute your compiled program on openlab
    - Do this by typing “./p1”.
    - Take a screenshot of the output produced by “./p1”.
    - Save the screenshot image with the name “p2_5.XXX”. The XXX will depend on the image format that you use.

You should submit a total of 5 screenshot files, one for each step of the process. Each screenshot image can be saved in either .png format, .jpg format, or .gif format. There are many ways to take screenshots of a window and you can use any which you are comfortable with. I use the Snipping Tool on a Windows machine.

Problem - 3

Write a program that prompts the user to enter a string and two integers, `index and length` and prints out a substring based on the 2 integer parameters. The program should first print a “>” symbol as a prompt to the user. All three arguments should be separated by commas. The index argument indicates the index at which the substring starts and the length dictates the length of the substring. If the length of the substring starting from the index exceeds the end of the string, then the string is printed until the end of the string. Assume that the length of the string is no longer than 256 characters including the null terminating character. (‘\0’).

The following example shows the behavior for the string `“I love C Programming”` with `index = 10` and `length = 11`.

```
> I love C Programming, 10, 11
rogramming
```

Problem - 4

Write a program that records the trips made by a taxi for a taxi company. Your program should allow the user to enter a series of commands which can add taxis, indicated by a unique ID, record trips for a particular taxi in terms of miles driven, reset the mile counter for a taxi as well as print the mileage of all taxis in the database. The prompt for the program should print the character '>' At the prompt, the user will type a command followed by the set of arguments associated with the command. The interactive program should exit when the user enters the string "quit". For implementing the database and keeping track of cars/trips, we suggest you use a structure that keeps track of the car ID and miles driven associated with the car. You can assume that at any given time, there will be no more than 10 cars in the system.

Your program should accept the following commands:

- AddCar <CAR ID>: The AddCar command adds a taxi with the unique ID specified by the user. The CAR ID is a unique, non-negative integer. Only one car with a specific CAR ID should exist in the database at a given time.
- AddTrip <CAR ID> <MILES DRIVEN>: The AddTrip command records a trip made by a particular taxi. The CAR ID is a non negative unique integer and MILES DRIVEN is a non negative float. This command adds the number of miles specified to its total miles driven. Users can enter multiple trips for a particular car. If so, the mileage is added to the total mileage of the car.
- Reset <CAR ID>: The Reset command resets the miles driven counter for a particular taxi specified by the ID.
- Display: The Display command prints out all the taxis in the database alongside the total mileage.
- quit: This command quits the program.

The following is an example of execution of the program:

```
> AddCar 7

> AddCar 10

> AddCar 8

> AddCar 10

Error! Car with ID 10 already exists in the database.

> AddTrip 7 14.7

> AddTrip 8 21.9
```

```
> AddTrip 1 1.8

Error! Car with ID 1 doesn’t exist in the database.

> AddTrip 7 5.2

> Display

7    19.9

10   0.0

8    21.9

> Reset 8

> Display

7    19.9

10   0.0

8    0.0

> quit

$
```

Problem - 5

Use the gcov utility to determine the line coverage produced by executing the program that you wrote for Problem 4. Execute the program and supply it with a single input, “AddCar 1”. Use gcov to compute the statement coverage and to produce the “p4.c.gcov” file. Submit the “p4.c.gcov” file.

Submission Instructions

You will upload your solutions to Gradescope. Upload the necessary files described in each problem. Name the files p1.c (Problem 1), p2_1.XXX, p2_2.XXX, ... p2_5.XXX (Problem 2), p3.c (Problem 3), p4.c (Problem 4) and p4.c.gcov (Problem 5). Remember that the “XXX” in the filenames for Problem 2 will actually be replaced by either “.png”, “.jpg”, or “.gif”, depending on which file format you use.

Remember that each C program must compile and execute properly on openlab.ics.uci.edu when it is compiled using the gcc compiler version 4.8.5. The only compiler switches which can be used are - fprofile-arcs, -ftest-coverage (to allow the use of gcov), -std=c99, and -o (to change the name of the executable).