Lab Activity 5

ESE 124 (Programming Fundamentals)

Developed by Dr. Alex Doboli

Extended and edited by Mrs. Jenny Chen

Note*: You will earn 30%* *of your lab grade for completing and submitting your lab work at the end of the lab activity. You must submit the code that you developed for the exercises that you worked on during the lab, and the three associated to-dos. The TODOs should be short. Please don't spend more than 3 minutes on each. You will be graded on your effort during the lab session. You will not be graded on the correctness of the submission.*

The remaining 70% will be earned after submitting your final solutions for the lab exercises, and they will be graded on correctness to the test cases.

For both submissions, each lab exercise is 100 points. The grade for the lab is the average number of points for the submitted exercises.

***The homework exercises** are submitted together with the lab exercises, the latest before the start of your next week's lab session. Each homework exercise is 100 points. The grade for the homework is the average number of points for the submitted exercises.*

Submission: Please follow the instructions posted on Brightspace. Your final solutions are due prior to the start of your next week's lab session.

Lab Activity Description

Midterm exam practice and 1D array processing

A. Part 1 (1 hour 20 minutes)

Practice for the midterm exam

Solve the following three exercises:

1) Devise a C program that determines whether a 1D array of char (**string**) is a sparse array or not. The 1D array has dimensions 15 elements. We define an array to be sparse if its number of letters

(lower case or capitals) is at least twice as large as the number of the rest of the characters in the string. The program reads the strings from an input file. The results re displayed on the screen.

Examples:

Input:

NotSparse$$$$##.

Letters count: **9**

Non-letter count: **6** //which means, there has to be **12 letters** in this array to make this array a "**Sparse array**."

Consequently, this is **not** a Sparse array.

Output:

This is not a sparse string.

Test case:

Input 1:

Hello, this is a test string.

Output 1:

This is a sparse string.

Input 2:

NotSparse$$$$##.

Output 2:

This is not a sparse string.

2) Devise a C program that reads strings from an input file, and then modifies each string by removing all consecutive duplicate characters. The strings with the removed characters are saved in an output file. (e.g., input "Mississippi" is saved as "misisipi" in the output file)

Test case 1:

Input:

The successful entrepreneur books a committee meeting on a Wednesday.

Output:

The sucesful entrepreneur boks a comite meting on a Wednesday.

Test case 2:

Input:

I took a boat trip down the Mississippi River last summer.

Output:

I tok a boat trip down the Misisipi River last sumer.

3) Devise a program in C to display the sum of the series 9 + 99 + 999 + 9999 ... for ***n*** terms, where ***n*** is read from the keyboard. What is the largest ***n*** for which the sum is correctly computed? [This exercise was discussed in a previous lab, however, try now to solve it by yourselves.]

Additional practice exercises:

4) Devise a C program that counts the number of positive, zero, and negative values that are read from an input file. The three counts are displayed on the screen.

5) Devise a C program that reads a number of ***X*** integer numbers from an input file (where ***X*** is always less than 30), and displays on the screen the ***K*** largest numbers in the file. Store the read numbers in a 1 D array of integer values. [Note: 1D arrays are handled in the same way as 1D arrays of char (strings), e.g., we use indexes to access the elements of the vector.]

6) Devise a C program that reads a set of characters from an input file. The program counts the number of occurrences of every letter in the file, and then creates an output file that includes each letter and its number of occurrences. Lower and upper case letters are counted as the same letter, e.g., ‘a’ and ‘A’ are counted together.

Test case 1:

AaBbCcDd

Output:

```
a: 2
b: 2
c: 2
d: 2
```

Test case 2:

This is a simple sentence.

Output:

```
a: 1

c: 1

e: 4

h: 1

i: 3

l: 1

m: 1

n: 2

p: 1

s: 4

t: 2
```

B. Part 2 (1 hour 15 minutes): 1D arrays

1. Devise a C program that reads up to 30 decimal numbers from an input file, and then sorts them in decreasing order. The sorted decimal values are saved in an output file. Perform sorting using the bubble-sort algorithm.

   <u>Steps</u>:
   - TAs will explain the pseudocode of bubble-sort
   - Select the variables of the C program
   - Devise the code to read decimal numbers from an input file
   - Devise the code to write decimal values into an output file
   - Code the pseudocode
   - Complete the C program
   - Test your C program

2. Devise an interactive C program to play the hangman game. The word to be guessed is stored in a string (1 D array of characters), which is read from an input file. The player must guess the word. The program terminates when either all letters have been correctly guessed (the player wins), or when a specified number of incorrect guesses have been made (the computer wins). The maximum number of incorrect guesses is read from the keyboard.
   - **<u>Hint</u>**: Use a second array (let's call it ***guessed***) to keep track of the solution found so far. Initialize all elements of array ***guessed*** as the '*' character. Each time a letter in the word

is correctly found, replace the '*' in array ***guessed*** corresponding to the letter with that letter.

Submit the C code for exercises 1 and 2 of Part C.