Department of Computer Science and Engineering

Assignment 9

Subject : Programming and Data Structure (CS19003)

Instructions:

- Give the name of the programs files as <Your roll>_<assignment number>_<question number>.c. For example, 21XX12345_A1_Q1.c for question 1 of assignment 1 of a student with roll 21XX12345. **Follow the file naming convention strictly**.
- Apart from the main .c file for each program, you should also upload one additional temporary .c file for each program (such as when you have finished half of the code). The naming for the temporary file should be in the format <Your roll>_<assignment number>_<question number>_temp.c. For example, 21XX12345_A1_Q1_temp.c **Make sure that your main code do not deviate much from its temporary code for each program**.
- You should upload the main .c file and the temporary .c file individually to the Moodle course web page once you finish answering each question. No need to zip the files.
- The **deadline** to upload the programs is 12:00PM. Beyond that, submission will be closed (No extensions will be granted).
- If you do not follow the instructions, your marks may be deducted.

[Total Marks = 100 (30 + 30 + 40)]

Answer all questions

1. A dynamic array is an array whose size is only known during the runtime of a program. You can use the following two functions, malloc() or calloc() to define dynamic arrays. **A strict sawtooth array** is an array where the values of the sequence of elements should follow the up-down principle. The successive elements of the array should not be the same. The successive differences across the array should follow alternate +ve and -ve values. For example (1,4,2,5,4,8,6,10,9) and (-10,-15,0,-2,6,3) are strict sawtooth arrays of size 9 and 6, respectively, but (1,5,7,3,6) and (34,23,23,44) are not strict sawtooth arrays. In this program you have to use the concept of dynamic arrays to perform the following tasks with respect to sawtooth arrays.
   - Prompt the user to enter the size of the array as 'n'. Declare a dynamic integer array of the same size.
   - Take input 'n' different integers into the array from the keyboard.
   - Check if the array is a sawtooth array or not. Use the function **int isSawtooth(int *)** that takes as argument a pointer to the array and returns 1 if the array is sawtooth, otherwise it returns 0. Print the appropriate message on the console.
   - If the array is NOT a sawtooth array, then find out the first position in the array where the sawtooth condition is violated. Use the function **void position(int *, int *)** which takes in two pointer arguments, first one is the pointer to the array and the second one a pointer to another integer variable declared in main. The position where the sawtooth condition is violated is written into the variable pointed by the second argument (Note that this function should NOT return the position). Also this function should be called only if the array is not sawtooth.

- The program then frees the dynamic array created. It asks whether the user wants to continue again or not. It prompts the user to input 'Y' (for Yes) or 'N' (for No) to continue the program. If 'Y' is entered, then the steps are repeated again and if 'N' is entered, the program exits. The array has to be dynamically allocated each time the program runs.

Example:

Enter n: 9

Enter the array:

1,4,2,5,4,8,6,10,9

The array is a strict sawtooth array.

Do you want to continue (Y/N): Y

Enter n: 5

Enter the array: 1,5,7,3,6

The array is not a strict sawtooth array.

First violating position: 3

Do you want to continue (Y/N): N **[30]**

2. A user is asked to input two n-dimensional vectors but the dimension is not known initially. We use a structure to define a vector type variable, which contains two elements, an integer number denoting the dimension of the vector the structure is currently storing and a float pointer that points to a dynamically allocated array that stores the vector components corresponding to each of the n dimensions. Take input a number n and correspondingly take input two n-dimensional vectors, storing the dimension and the components of each vector into two corresponding vector type struct variables. Note that before taking input the vector components, remember to initialize the array within the struct variable which will hold the componants. Write the C-functions to perform the following.

   - **struct Vector VectorSum(struct Vector *v1, struct Vector *v2)**: Performs the sum of the two vectors of the same dimension and returns another structure of type vector that holds the sum. For two n-dimensional vectors with components [x1, x2, x3,...,xn] and [y1, y2, y3,...,yn], their sum is another n-dimensional vector [x1+y1, x2+y2, x3+y3,....,xn+yn].
   - **int OrthogonalTest(struct Vector *v1, struct Vector *v2):** Returns 1 if two vectors v1 and v2 are orthogonal, 0 otherwise. Two n-dimensional vectors with components [x1, x2, x3,...,xn] and [y1, y2, y3,...,yn] are said to be orthogonal if their inner product (dot product) is zero, x1.y1+x2.y2+x3.y3+....+xn.yn = 0.

   After taking input the two vectors, first print both the vectors on console in the format **'(n, [.....])'**, where n represents the dimension and put the dimension components within the square brackets. Hence call the above mentioned functions from main() and print the results returned.

   [30]

3. You can put multiple strings within a 2D character array where each row of the character array holds one string. For example char arr[3][100] can hold three strings, each of length of maximum 100 characters and the first string is identified by arr[0], the second string by arr[1] and so

on. Declare a 2D array **string[5][1000]** that can hold 5 strings of maximum 1000 characters each and take input the 5 strings. Each string is a sentence containing any number of words. Words consist of alphabets (you may consider all lowercase characters for simplicity) only and no numericals or special characters, and two words are separated only by one blank space. No need to use fullstop also to mark the end of the string. Hence perform the following operations as defined by the following functions.

- **void characterCount(char *str, int *vowel, int *consonant)** - It counts the total number of vowels and number of consonants of a string pointed by char *str, and writes the vowel count to the address pointed by int *vowel and similarly the consonant count to the address pointed by int *consonant. A space is neither a vowel nor a consonant.
- **int wordCount(char *str)** - It returns the number of words of the string pointed by char *str.
- **void excessWord(char *str1, char*str2)** - This function first calls wordCount() to get to know the number of words of each of the two strings char *str1 and char *str2. Hence it prints the excess words that are present in the longer string. For example if the first string contains 4 words and the second string contains 6 words, then it prints words 5 and 6 of the second string. If both the strings contain equal number of words, then it prints "No excess word".
- **void str_pair_compute(char **arr, int number_of_strings)** - This function takes as input the base address of the entire 2D array containing all the strings and another integer denoting the number of strings within the 2D array. Hence it displays the pairs of strings which contain the same words. The words need not be in the same order. In case no two strings contain the same words, then it prints "Not Applicable". This function may also use wordCount() if required. Consider the example as shown. Suppose the 5 strings in string[][] are,
  - abcd sdfg qwerty oiuyt
  - cfr ojhg nsert
  - evjs irjcl perry
  - nsert cfr ojhg
  - qwerty sdfg abcd oiuyt

  Here, (string[0] and string[4]) and (string[1] and string[3]) contain the same words. So the function displays (1,5) and (2,4) as output, i.e the 1st string and 5th string and 2nd string and 4th string.

From main() after taking the 5 string as input, first call characterCount() for each of the 5 strings to print the vowel and consonant count within main() itself. Declare integer variables as required. Hence take input two numbers between 1 and 5 that represent any two strings and call excessWord() that prints the excess words among the two strings. Note that excessWord() in turn calls wordCount(). It is upto you how you wish to display the different outputs, but it should maintain the specifications as given. Finally call str_pair_compute() to display the pairs of strings which have the same words in different orders. **[40]**
