Section 16

PDS Lab **Lab-8** **20.10.2023**

Instructions:

- This lab is based on the topics: Pointers, Dynamic Memory Allocation
- You should save each program with the file name as specified against each problem as <Lab#>_<Assignment#>_<Roll#>.c. For example, **08_01_23CS10006.c** to save Program to 1st assignment in Lab 8 with Roll Number 23CS10006
- You should upload each program to the Moodle system. Also, copy+paste your programs to the text window on the test page.
- There will be no evaluation and hence grade, if you don't submit your .c files to the Moodle server. Use **emacs** editor and **gcc command** in terminal to run the following programs.
- Document your programs meaningfully using appropriately named variable and sufficient amount of comments. Documentation and proper code indentation carry marks.
- Do not use advanced concepts like structures anywhere in your code.
- **The top two lines of your programs must contain the following information:**

  //Roll No.: <Type in your roll no.>

  //Name: <Type in your name>

Document your programs meaningfully using appropriately named variables and sufficient amount of comments.

1. Write the following text editor program in C by implementing the following functions. Assume that all inputs to the program would be in only lower case characters.

   a) **main:** Define **sptr** as a pointer to an array of string pointers. In an infinite loop, display a menu and prompt the user to enter a choice, a number in the range **[1,7].** Based on the user choice, call the appropriate function as follows: (1→create(), 2→lengthStat(), …, 6→searchReplace(), 7→arrange()) described in the following. Terminate, when the user enters '0'.
   b) **create**: It should take **sptr** a pointer to an array of string pointers as its argument. First ask the user how many words **(< 100)** to enter. Store it in an integer variable **n**. Dynamically allocate an array of size **n** of pointers to strings and store it in **sptr**. Read n words (usually of different lengths) from the keyboard one at a time, dynamically allocate just enough memory for each word entered, and store it so that the next element of **sptr** points to it [Note that each element of **sptr** is a string pointer]. The entered words therefore get sequentially placed in **sptr.** Display all the words that have been entered by the user. **[10]**
   c) **lengthStat**: This function receives **sptr** a pointer to an array of pointers to strings as its argument. Display the number of words that are of lengths: between **1-- 2** letters, between **3 -- 5** letters, and larger than 5 letters. **[5]**
   d) **letterStat**: This function receives **sptr** a pointer to an array of pointers to strings as its argument. It should find and display the number of occurrences of the vowels (**'a' to 'u'**) by considering all the words together. **[5]**
   e) **search:** This function receives **sptr** a pointer to an array of pointers to strings (**sptr**) as its argument. Read a word from the user and check if the word is present in **sptr**. If present, display the sequence number in **sptr** at which it is present. If it is present multiple times, display each occurrence and the sequence number in **sptr** at which the word is present. **[5]**
   f) **removeDuplicate**: This function receives **sptr** a pointer to an array of string pointers (**sptr**) as its argument. It should find all duplicate words, deallocate the duplicates, and display the updated list of words. **[5]**
   g) **searchReplace**: This function receives **sptr** a pointer to an array of string pointers (**sptr**) as its argument. It should next prompt the user and read a word from the user, and check if the word is present in **sptr**. Prompt the user to enter a new word. Replace (deallocate old word and dynamically allocate memory for the new word) the word with the new word. If a word being searched is present at multiple locations, replace only the first occurrence. Display the updated list of words. **[5]**
   h) **arrange:** This function receives **sptr** a pointer to an array of string pointers (**sptr**) as its argument displays the strings (also their lengths) in the order they are stored in **sptr**. It should then rearrange the strings in **sptr** such that each string is either equal to or less than the length of the subsequent string. After the rearrangement, displays the strings (also their lengths) in the order they are stored in **sptr.** **[5]**

[40 Marks]

Note:

1. **The user will always choose create before calling the other functions. Different functions alter the original array, but that is OK**

2. **If you find this problem difficult, you may define a global array of 100 string pointers and name it as sptr. It will also avoid passing the array of strings as parameters to functions. But, dynamically allocate the strings as mentioned in the question. 5 marks penalty for part b) and 1 mark penalty for each other part will apply.**

3. **Those who find dynamic memory allocation and pointer handling too difficult may globally define an array of 10 strings, each of string size at most 20 characters and complete all the functions. 50% penalty will apply for all parts.**