**Birla Institute of Technology and Science, Pilani,**
**First Semester 2022-23**
**CS F211 – Data Structures and Algorithms**
**Lab Test – I (Open Book)**

# SET - PINK

==========================================================================

**05/03/2023** **Max. Marks: 75 M** **Duration: ∞**

==========================================================================

## General Instructions

- This question paper comprises one problem containing various sub-problems, whose details are described on the next page.
- Read all the instructions and the problem statements very carefully before attempting the test.
- Carefully follow the submission instructions mentioned at the end of this document before uploading your solution on the **Dom Judge** portal.
- If you make multiple submissions, only the latest one will be considered for evaluation.
- **It is your responsibility to make sure that you are submitting the right file**. **Also, ensure to save your file before you submit it.**
- **It is your responsibility to make sure that your solution is properly submitted to the Dom Judge portal. After submitting your solution, please download it and verify if it has been correctly uploaded or not. Repeat the submission if not correctly uploaded.**
- **Please get your submission verified by one of the invigilators before leaving the room.**

## Instructions to attempt the test

Create a directory: **DSA_Labtest1_<yourIDNumber> (Example: DSA_LabTest1_2021A7PS1234P)** in your home directory. Create a file: "**LT1_Program_<yourIDNumber>.c**" in the above directory **(Example: LT1_Program_2021A7PS1234P.c)**. You will have to write your entire program within this file only. You will have to create structures and implement the functions as specified in the problem statements. You are free to choose their signatures, however, they should adhere to the specifications given in the problem statements. You are also allowed to declare any number of variables of any kind.

Carefully observe the sample execution shown at the end of this question paper. Your functions should be implemented in such a way that the final compiled code when executed must give output similar to what is shown in the sample execution.

## Marking Scheme

- 50% of the marks for execution and 50% of the marks for presentability and logic.
- Well-documented and readable programs that use good indentation will get a better score for presentability.
- No marks for writing logically correct statements that are out of context.
- If your solution to a question (even fully working) is not adhering strictly to the time and/or space complexities specified, you will be awarded ZERO.

## Problem Statement

Given “file1.txt” containing “**n**” number of English language words, all containing only the lower-case letters. Each line consists of one word only. You will need to create an array (***wordBuckets***) of 26 elements, where each element represents one letter among 26 letters. Each element of the ***wordBuckets*** array stores a bucket (a **SINGLY LINKED LIST**), which stores the words that begin with the letter that the current element represents. You need to store all the words from “file1.txt” inside this data structure. Do the following based on the above description:

1. Define the following structures:
    a. **struct wordsLLHeader:** structure to store the header of the linked list of words
    b. **struct wordsLLNode:** structure to store the node of the linked list of words
    c. **struct record:** Each element of the ***wordBuckets*** array should be of the type **struct record**.

   You should choose the fields of these structures as required by your problem statement. You are free to create additional structure definitions of your choice. ***All structures should be defined immediately after the header inclusions in the global area of the program.*** **5M**

2. Create a function **readData()** that takes the name of the file as an input parameter, creates a **wordBuckets** array, reads the words from the input file line by line and inserts them into their appropriate buckets of the **wordBuckets** array, and returns the **wordBuckets** array. The time complexity of this function should NOT exceed **O(n)**. **16M**.

3. Write a function **findmaxGap()** that takes the **wordBuckets** array as an input parameter, and finds the maximum gap between any two adjacent words stored in the **wordBuckets** array, when they are lexicographically ordered. For example, if the words we have are: ***“ravi”***, ***“kavi”*** and ***“pavi”***, when these words are arranged lexicographically they would be in the order: ***“kavi”***, ***“pavi”*** and ***“ravi”***. Gap between two words say ***“ravi”*** and ***“kavi”*** can be computed as |ascii(***r***) – ascii(***k***)|. *MaxGap* is the maximum of the gaps computed for every adjacent word pairs in the lexicographically arranged list of words. For the above example, *MaxGap* is maximum of gaps computed between ***“kavi”*** & ***“pavi”***, and ***“pavi”*** & ***“ravi”***. The time complexity of this function should NOT exceed **O(n). 10M**

4. Create a function **partitionLists()** that takes the **wordBuckets** array as an input parameter, and partitions each linked list into two partitions:
    a. the first (or the left) partition contains all the words that have their ***third character*** ∈ **[a .. m]**
    b. the second (or the right) partition contains all the words that have their ***third character*** ∈ **[n .. z]**

   You can assume that the input file has words of length >= 3 only. This function should not return anything. The time complexity of this function should NOT exceed **O(n)**. **16M**

5. Create a function **printData()** that takes the **wordBuckets** array as an input parameter, and prints the words stored in it. It essentially traverses the linked lists from ‘a’ to ‘z’ and prints the words stored in them. This function should not return anything. Follow the sample execution given on the next page. **6M**

6. Create a function **mergeSortBuckets()** that takes the **wordBuckets** array as an input parameter, and lexicographically orders each linked list stored in it using ***iterative merge sort***. Your algorithm should use auxiliary linear space only for the merging step. This function should not return anything. **16M**

7. Create the function **main()**, which invokes the above functions to accomplish the following tasks in the order given below: **6M**
    a. Read data from “file1.txt” and populate the data in the **wordBuckets** array
    b. Print the maximum gap in the **wordbuckets** array
    c. Partitions the lists of **wordBuckets** array
    d. Prints the **wordBuckets** array
    e. Creates lexicographical ordering within the buckets of **wordBuckets** array
    f. Prints the **wordBuckets** array

## Sample Execution of the Code

If “file1.txt” consists of the following words:

radi
kavi
pavi
rani
navi
rasi
mavi

Your program execution would print something like this:

```
$ ./a.out file1.txt

Maximum gap is: 2

Printing wordBuckets array after partitioning:
kavi
mavi
navi
pavi
radi
rasi
rani

Printing wordBuckets array after lexicographical ordering:
kavi
mavi
navi
pavi
radi
rani
rasi
```

## Submission Instructions

The directory **labtest_<yourIDNumber>** now contains “**LT1_Program_<yourIDNumber>.c**”. Please upload this file on the portal as your submission. You can also make multiple submissions of the same file. After submitting your solution, please download it and verify if it has been correctly submitted or not. Repeat the submission if not correctly uploaded. Please get your submission verified by one of the invigilators before leaving the room.

=== End of document===