CS 6103D Software Systems Laboratory

PROBLEM 1C **13-AUG-2019**

The objective is to learn the following:

- Implementation of binary search tree using pointers
- implementation of stack using pointers (as a singly linked list)
- implementation of priority queue using heap

Submission date: on or before 19.08.2019 Monday 1:00 pm
Submission: a single file named as per the following format

- Submit as a single .tar file
- The name of this file must be *P1C _ < FIRSTNAME > < ROLLNO >.tar*
- (*eg* : *P1C_ARUN_M*190*xxxCS.tar*)

Modify the program developed for problem 1B as follows.

1. Implement the *regList* of each course using a *Binary Search Tree (BST)*. The field *RegList* in a course struct is now a pointer to the root of a *BST*. Each node should contain name, and pointers to its left child, right child and parent. Define functions *insert(x, t)*, to insert name x to the tree t (t is a pointer to the root of the tree), *delete(x, t)* to delete name x from tree t, and *treeWalk(t)*, a recursive function for doing the traversal of t.

   (1) Define a function *printRegListASC(c)*, which given a course code prints the names of students registered in that course in sorted order, by invoking *treeWalk(t)*.
   (2) Define a function *printRegListDESC(c)*, which given a course code prints the names of students registered in that course in reverse sorted order, by implementing new *treeWalkDESC(t)*.
   (3) Define a function *printRegListN(c, n , w)*, which given a course code and no of students prints the names of top or bottom N students registered in that course by implementing new *treeWalkN(t,n)*.(w=0 for top and w=1 for bottom)

   Define each *BST* operation as per the algorithms given in chapter 12 of CLRS(reference given below).

2. Provide a non recursive version of *treeWalk(t) for all above 3 cases*. This requires a stack of pointers to tree nodes. Implement this *stack* using an array. Define operations *push(S, x)* to add an element x to the top of the stack S, *pop(S)* to pop out the top most element from stack S and *isEmpty(S)* which returns true if the stack S is empty and false otherwise.

3. Maintain the *waitList* as a *max priority queue*. Each student entering the queue is given a priority value ranging from 1 to *maxLimit* where *maxLimit* is the maximum number of students allowed in the course. Implement this *priority queue* using a *maxHeap*. Define operations *insert(Q, x)* to insert an element x to the *priority queue Q*, *extract Max(Q)* to remove and return the element with the highest priority value from Q, *increaseKey(Q, x, k)* to increase the priority value of element x in Q to the new value k (new value is assumed to be at least as large as the current priority value of x). *decreaseKey(Q, x, k)* to decrease the priority value of element x in Q to the new

value k (new value is assumed to be at least as small as the current priority value of x).Implement changeCourse(old,new,x) to change elected course from old to new (Perform deleteKey(Q, x) for old course waitlist and insert(Q, x) for new course waitlist) .deleteKey(Q, x) to delete specific element from maxheap.
Each heap operation is to be implemented as per the algorithms given in section 6.5 of CLRS(reference given below).

- **Reference**: T. H. Cormen, C. E. Lieserson, R. L. Rivest, C. Stein. Introduction to Algorithms, PHI Learning, 3rd edition, 2010.