BPDC, Dubai – II Semester, 2020-2021

Course No: CS F211 **LAB EXERCISE 3** **Course Title: DSA**

Date: Feb 15 2021

1. Implement subroutines that deploys the basic operations (prototype given below) of a queue. You are provided with a program queue.c that tabulates the average running time per operation (enqueue/dequeue) in a file queueTime.txt for each of those test cases provided with. You may assume the queue only to deal with positive integers.

The dynamically allotted array *queue*[] refers to the memory allotted for the abstract data type queue along with the corresponding index variables *front* and *rear*, already being defined and initialized to 0. The size of the array is in variable *queue_size*. You may use the same in the implementations of the functions below.

Int size() //returns the total number of elements in the queue

int isQueueEmpty() //returns 1 of the queue is empty and 0 otherwise

int isQueueFull() //returns 1 if the queue is full and 0 otherwise

enQueue(x) //enqueues the value x to the rear end of the queue, returns -1 if in case the queue is full

deQueue() //dequeues the value x from the front end of the queue, returns -1 if in case the queue is empty

2. Write a menu driven program (in C or C++) which provides the user with 3 options, EnQueue, DeQueue, and Exit, maintaining a Queue in the background. Obviously, once the user opts to enqueue, should further be prompted for the value to be enqueued. In addition, you are supposed to implement the queue using two stacks, say stack1[] and stack2[], exploiting the fact that once we reverse a list (using a stack) twice, we do have the elements in the original order as it is. You should implement the pseudocode given below. Here stack2[] holds the values in the queue with the rearmost at the top, and we use stack1[] as a buffer to perform the required double-reversal. Further, stack1[] and stack2[] are of the same size. You have to implement the necessary supplementary functions, further, deal with the corner cases too.

stack1[] //memory allotted for stack1
top1 // index of the topmost variable in stack1
stack2[] //memory allotted for stack2

top2 // index of the topmost variable in stack2

isStack2Full()// returns 1 if stack2[] is full and 0 otherwise
isStack2Empty()//returns 1 if stack2[] is empty and 0 otherwise
isStack1full()// returns 1 if stack1[] is full and 0 otherwise
isStack1Empty()//returns 1 if stack1[] is empty and 0 otherwise
pop2() //pops and returns an element from stack2
push2(x)//pushes an element to stack2
pop1()//pops an element from stack1
push1(x)//pushes an element to stack1

```
Procedure isQueueFull()
        if isStack2Full() = 1 then
                return 1
        else
                return 0

Procedure isQueueEmpty()
        if isStack2Empty() = 1 then
                return 1
        else
                return 0


Procedure enQueue (x)
        if(isQueueFull())  then
                return -1
        push2(x) //push to the second stack
        return 1

Procedure deQueue()
        if(isQueueEmpty())
                return -1
        while isStack2Empty() = 0 do //stack2 is not empty
                x←  pop2()
                push1(x)
        y← pop1()
        while isStack1Empty() = 0 do //stack2 is not empty
                x←  pop1()
                push2(x)
        return y
```