NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

CL 218–DATA STRUCTURES LAB

Lab Session 06

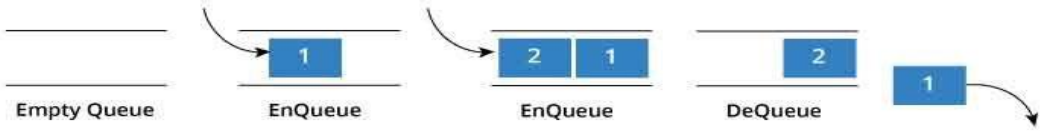
Queues

Instructors: Mubashra Fayyaz , Aqsa Zahid

Outline

- Queues
- Limitation

A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.
Queue follows the **First In First Out(FIFO)** rule - the item that goes in first is the item that comes out first too.



In the above image, since 1 was kept in the queue before 2, it was the first to be removed from the queue as well. It follows the FIFO rule.
In programming terms, putting an item in the queue is called an "enqueue" and removing an item from the queue is called "dequeue".
We can implement queue in any programming language like C, C++, Java, Python or C#, but the specification is pretty much the same.

Queue Specifications:

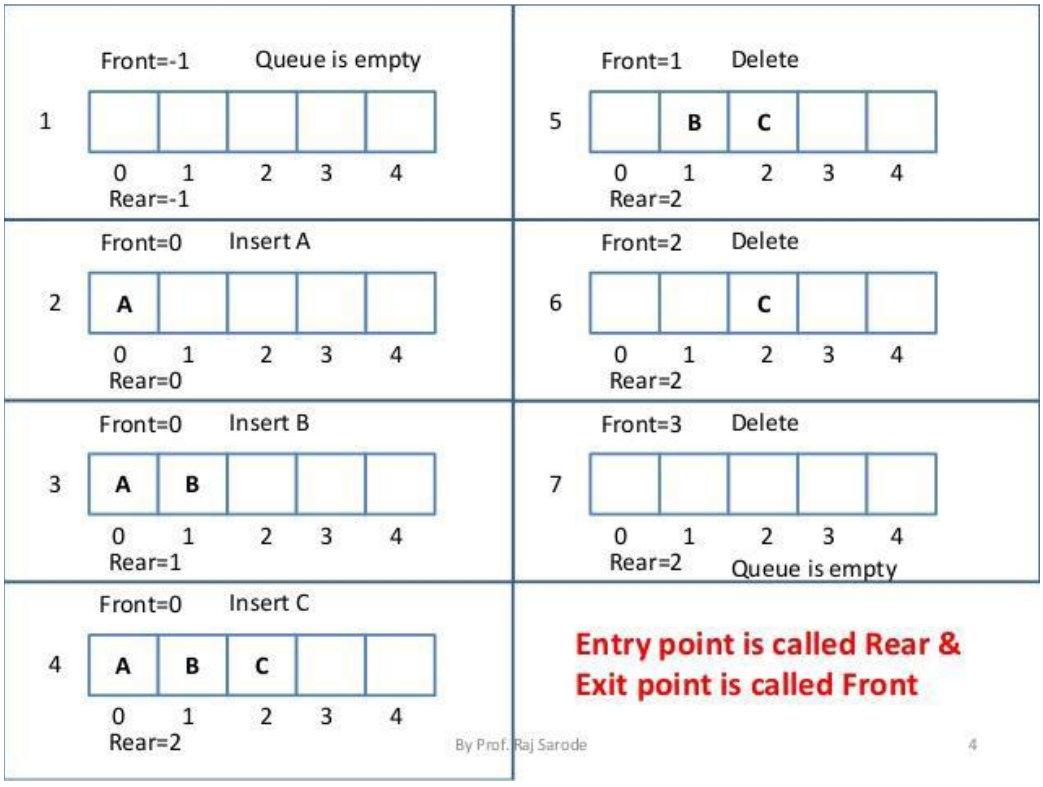
A queue is an object or more specifically an abstract data structure(ADT) that allows the following operations:

- Enqueue: Add element to end of queue
- Dequeue: Remove element from front of queue
- IsEmpty: Check if queue is empty
- IsFull: Check if queue is full
- Peek: Get the value of the front of queue without removing it

How Queue Works:

Queue operations work as follows:

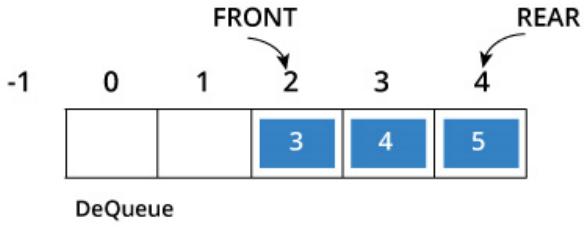
1. Two pointers called FRONT and REAR are used to keep track of the first and last elements in the queue.
2. When initializing the queue, we set the value of FRONT and REAR to -1.
3. On enqueing an element, we increase the value of REAR index and place the new element in the position pointed to by REAR.
4. On dequeueing an element, we return the value pointed to by FRONT and increase the FRONT index.
5. Before enqueing, we check if queue is already full.
6. Before dequeuing, we check if queue is already empty.
7. When enqueing the first element, we set the value of FRONT to 0.
8. When dequeing the last element, we reset the values of FRONT and REAR to -1.



```
#include<stdio.h>
#define SIZE 5
voidenQueue(int);
voiddeQueue();
voiddisplay();
int items[SIZE], front = -1, rear = -1;
intmain()
{
//deQueue is not possible on empty queue
deQueue();
//enQueue 5 elements
enQueue(1);
enQueue(2);
enQueue(3);
enQueue(4);
enQueue(5);
//6th element can't be added to queue because queue is full
enQueue(6);
display();
//deQueue removes element entered first i.e. 1
deQueue();
//Now we have just 4 elements
display();
return0;}

voidenQueue(int value){
if(rear == SIZE-1)
printf("\nQueue is Full!!");
else {
if(front == -1)
            front = 0;
        rear++;
        items[rear] = value;
printf("\nInserted -> %d", value);
    }}
voiddeQueue(){
if(front == -1)
printf("\nQueue is Empty!!");
else{
printf("\nDeleted : %d", items[front]);
        front++;
if(front > rear)
            front = rear = -1;
    }}
voiddisplay(){
if(rear == -1)
printf("\nQueue is Empty!!!");
else{
inti;
printf("\nQueue elements are:\n");
for(i=front; i<=rear; i++)
printf("%d\t",items[i]);
    }}
```

Limitation:
As you can see in the image below, after a bit of enqueueing and dequeueing, the size of the queue has been reduced.



The indexes 0 and 1 can only be used after the queue is reset when all the elements have been dequeued.
By tweaking the code for queue, we can use the space by implementing a modified queue called circular queue.

LAB TASK

Question 1:
Suppose you have a queue D containing the numbers (1, 2, 3, 4, 5, 6,7,8), in this order. Suppose further that you have an initially empty queue Q. Give a programming code of a method that uses only D and Q (and no other variables or objects) and results in D storing the elements (1,2, 3,5,4, 6, 7,8), in this order.

Question 2:
Design and implement a MinDeque data structure that can store comparable elements and supports all the deque operations add first(x), add last(x) remove first(), remove last() and size(), and the min() operation, which returns the minimum value currently stored in the data structure.

Question 3:

Given a stream of characters and we have to find first non repeating character each time a character is inserted to the stream.

Input : a a b c

Output : a -1 b b

Input : a a c

Output : a -1 c

Question 4: Circular Queue

Suppose there is a circle. There are n petrol pumps on that circle. You are given two sets of data.

- The amount of petrol that every petrol pump has.
- Distance from that petrol pump to the next petrol pump.

Calculate the first point from where a truck will be able to complete the circle (The truck will stop at each petrol pump and it has infinite capacity). Expected time complexity is O(n). Assume for 1-litre petrol, the truck can go 1 unit of distance.

For example:

let there be 4 petrol pumps with amount of petrol and distance to next petrol pump value pairs as {4, 6}, {6, 5}, {7, 3} and {4, 5}. The first point from where the truck can make a circular tour is 2nd petrol pump. Output should be "start = 1" (index of 2nd petrol pump).