DHA Suffa University
Department of Computer Science
CS 2001L – Data Structures and Algorithms Lab
Spring 2021

Lab 08 – Queues

Objective:

To learn about

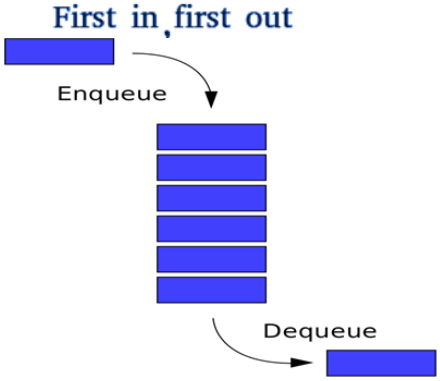
- Queue Operations
- Queue Variations

Queues

Queues are data structures that, like the stack, have restrictions on where you can add and remove elements. To understand a queue, think of a cafeteria line: the person at the front is served first, and people are added to the line at the back. Thus, the first person in line is served first, and the last person is served last. This can be abbreviated to First In, First Out (FIFO).

The cafeteria line is one type of queue. Queues are often used in programming networks, operating systems, and other situations in which many different processes must share resources such as CPU time.

In queues insertions are permitted at one end – called the **rear**, and deletions are permitted from the other end – called the **front**. This means that the removal of elements from a queue is possible in the same order in which the insertion of elements is made into the queue.



Operations on Queue:

Mainly the following basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is not full, this function adds an element to the back of the queue, else it prints "OverFlow".

```
enqueue(int queue[], int element, int&rear, int arraysize)
if(rear==arraysize)
      Print("overflow")
else
      queue[rear]=element
      rear++
```

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

```
dequeue(int queue[], int& front, int rear)
if(front==rear=-1)
      Print("Underflow")
else
      queue[front]=0
      front++
```

Queue variations

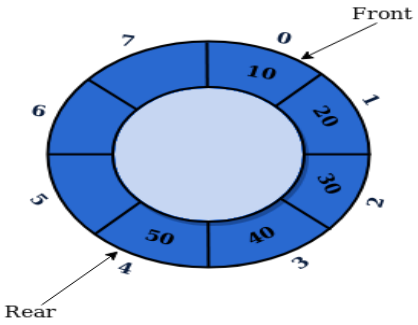
The standard queue data structure has the following variations:

1. Double-ended queue
2. Circular queue
3. Priority queue

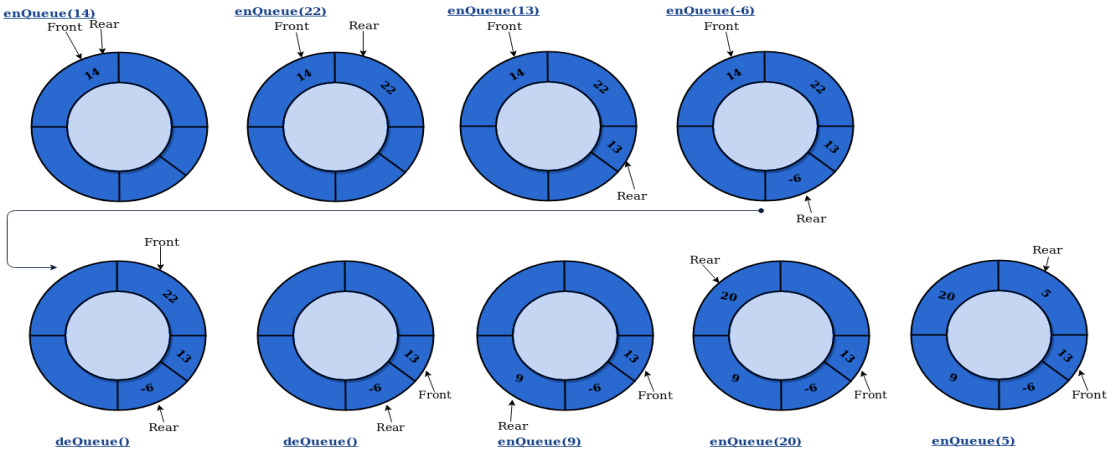
We are going to discuss Circular Queue implementation in detail.

Circular Queue

Circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **'Ring Buffer'**.



In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we cannot insert the next element even if there is a space in front of queue.



Operations on Circular Queue:

Enqueue

```
enqueue(int item)

    if(isFull())              // Queue is full

            print "OverFlow"

    else
```

```
        pos = (front+size)%capacity
        array[pos] = item
        size = size + 1
```

Dequeue

```
dequeue() {
    if(isEmpty())            // Queue is empty
        print "UnderFlow"
    else
        item = array[front]       // Delete the front element
        front = (front + 1)%capacity
        size = size - 1
        return item
```

Front

```
front(int queue[], int front)
    return array[front]
```

Size()

```
size()
    return size
```

IsEmpty()

```
isEmpty()
    return (size == 0);
```

isFull()

```
isFull ()
    return (size == capacity);
```

LAB ASSIGNMENT

1. Implement queue using Arrays. Make proper functions for each operation.

2. Implement linear queue using Linear Linked list. Make proper functions for each operation.

3. Implement Circular queue using Circular Linked list. Make proper functions for each operation.

SUBMISSION GUIDELINES

- Take a screenshot of each task (code and its output), labeled properly.
- Place all the screenshots and the code files (properly labeled) in a single folder labeled with Roll No and Lab No. e.g. **'cs191xxx_Lab01'**.
- Submit the folder at LMS
- **-100%** policies for plagiarism.