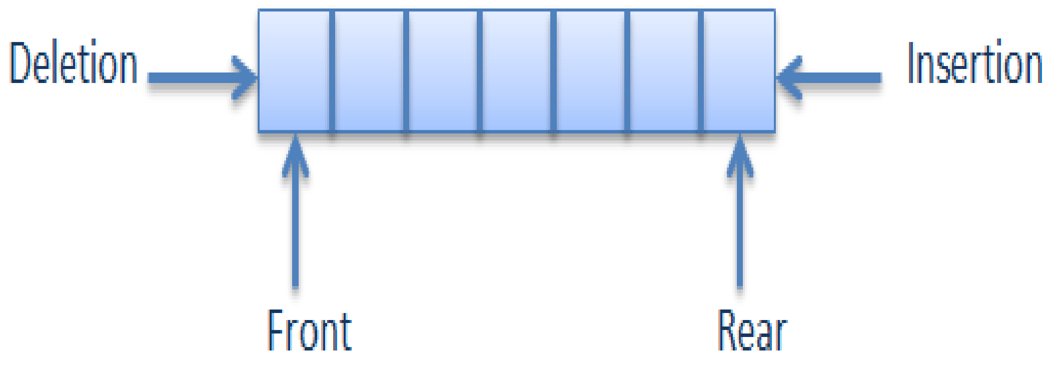
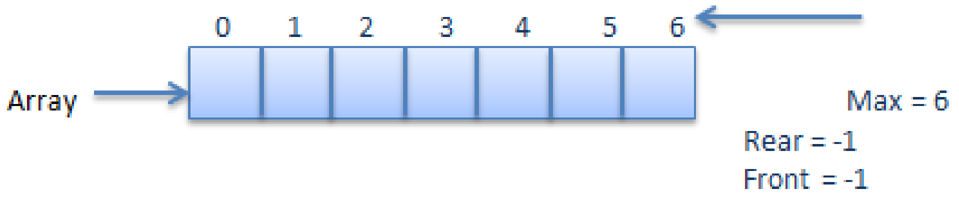
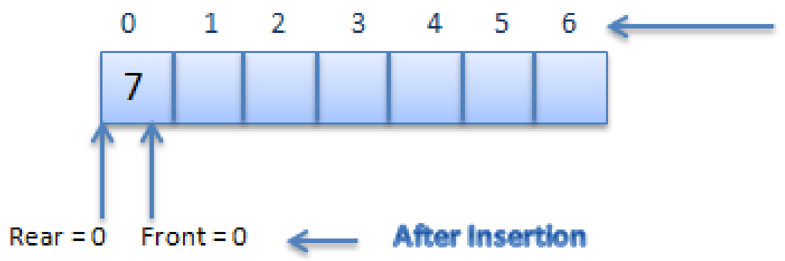
Queues

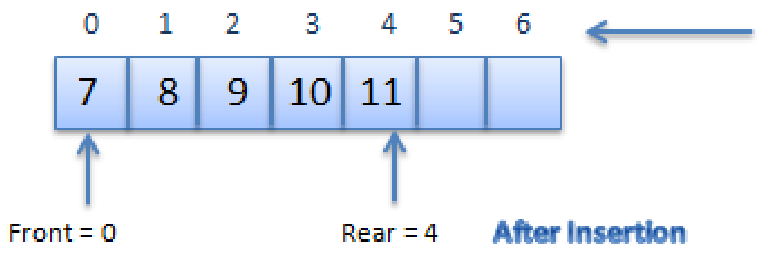
- Queues is a kind of abstract data type where items are inserted one end (rear end) known as **enqueue** operation and deteted from the other end(front end) known as **dequeue** operation.
- This makes the queue a **First-In-First-Out (FIFO)** data structure.
- The queue performs the function of a buffer.

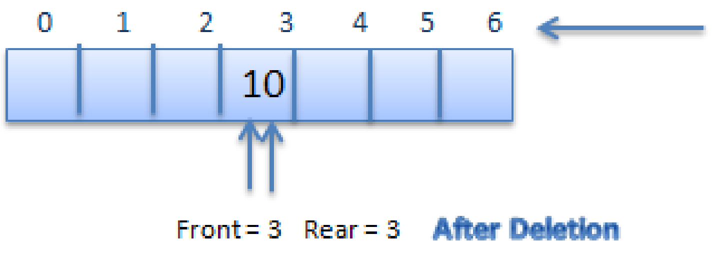


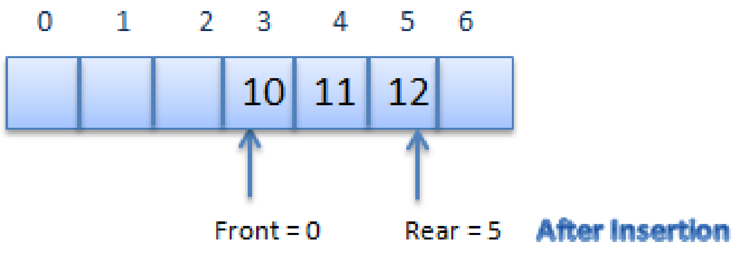
Operation on Queues











Application of queues

Queues are mostly used in operating systems.

- Waiting for a particular event to occur.
- Scheduling of processes.

Algorithm for Insertion in Queue

```
Insert ( )
1. If (REAR == N) Then [Check for overflow]
2.             Print: Overflow
3. Else
4.             If (FRONT and REAR == 0) Then [Check if QUEUE is empty]
                       (a) Set FRONT = 1
                       (b) Set REAR = 1
5.                     Else
6.                               Set REAR = REAR + 1 [Increment REAR by 1]
                                 [End of Step 4 If]
         7.                      QUEUE[REAR] = ITEM
         8.                      Print: ITEM inserted
               [End of Step 1 If]
         9.    Exit
```

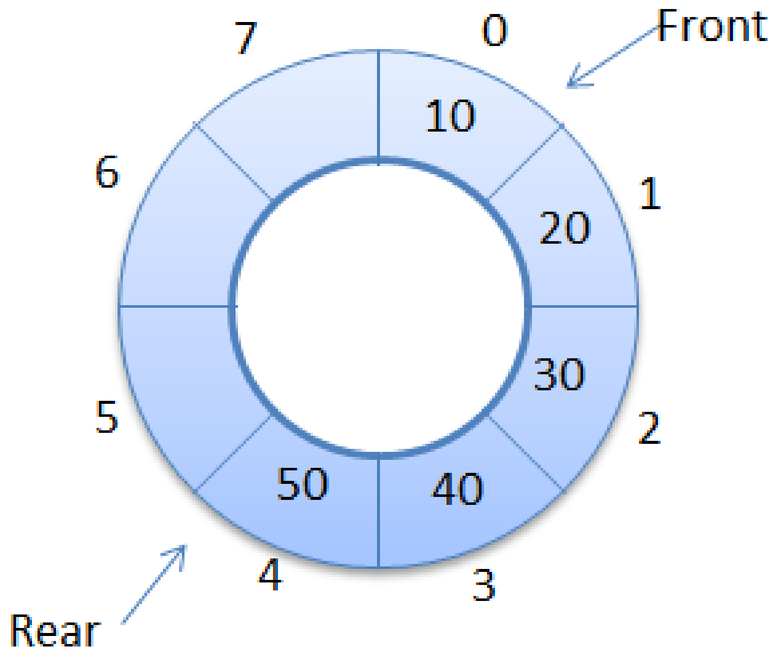
Algorithm for Deletion in Queue

```
Delete ( )
1. If (FRONT == 0) Then [Check for underflow]
2.             Print: Underflow
3. Else
4.             ITEM = QUEUE[FRONT]
5.             If (FRONT == REAR) Then [Check if only one element is left]
                       (a) Set FRONT = 0
                       (b) Set REAR = 0
6.             Else
         7.                      Set FRONT = FRONT + 1 [Increment FRONT by 1]
                       [End of Step 5 If]
         8.            Print: ITEM deleted
               [End of Step 1 If]
         9. Exit
```

Circular Queue

- A circular queue is an abstract data type that contains a collection of data which allows addition of data at the end of the queue and removal of data at the beginning of the queue. Circular queues have a fixed size.
-

  Circular queue follows FIFO principle. Queue items are added at the rear end and the items are deleted at front end of the circular queue.



Algorithm for Insertion in a circular queue

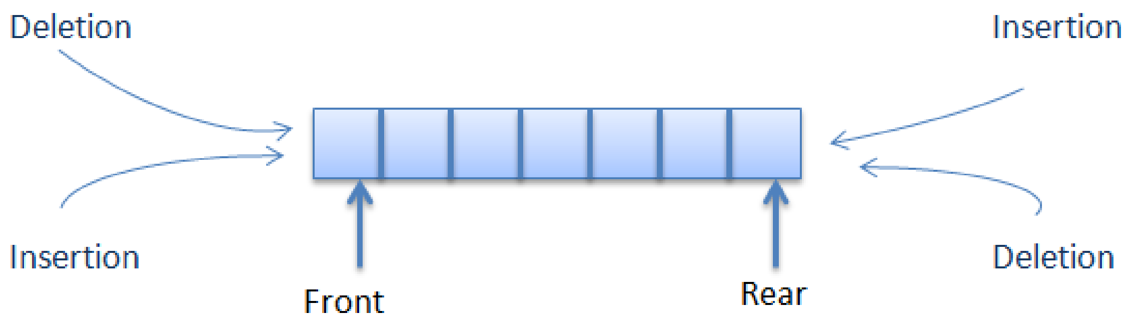
```
Insert CircularQueue ( )
1. If (FRONT == 1 and REAR == N) or (FRONT == REAR + 1) Then
2.      Print: Overflow
3. Else
4.      If (REAR == 0) Then [Check if QUEUE is empty]
                (a) Set FRONT = 1
                (b) Set REAR = 1
5. Else If (REAR == N) Then [If REAR reaches end if QUEUE]
6.              Set REAR = 1
        7. Else
        8.              Set REAR = REAR + 1 [Increment REAR by 1]
            [End of Step 4 If]
        9. Set QUEUE[REAR] = ITEM
        10. Print: ITEM inserted
        [End of Step 1 If]
        11. Exit
```

Algorithm for Deletion in a circular queue

```
Delete CircularQueue ( )
1. If (FRONT == 0) Then [Check for Underflow]
2.      Print: Underflow
3. Else
4.      ITEM = QUEUE[FRONT]
5.              If (FRONT == REAR) Then [If only element is left]
                        (a) Set FRONT = 0
                        (b) Set REAR = 0
6. Else If (FRONT == N) Then [If FRONT reaches end if QUEUE]
        7.                      Set FRONT = 1
        8. Else
        9.                      Set FRONT = FRONT + 1 [Increment FRONT by 1]
                        [End of Step 5 If]
        10. Print: ITEM deleted
                        [End of Step 1 If]
        11. Exit
```

Double-Ended Queue

A double-ended queue is an abstract data type similar to an simple queue, it allows you to insert and delete from both sides means items can be added or deleted from the front or rear end.



Algorithm for Insertion at rear end

```
Step -1: [Check for overflow]
        if(rear==MAX)
                Print("Queue is Overflow”);
         return;
Step-2: [Insert element]
         if rear=0 && front=0
                         rear=1;
                         front=1;
                 else
                         rear=rear+1;

                         q[rear]=no;
                         [Set rear and front pointer]

        Step-3: return
```

Algorithm for Insertion at front end

```
Step-1 : [Check for the front position]
        if(front<=1)
                Print ("Cannot add item at front end");
        return;
Step-2  : [Insert at front]
        else
                front=front-1;
                q[front]=no;
Step-3  : Return

```

Algorithm for Deletion from front end

```
Step-1 [ Check for front pointer]
        if front=0
                print(" Queue is Underflow");
        return;
Step-2 [Perform deletion]
        else
                no=q[front];
                print("Deleted element is",no);
                [Set front and rear pointer]
                if front=rear
                        front=0;
                        rear=0;
                else
                        front=front+1;
        Step-3 : Return
```

Algorithm for Deletion from rear end

```
Step-1 : [Check for the rear pointer]
        if rear=0
                print("Cannot delete value at rear end");
        return;
Step-2: [ perform deletion]
        else
                no=q[rear];
                [Check for the front and rear pointer]
        if front= rear
                        front=0;
                        rear=0;
                else
                        rear=rear-1;
                        print("Deleted element is",no);

Step-3 : Return
```