
# Data Structure-Queue

Kaustuv Bhattacharjee
University of Engineering & Management, Kolkata

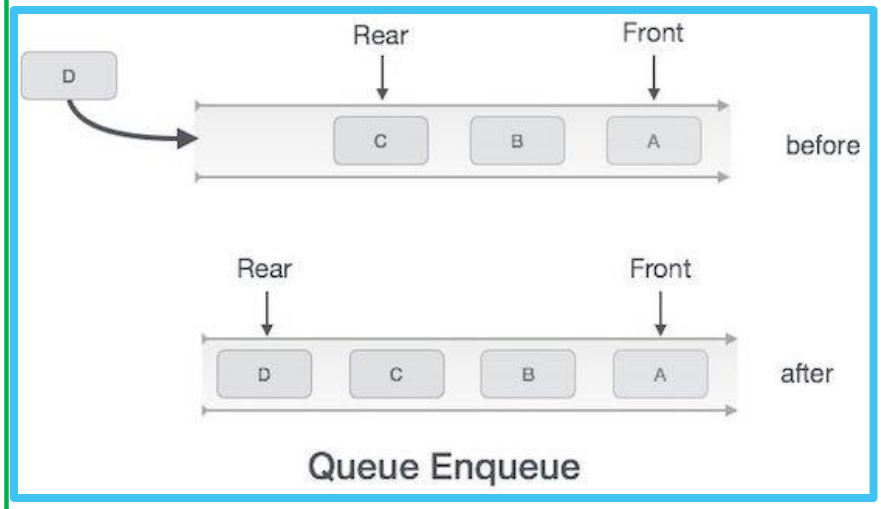
# Queue-Introduction

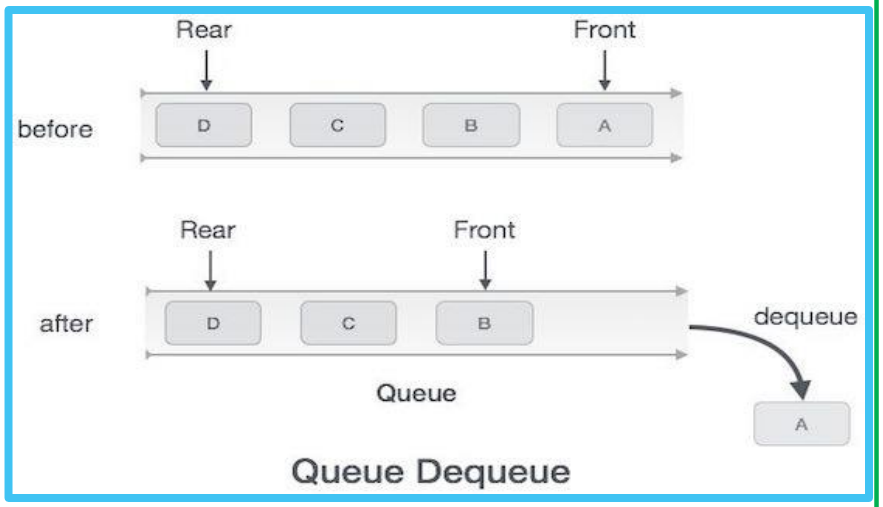
- Queue is an ordered list in which, insertion operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.
- Queues are sometimes called as First-In-First-Out (FIFO) lists i.e. the element which is inserted first in the queue, will be deleted first from the queue.

# Queue–Introduction

- Queue can be implemented using either arrays or linked lists
- There are three operations which can be performed on queue:
  - enqueue: Adding an element to the queue (from REAR)
  - dequeue: Removing an element from the queue (from FRONT)
  - Peep: Returns the value of the element at the FRONT without removing it

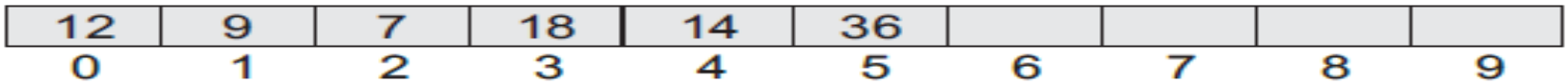
# Queue-Pictorial Representation



# Array Representation of Queue

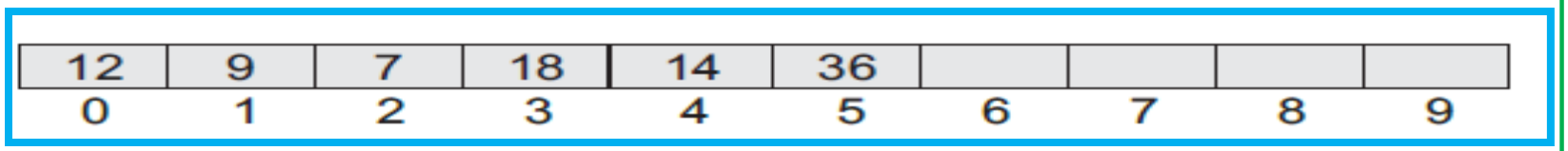
- Queue can be represented as a *linear array*
- Every queue has a variable called *FRONT*, which is used to delete element from queue
- Every queue has a variable called *REAR*, which is used to insert element in a queue
- *REAR=MAX-1* indicates queue is full, thus no elements can be inserted
- *FRONT=-1* or *FRONT>REAR* indicates queue is empty, thus deletion not possible

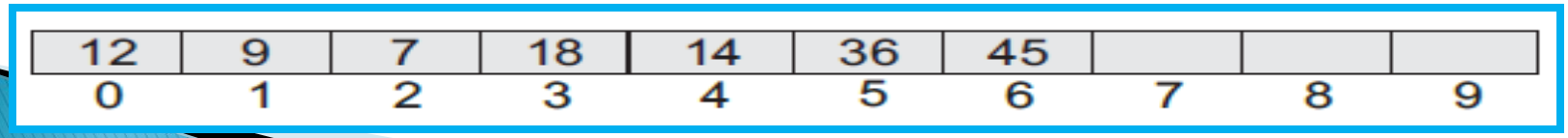


# Operations on Queue–Enqueue

## Enqueue operation

- Used to insert an element into the queue
- The new element is added at the end of the REAR position of the queue
- Before inserting the value, first check if REAR=MAX-1, indicating queue OVERFLOW (queue is full)



# Operations on Queue-Enqueue Contd…

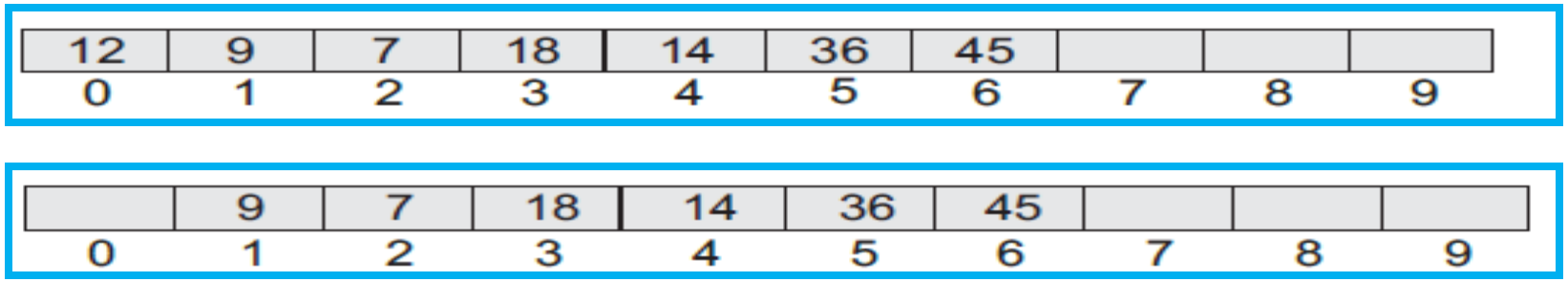
- Algorithm/Steps for Enqueue operation (using Array)

```
Step 1: IF REAR = MAX-1
        PRINT OVERFLOW
        GO TO STEP 4
   [END OF IF]
Step 2: IF FRONT=-1 and REAR=-1
                SET FRONT = REAR = 0
            ELSE
                SET REAR=REAR+1
         [END OF IF]
Step 3: SET QUEUE[REAR] = VALUE
Step 4: END
```

# Operations on Queue-Dequeue

## Dequeue operation

- Used to delete the element from the FRONT of the queue
- Before deleting the value, first check if FRONT=-1 and REAR=-1, indicating queue UNDERFLOW (queue is empty)



# Operations on Queue-Dequeue Contd…

- Algorithm/Steps for Dequeue operation (using Array)

```
Step 1: IF FRONT = -1 or FRONT > REAR
            PRINT UNDERFLOW
         GO TO Step 2
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
      [END OF IF]
Step 2: END
```

# Queue using Array: Assignments

Assignments:

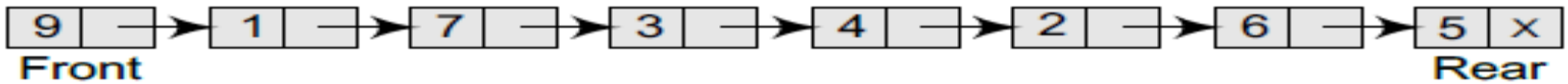
1. Write a program to insert an element into the queue using an array (Enqueue Operation).
2. Write a program to delete an element from the queue using an array (Dequeue Operation).
3. Write a program to return the value of the FRONT element of the queue(without deleting it from the queue) using an array (Peep operation).
4. Write a program to display the elements of a queue using an array.

# Queue using Array: Limitations

- Array must be declared to have some fixed size
- If queue is very small one or its maximum size is known in advance, then the array implementation of the queue gives an efficient implementation
- It is difficult to implement queue using array if the array size can't be determined in advance

# Queue using Linked List

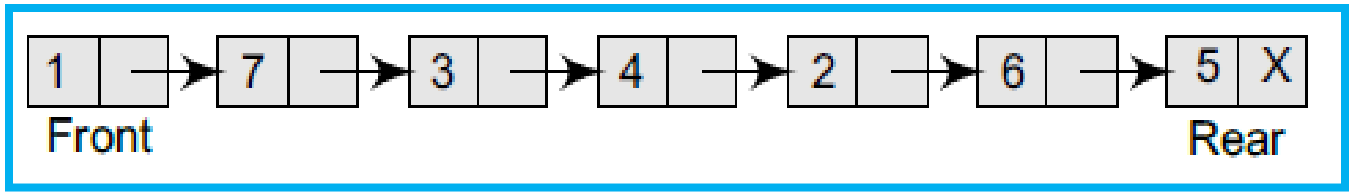
- Queue can be represented using linked list
- Every linked list node has two parts – one that stores data and another that stores the address of the next node
- The START pointer of the linked list is used as FRONT
- Another pointer called REAR, will be used to store the address of the last element in the queue.
- All insertions will be done at the REAR end and all the deletions will be done at the FRONT end.
- FRONT=REAR=NULL indicates queue is empty

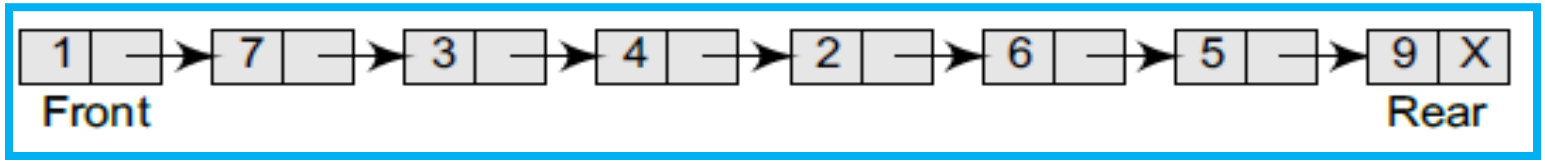


# Operations on Linked Queue- Insert

## Insert operation

- Used to insert an element into the queue
- The new element is added as the last element of the queue



# Operations on Linked Queue- Insert Contd…

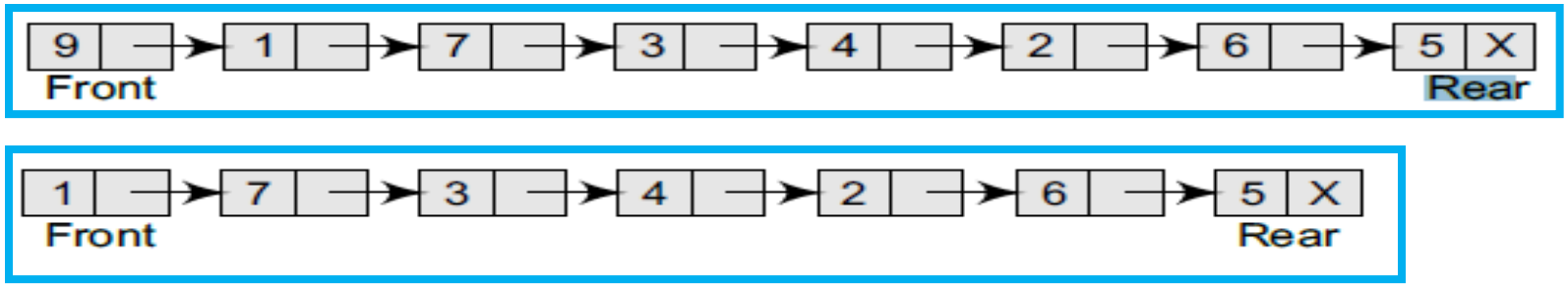
- **Algorithm/Steps for Insert operation (using Linked List)**

Step 1: Allocate memory for new node , name it as PTR
Step 2: SET PTR ->DATA = VAL
Step 3: IF FRONT = NULL
SET FRONT=REAR=PTR
SET FRONT->NEXT=REAR->NEXT=NULL
ELSE
SET REAR->NEXT=PTR
SET REAR=PTR
SET REAR->NEXT=NULL
[END OF IF]
Step 4: Stop

# Operations on Linked Queue- Delete

## Delete operation

- Used to delete the element that is first inserted in a queue
- Before deleting the value, first check if FRONT=NULL, indicating queue UNDERFLOW (queue is empty)



# Operations on Linked Queue- Delete Contd…

- Algorithm/Steps for Delete operation (using Linked List)

Step 1: IF FRONT = NULL
    PRINT UNDERFLOW
    GO TO STEP 5
  [END OF IF]
Step 2: SET PTR=FRONT
Step 3: SET FRONT=FRONT->NEXT
Step 4: FREE PTR
Step 5: END

# Queue using Linked List: Assignments

Assignments:

1. Write a program to insert an element into the queue using linked list (Insert Operation).
2. Write a program to delete an element from the queue using linked list (Delete Operation).
3. Write a program to return the value of the front element of the queue (without deleting it from the queue) using linked list (Peep operation).
4. Write a program to display the elements of a queue using linked list .

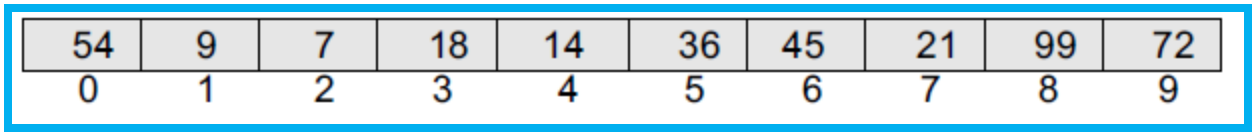
# Types of Queues

- Circular Queue
- Deque
- Priority Queue
- Multiple Queue

# Circular Queue

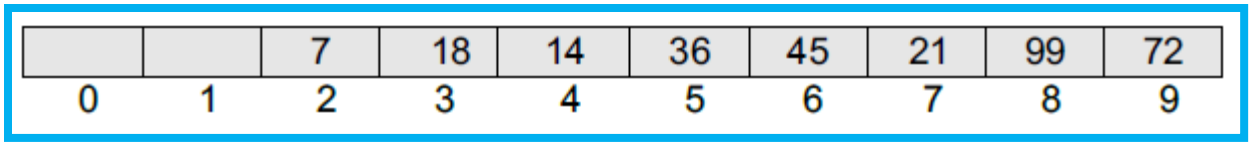
## Limitations of Liner Queue

- Consider the scenario given below:



| 54 | 9 | 7 | 18 | 14 | 36 | 45 | 21 | 99 | 72 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- In the above case FRONT=0, REAR=9; thus no new elements can be inserted as the queue is full

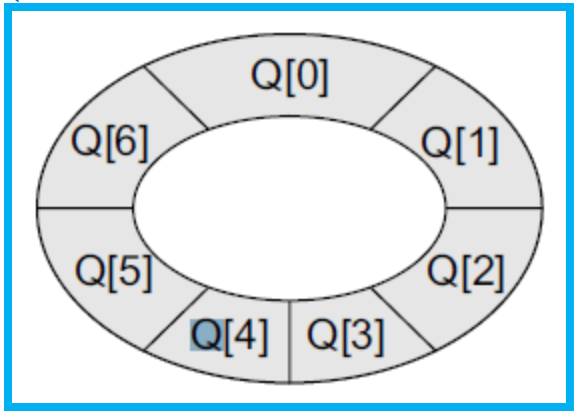


| | | 7 | 18 | 14 | 36 | 45 | 21 | 99 | 72 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- Now FRONT=2, REAR=9; even though there are spaces available, the overflow condition exists as the condition REAR=MAX-1 still holds good. Thus we are unable to insert element even though there are spaces available.

# Circular Queue

- In the Circular queue, the first index comes right after the last index.
- So, a circular queue will be full only when (FRONT = 0 and REAR= MAX - 1) OR (FRONT=REAR+1)



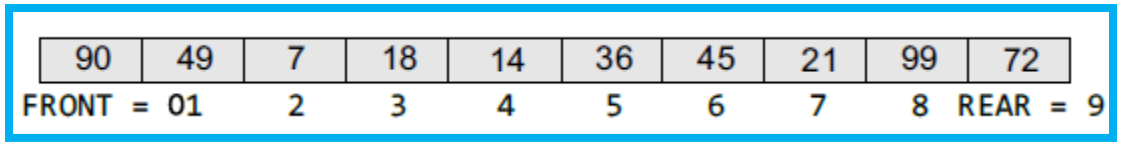
# Circular Queue-Insertion

**Algorithm/Steps for Insert operation in Circular Queue**

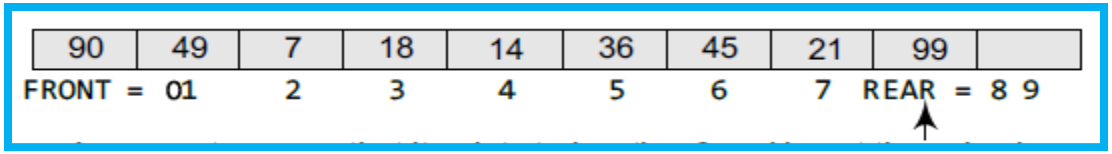
```
Step 1: IF (FRONT=0 and REAR = MAX-1) OR (FRONT=REAR+1)
        PRINT OVERFLOW
        GO TO STEP 4
   [END OF IF]
Step 2: IF FRONT=-1 and REAR=-1
                SET FRONT = REAR = 0
        ELSE IF REAR=MAX-1 and FRONT!=0
                SET REAR=0
          ELSE
                SET REAR=REAR+1
         [END OF IF]
Step 3: SET QUEUE[REAR] = VALUE
Step 4: END
```

# Circular Queue-Insertion Illustration

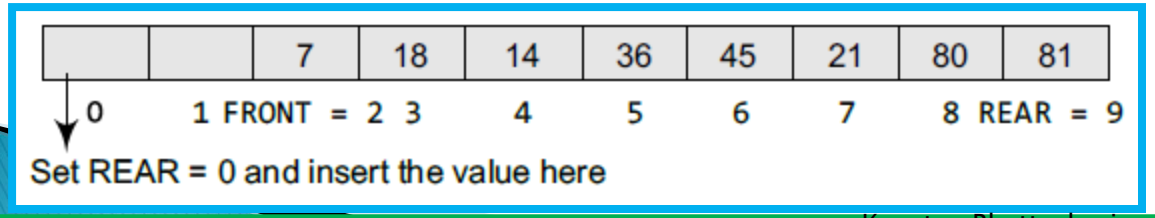
- Scenario 1: FRONT=0 and REAR=MAX-1, the circular queue is full.




- Scenario 2: If REAR!=MAX-1 then REAR will be incremented and value will be inserted.




- Scenario 3: If FRONT!=0 and REAR=MAX-1, then it means the queue is not Full. So set REAR=0 and insert new element there.



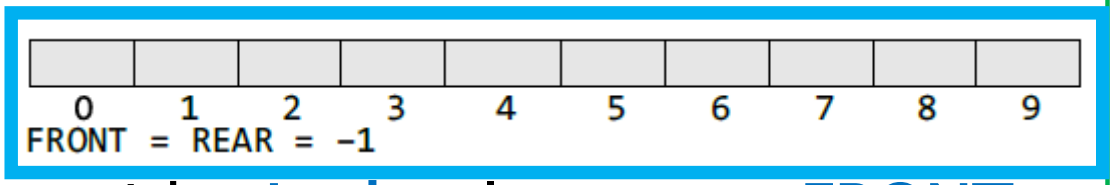
# Circular Queue-Deletion

**Algorithm/Steps for Delete operation in Circular Queue**

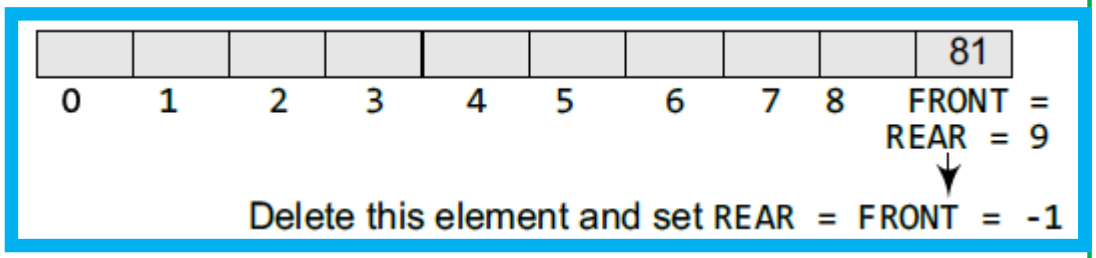
```
Step 1: IF FRONT = -1
                PRINT UNDERFLOW
          GO TO Step 4
        [END OF IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT=REAR
                SET FRONT=REAR=-1
          ELSE
                IF FRONT=MAX -1
                        SET FRONT=0
                ELSE
                        SET FRONT = FRONT + 1
                 [END OF IF]
       [END OF IF]
Step 4: END
```

# Circular Queue–Deletion Illustration

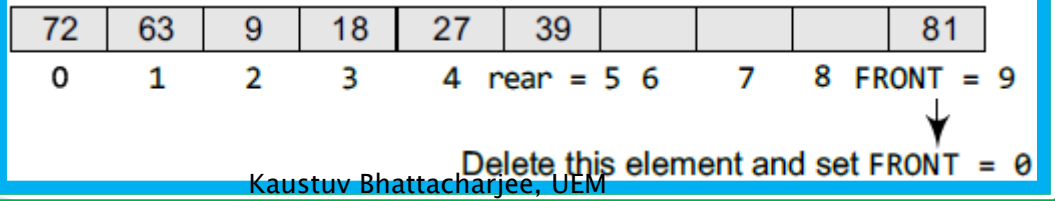
- Scenario 1: Empty queue, FRONT=-1, no elements to delete.




- Scenario 2: Queue with single element, FRONT = REAR: after deleting the element, queue is empty. Set FRONT=REAR=-1.




- Scenario 3: Queue is not empty and FRONT = MAX-1 before deletion: delete the element at the FRONT and set FRONT=0.



# Circular Queue: Assignments

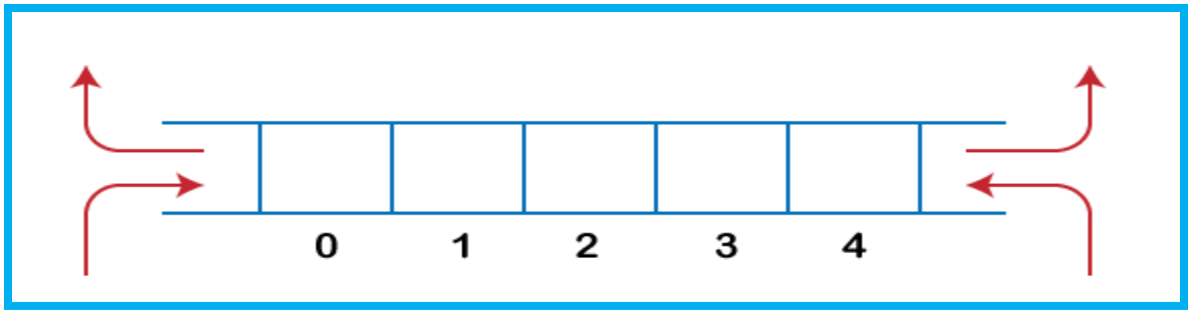
**Assignments:**

1. Write a program to **insert** an element into the circular queue.
2. Write a program to **delete** an element from a circular queue.
3. Write a program to return the value of the **FRONT** element of the circular queue(without deleting it from the queue).
4. Write a program to **display** the elements of a circular queue.

# Deque

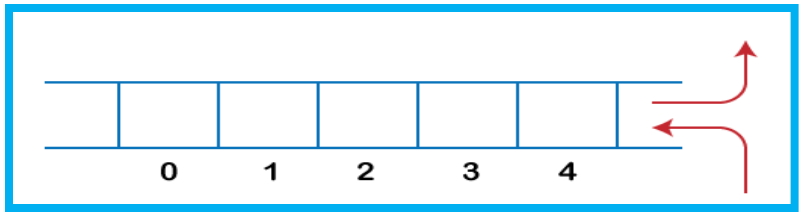
- A list in which the elements can be inserted or deleted at either end [also known as a head-tail linked list *because elements can be added to* or removed from either the front (head) or the back (tail) end]
- No element can be added and deleted from the middle.
- Implemented using either a circular array or a circular doubly linked list.
- Two pointers are maintained, LEFT and RIGHT, which point to either end of the Deque.
- The elements in a Deque extend from the LEFT end to the RIGHT end and since it is circular, Deque[N-1] is followed by Deque[0].

# Deque-Contd…

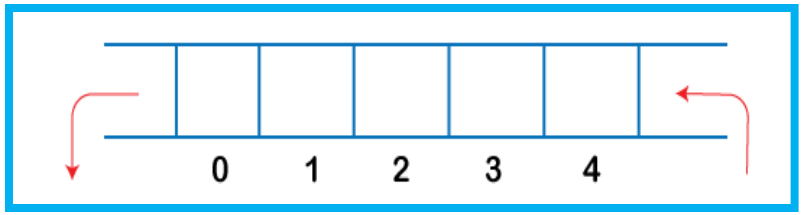



**A generalized version of a queue:**

- Can be used both as stack and queue as it allows the insertion and deletion operations on both ends




Deque as a Stack

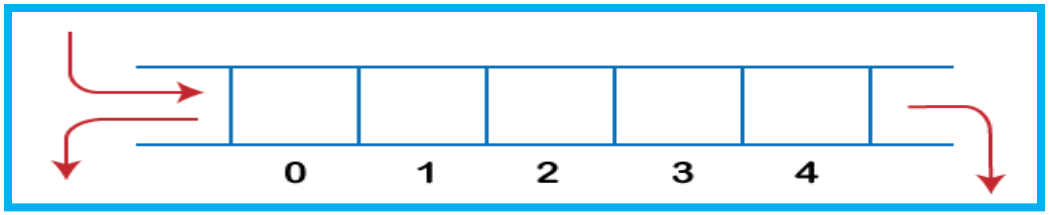



Deque as a queue

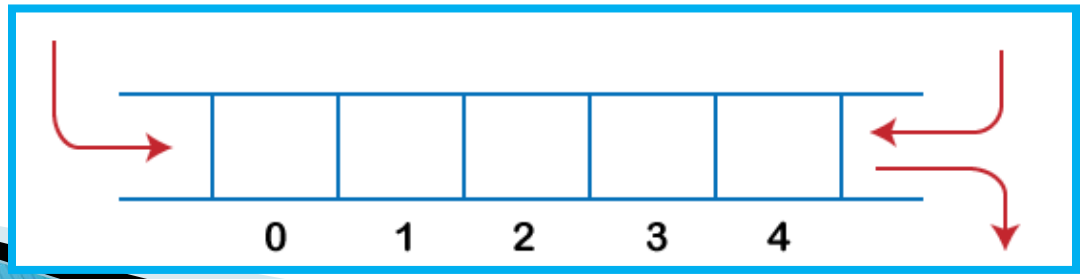
# Deque–Contd…

## Two variants of a double-ended queue:

- **Input restricted Deque:** Insertions can be done only at **one of the ends**, while deletions can be done from **both ends**.




- **Output restricted Deque:** Deletions can be done only at **one of the ends**, while insertions can be done on **both ends**.



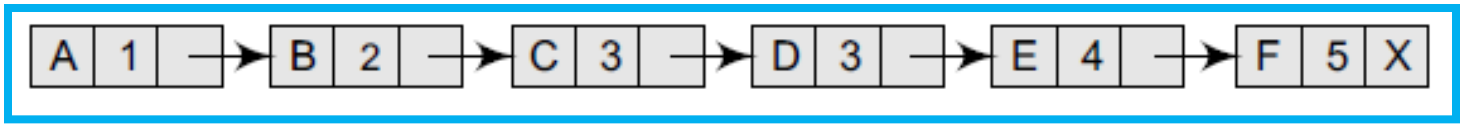
# Priority Queue

- A data structure in which each element is assigned a priority.
- The priority of the element will be used to determine the order in which the elements will be processed.
- General rules of processing the elements of a priority queue:
  - An element with higher priority is processed before an element with a lower priority.
  - Two elements with the same priority are processed on a first-come-first-served (FCFS) basis.
- Used in operating systems to execute the highest priority process first.

# Priority Queue– Implementation approaches

- **Linked Representation of Priority queue:**

  Every node will have three parts: (a) the information or data part, (b) the priority number of the element, and (c) the address of the next element.

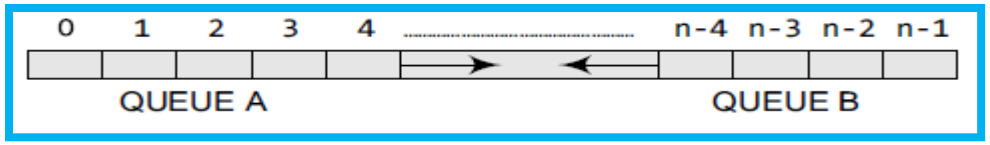


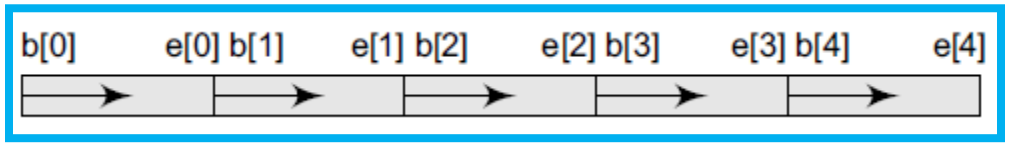

- **Array Representation of Priority queue:**

  A separate queue for each priority number is maintained. Each of these queues will be implemented using circular arrays or circular queues. Every individual queue will have its own FRONT and REAR pointers. 2D array is used here.

# Multiple Queue

- An array Queue[n] is used to represent two queues, Queue A and Queue B.
- The value of n is such that the combined size of both the queues will never exceed n.
- While operating on these queues,—queue A will grow from left to right, whereas queue B will grow from right to left at the same time.
- Extending the concept, a queue can also be used to represent n number of queues in the same array.



# Application of Queue

- Used as waiting lists for a single shared resource like printer, disk, CPU.
- Used to transfer data asynchronously (data not necessarily received at same rate as sent) between two processes (IO buffers), e.g., pipes, file IO, sockets.
- Used as buffers on MP3 players and portable CD players, iPod playlist.
- Used in Playlist for jukebox to add songs to the end, play from the front of the list.
- Used in operating system for handling interrupts. When programming a real-time system that can be interrupted, for example, by a mouse click, it is necessary to process the interrupts immediately, before proceeding with the current job. If the interrupts have to be handled in the order of arrival, then a FIFO queue is the appropriate data structure.