Queue…

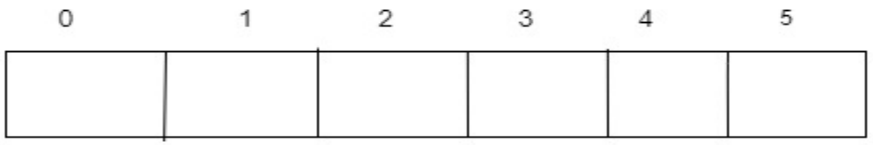
- **Implementation of queue:**
  - Array Implementation of queue
  - Linked List Implementation of queue
- **Array Implementation of queue:**
  - Queues can be represented using linear arrays.
  - **Queue Operations**:
    - **Insertion:**
      - This operation is used to insert an element into the queue at rear end. Before inserting an element we have to check whether the queue is full or not . if queue is already full it has to display **queue overflow** otherwise it permits insertions.
      - While inserting the first element both rear and front are set to zero, from second insertion onwards only rear will be incremented by 1.

```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto 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] = NUM
Step 4: EXIT
```

Array Implementation of Queue …

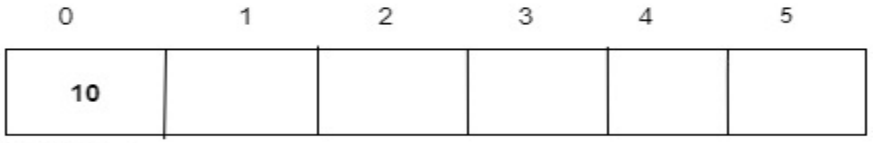
Insertion Example:

MAXSIZE=6



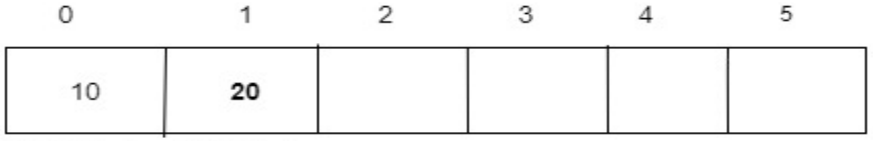
FRONT= -1

REAR= -1

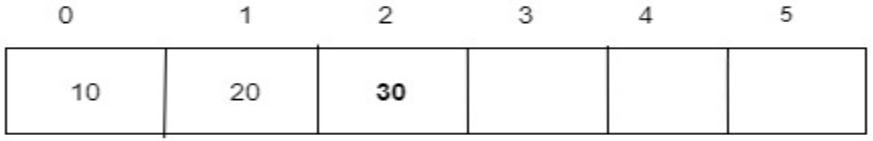


FRONT= 0

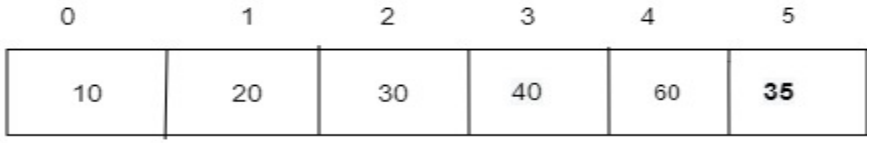
REAR= 0



FRONT= 0 REAR= 1



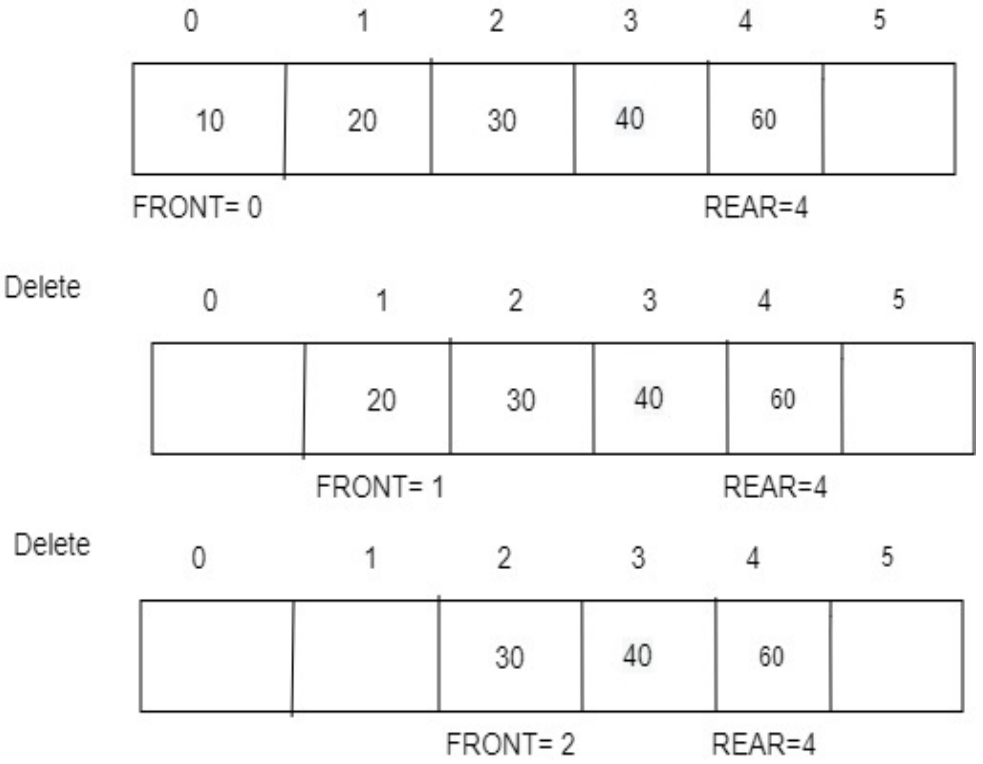
FRONT= 0 REAR= 2



FRONT= 0 REAR= 3 REAR=4 **REAR=5**

Array Implementation of Queue …

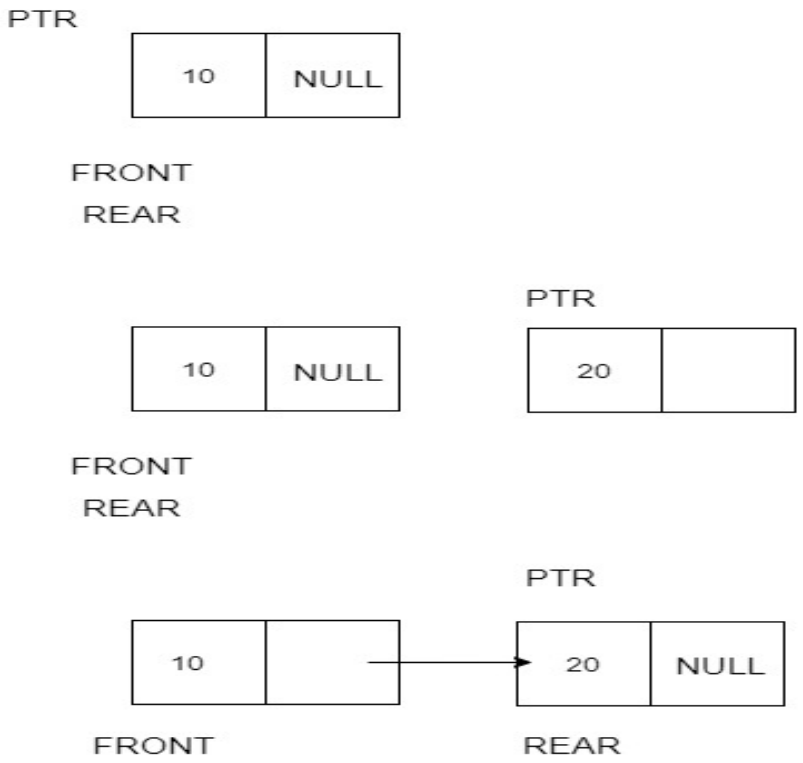
- **Deletion:**
  - This operation is used to delete an element from front end of the queue.
  - While deleting an element ,we have to check for underflow condition. An underflow occurs if front = –1 or front > rear.i,e we cannot delete an element from empty queue.
  - When an element is deleted, Front is incremented by 1.
  - Rear value does not change while deleting an element from queue.
  - Example:




```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

Linked List Implementation of Queue

- **Insertion:**
  - The new element is added as the last element of the queue.
  - Initially FRONT=NULL and REAR = NULL.
  - if FRONT=NULL, then the queue is empty.
  - There is no overflow condition in linked list implementation of queue.
  - **Example:**

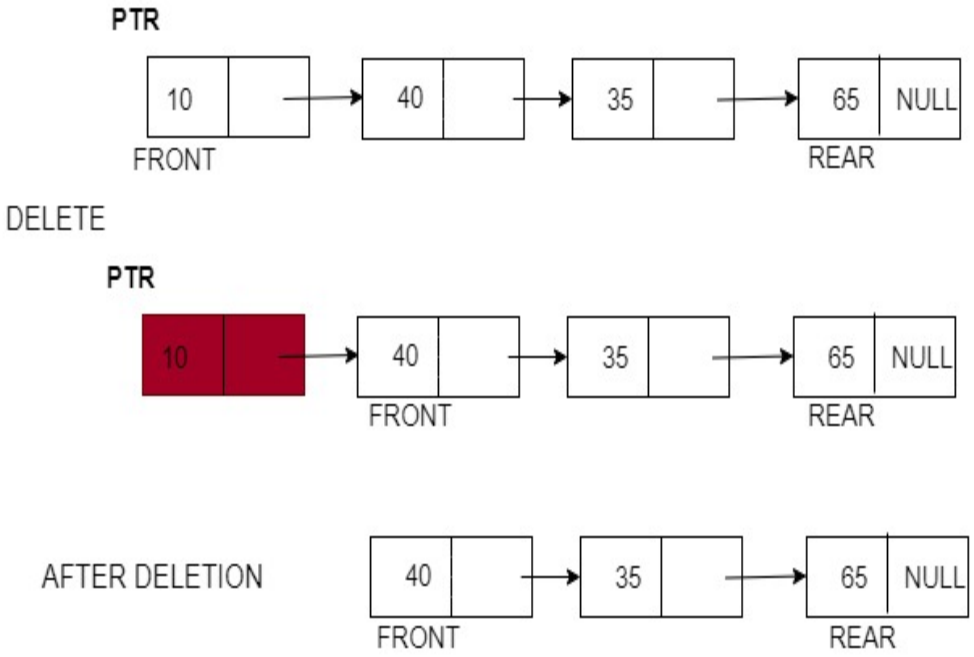



```
Step 1: Allocate memory for the new node and name
        it as PTR
Step 2: SET PTR -> DATA= VAL
Step 3: IF FRONT = NULL
            SET FRONT = REAR = PTR
            SET FRONT -> NEXT = REAR -> NEXT = NULL
        ELSE
            SET REAR ->NEXT = PTR
            SET REAR = PTR
            SET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

Linked List Implementation of Queue

➢ Deletion:

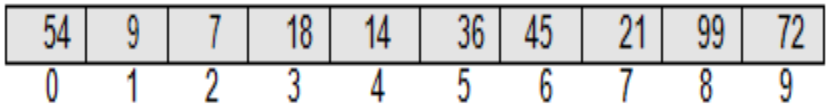
- ➢ The delete operation is used to delete the element that is first inserted in a queue, i.e., the element whose address is stored in FRONT.
- ➢ Before deleting the value, we must first check if FRONT=NULL because if this is the case, then the queue is empty and no more deletions can be done.
- ➢ If an attempt is made to delete a value from a queue that is already empty, an underflow.
- ➢ **Example:**



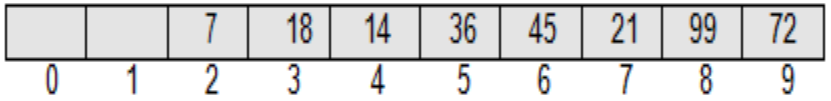

```
Step 1: IF FRONT = NULL
            Write "Underflow"
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = FRONT
Step 3: SET FRONT = FRONT -> NEXT
Step 4: FREE PTR
Step 5: END
```

Types of Queue

- **Circular Queue:**
  - In linear queues, insertions can be done only at one end called the REAR and deletions are always done from the other end called the FRONT.



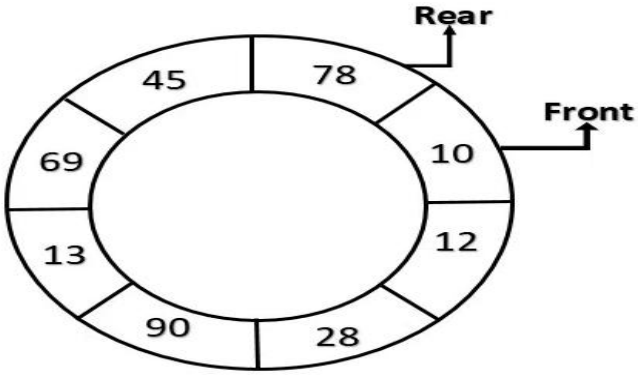
  - Suppose if we delete first two elements i,e 54 and 9 from the above queue, we get

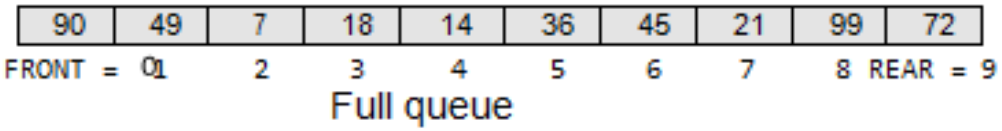


  - After deletion, the space cannot be reused for inserting the new elements even though there is space available, the overflow condition still exists because the condition REAR = MAX – 1 still holds true. This is a major drawback of a linear queue.
  - To resolve this problem, we have two solutions.
    - First, shift the elements to the left so that the vacant space can be occupied and utilized efficiently. But this can be very time-consuming, especially when the queue is quite large.
    - Second, to use a circular queue.

Circular Queue…

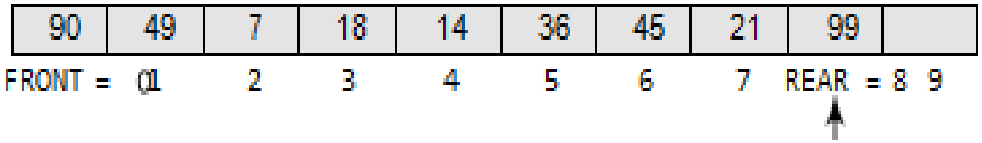
- In the circular queue, the first index comes right after the last index.
- The circular queue will be full only when
  FRONT = 0 and REAR = Max – 1.
- Operations performed on circular queue are
  - Insertion
  - Deletion
- Insertion:
  - For insertion, we now have to check for the following three conditions:
    1. If front=0 and Rear=MAX-1, then circular queue is full.





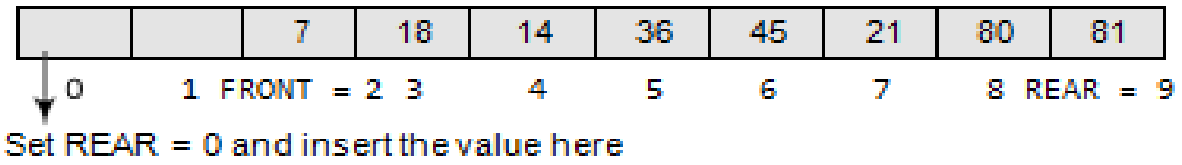
Full queue

    2. If REAR != MAX – 1 , then REAR will be incremented and the value will be inserted.



Circular Queue…

3. If FRONT != 0 and REAR = MAX – 1, then it means that the queue is not full. So, set REAR = 0 and insert the new element there.

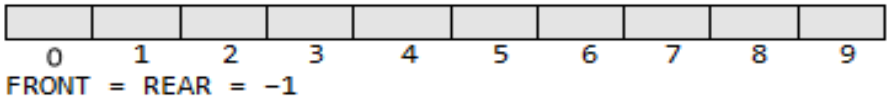



```
Step 1: IF FRONT = 0 and Rear = MAX - 1
            Write "OVERFLOW"
            Goto step 4
        [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1 and FRONT != 0
            SET REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```

Circular Queue…

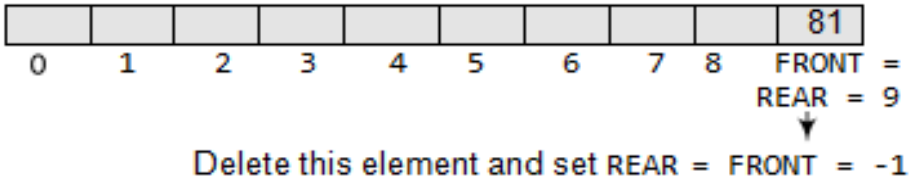
- **Deletion:**
  - To delete an element, we check for following three conditions:

1. If FRONT = –1, then there are no elements in the queue,
   This condition is called as Underflow condition.

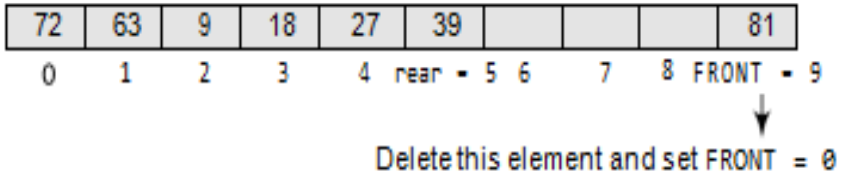



```
Step 1: IF FRONT = -1
            Write "UNDERFLOW"
            Goto Step 4
        [END of IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT = REAR
            SET FRONT = REAR = -1
        ELSE
            IF FRONT = MAX -1
                  SET FRONT = 0
            ELSE
                  SET FRONT = FRONT + 1
            [END of IF]
        [END OF IF]
Step 4: EXIT
```

2. If the queue is not empty and FRONT = REAR, then after deleting the element at the front the queue becomes empty and so FRONT and REAR are set to –1.



3. If the queue is not empty and FRONT = MAX–1, then after deleting the element at the front, FRONT is set to 0.



Dequeue

Dequeue:

- A Dequeue is a list in which the elements can be inserted or deleted at either end.
- It is 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.

- **Types of Dequeue:**
  1. **Input Restricted Dequeue :** In this dequeue insertions can be done only at one end but deletions can be done from both the ends.
  2. **Output Restricted Dequeue :** In this dequeue deletions can be done only at one end but insertions can be done on both the ends.

Queue Applications

- **Applications of Queue are:**

1. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
2. Queues are used to transfer data asynchronously (data not necessarily received at same rate as sent) between two processes.
3. Queues are used as buffers on MP3 players and portable CD players, iPod playlist.
4. Queues are used in Playlist for jukebox to add songs to the end, play from the front of the list.
5. Queues are 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.

****