Subject Name: Data Structure

Unit No:2 Unit Name: Stack and Queues

Faculty Name: Kausar Fakir

Lecture No: 6
ADT of Queues, Operation on Queue, Circular Queue, Priority Queue, Double Ended Queue

Introduction to Queue

- Queue is an abstract data structure, somewhat similar to Stack.
- In contrast to Stack, Queue is opened at **both end.**
- Queue is a list of elements in which elements can be **inserted/ stored from one end** and **elements are deleted from other end.**
- Queue follows **First-In-First-Out** methodology, i.e., the data item inserted/ stored first will be processed/ accessed first.

Queue Example

- Single-lane one-way road, where the vehicle enters first, exits first.

- Queues at ticket windows & bus-stops.

Introduction to Queue

- The end from which the elements are **inserted/ store**d is called as "**REAR**"
- The end from which the elements are **deleted** is called as " **FRONT**".

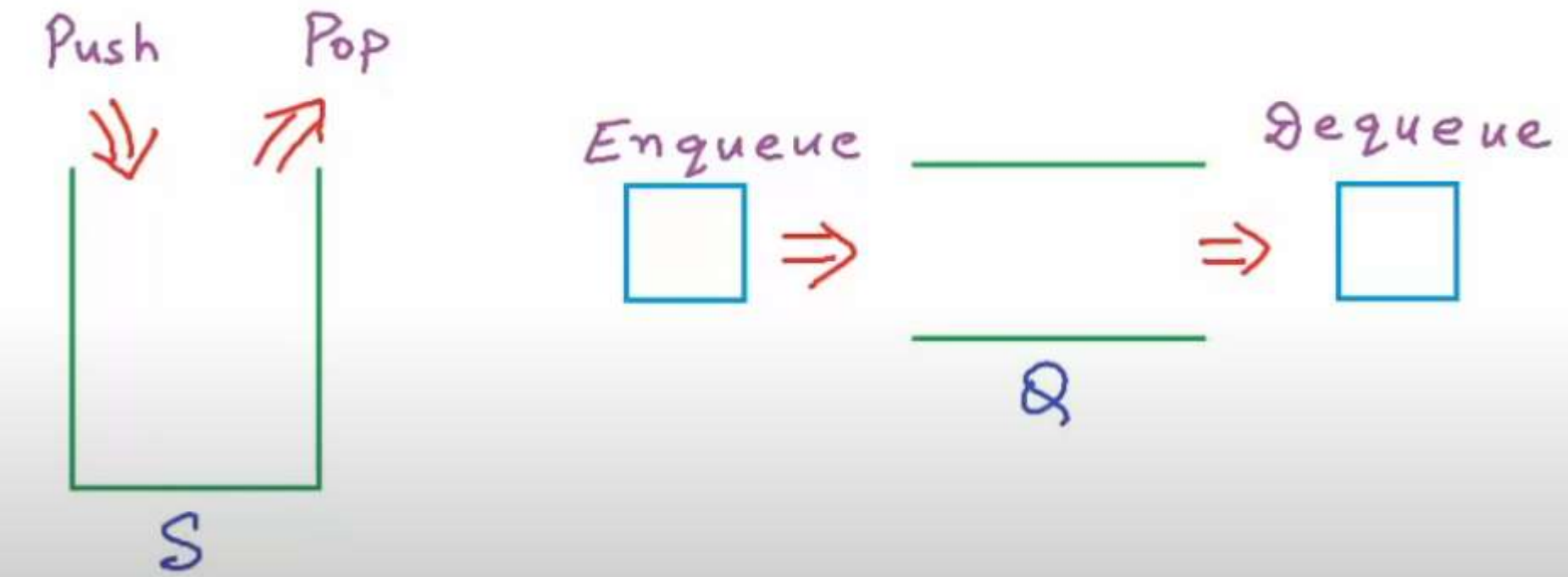
QUEUE– First In First Out (FIFO)

Introduction to Queue

- The process of inserting an element in queue is called as **Enqueue.**
- The process of deleting an element in queue is called as **Dequeue.**

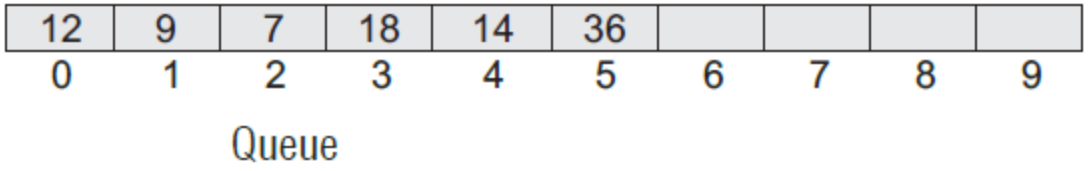
QUEUE– First In First Out (FIFO)

Difference between Stack and Queues

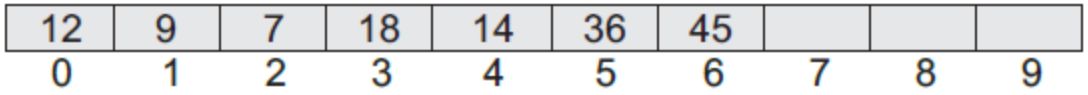


Representation of Queue

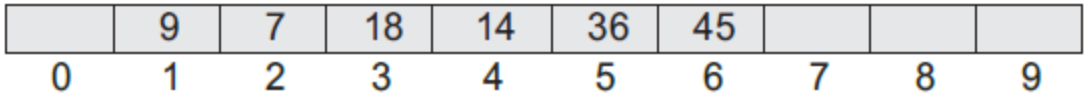
- Queues can be easily represented using linear arrays. As stated earlier, every queue has front and rear variables that point to the position from where deletions and insertions can be done, respectively.



Queue



Queue after insertion of a new element



Queue after deletion of an element

Operations of Queue

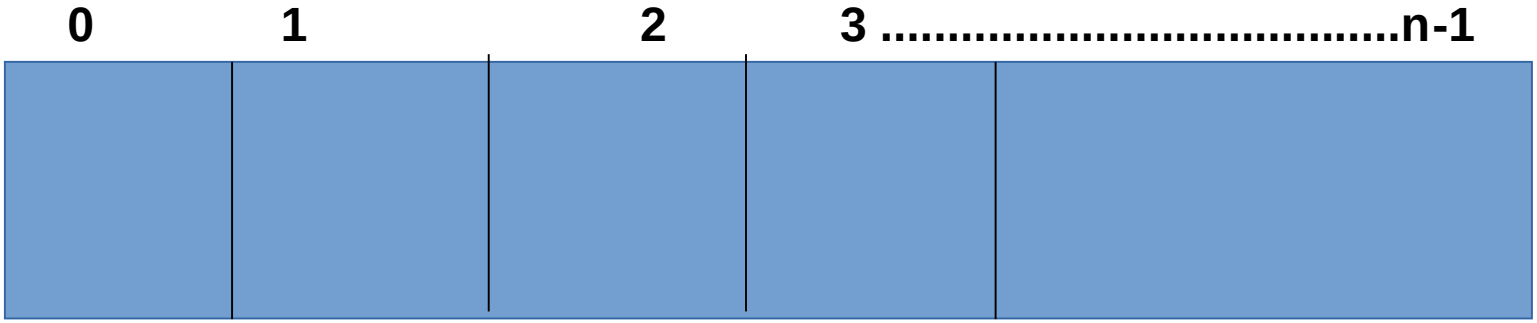
- **Create Queue**
- **Full**
- **Empty**
- **Enqueue (Insert)**
- **Dequeue (Delete)**
- **Display**

Create Queue

Create *queue* operation **creates an empty queue**.

int queue[array size];

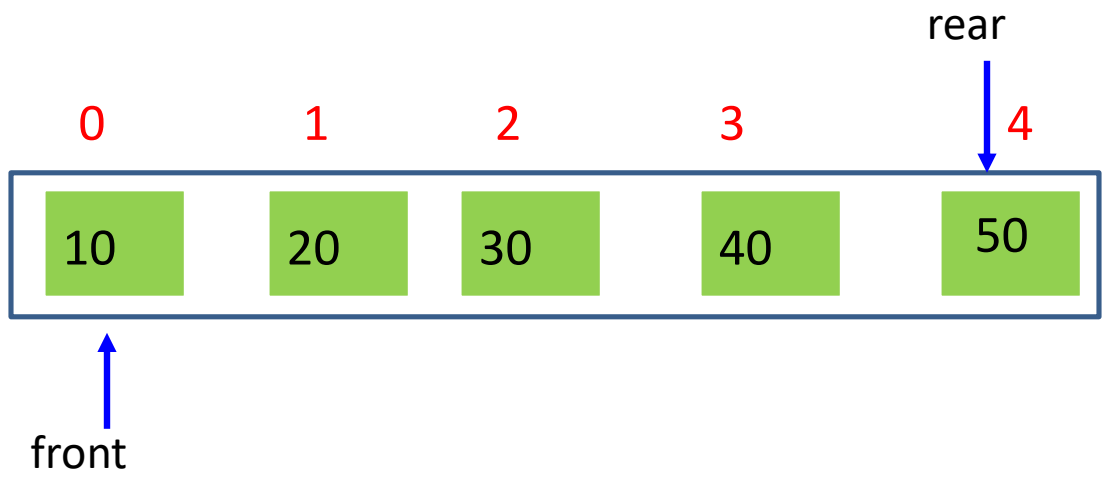
For an empty queue : front = -1; and rear=-1



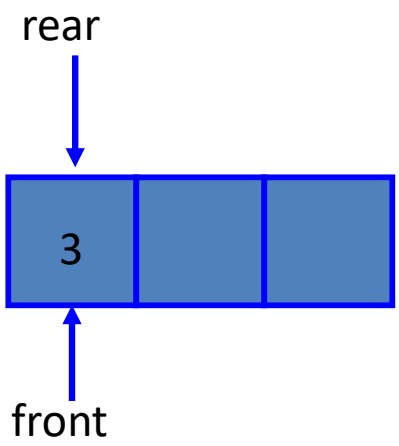
Queue Operations: Full

- The full operation checks the status of the queue.
- This operation returns true if the queue is full and false if the queue is not full.

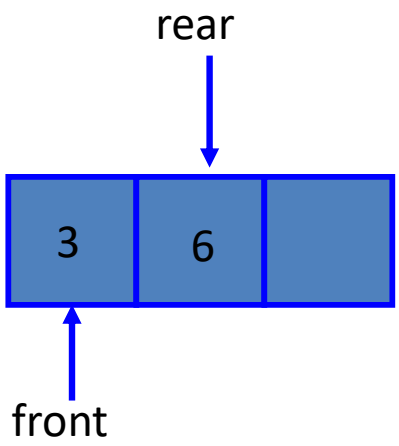
```
if(rear=max-1)
        return 1;
else
        return 0;
```



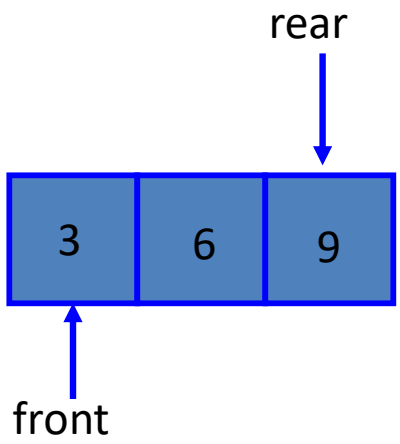
Insertion in Queue (Enqueue)



Enqueue(3)

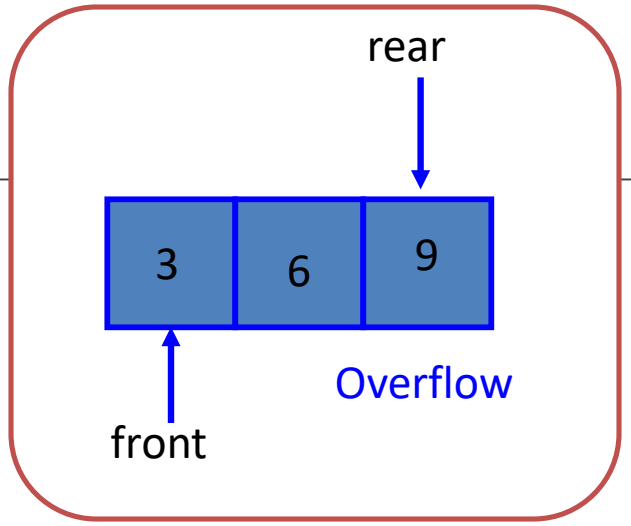


Enqueue(6)



Enqueue(9)

Insertion in Queue (Enqueue)

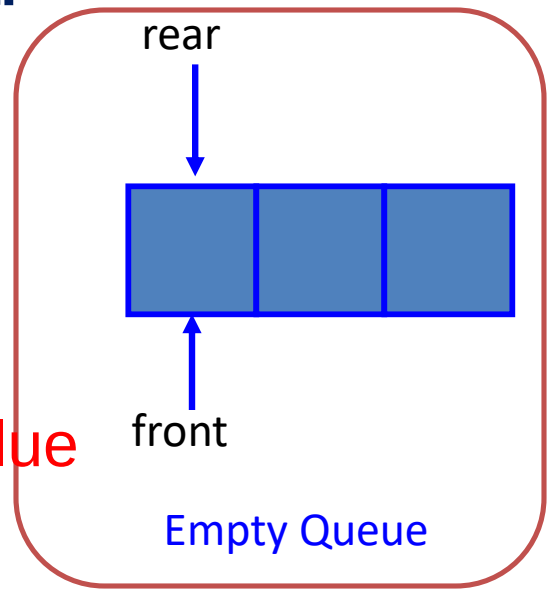


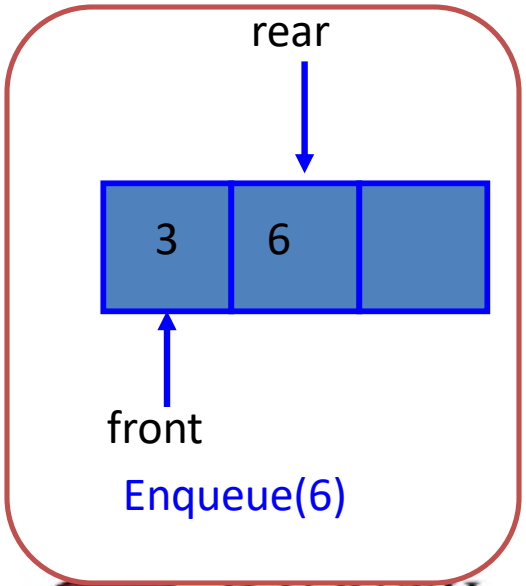
Step 1: If rear=Max-1,then
print “overflow”
[End of If]
Goto Step 4

Step 2: If front= -1 && rear = -1

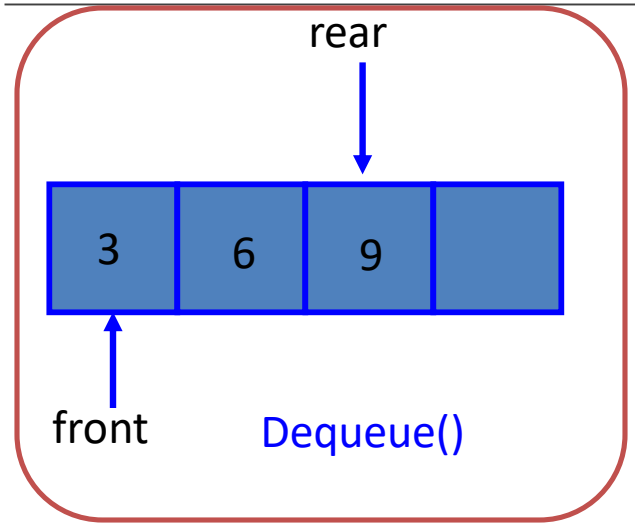
Set front = rear = 0
else
Set rear = rear+1
[End of If]

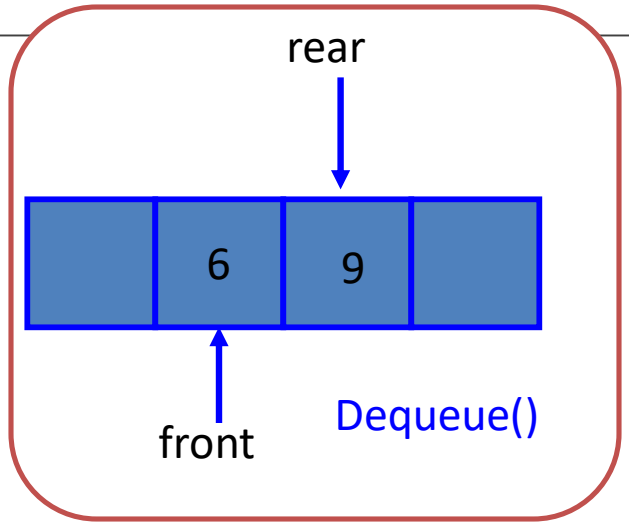
Step 3: Set Queue[rear]=value
Step 4:End

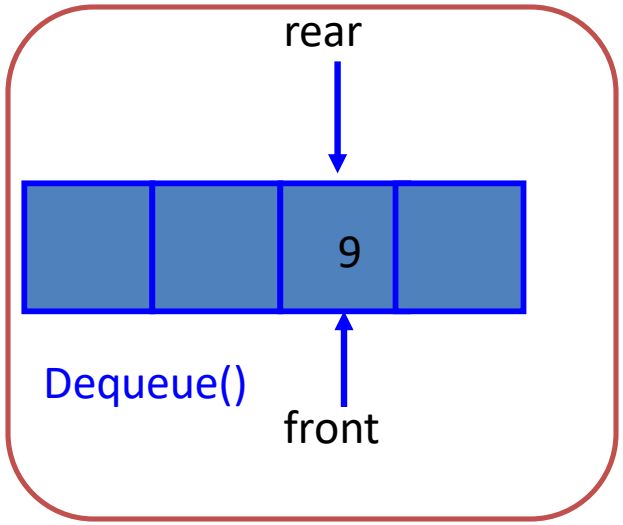


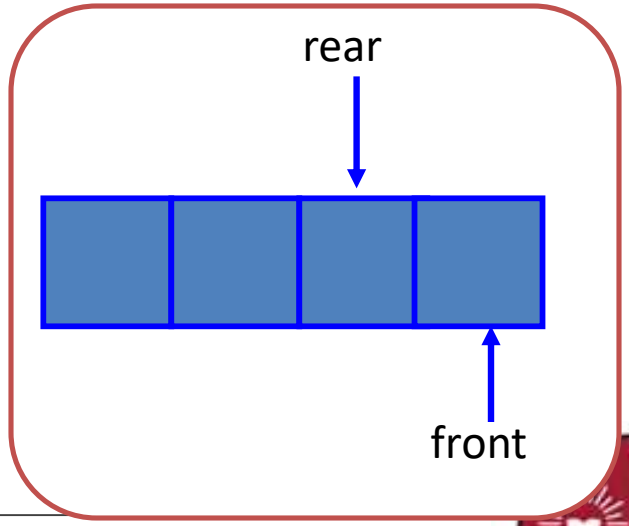


Deletion from Queue (Dequeue)









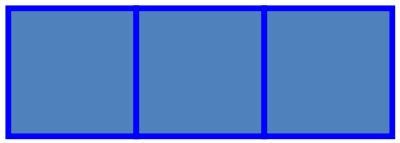
Empty :

front > rear

Queue Operations: Empty

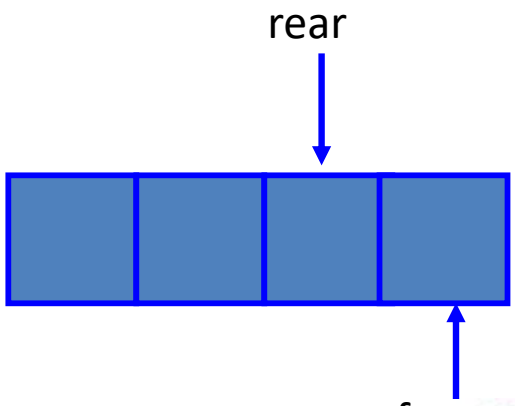
- This operation returns true if the queue is empty and false otherwise.

```
if(front = -1 II front > rear)
        return 1;
else
        return 0;
```



rear= -1

front = -1



front >rear

Deletion from Queue (Dequeue)

Step 1: If **front > rear II front = -1 ,** then

print “Underflow”

[End of If]

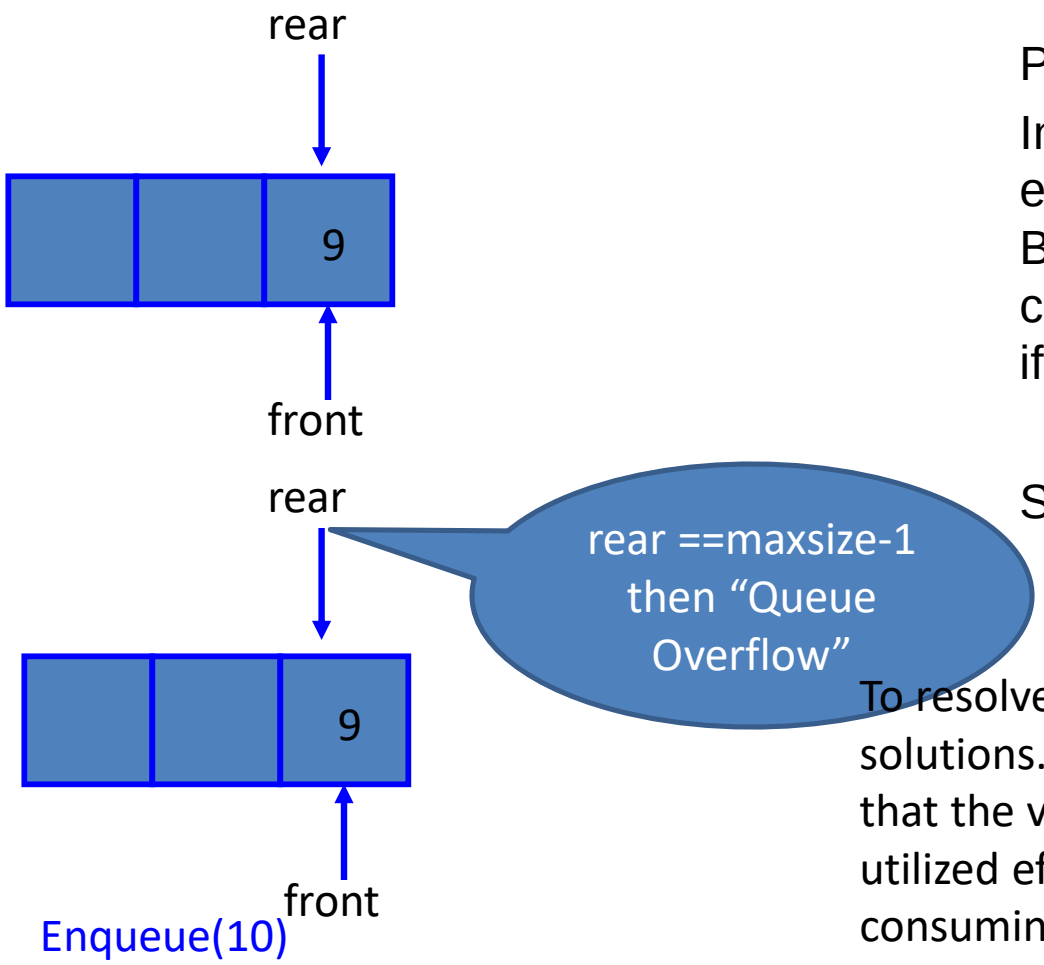
Goto Step 4

Step 2: **Set value=Queue[front]**

Step 3: Set **front = front + 1**

Step 4: End

Problem with Linear Queue



Problem :

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

Solution: ???????

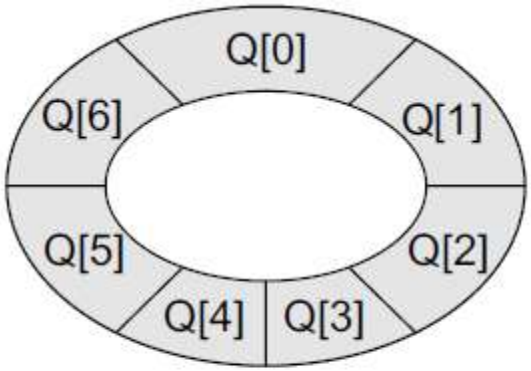
To resolve this problem, we have two solutions. First, shift the elements to the left so that the vacant space can be occupied and utilized efficiently. But this can be very time consuming, especially when the queue is quite large. The second option is to use a circular queue.

Types of Queue

A queue data structure can be classified into the following types:

1. Circular Queue
2. Deque
3. Priority Queue
4. Multiple Queue

Solution : 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.
- A circular queue is implemented in the same manner as a linear queue is implemented.
- The only difference will be in the code that performs insertion and deletion operations.

Advantage of Circular Queue

- In linear queue when we delete any element only **front increment by 1,** but that position is not used later. So when we perform more add and delete operation, memory wastage increase.
- But in Circular Queue, if we delete any element that position is used later, because it is circular organization.
- In circular queue we can insert new item to the location from where previous item to be deleted

Circular Queue Operations

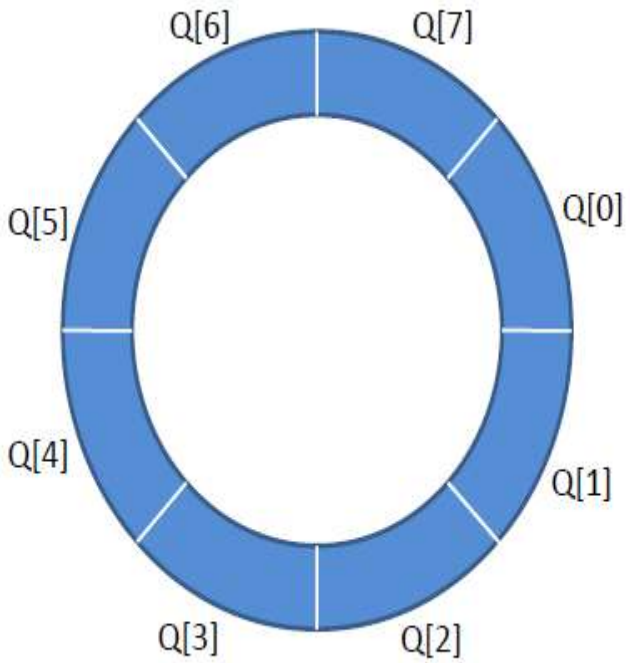
- Create
- Insertion
- Full
- Deletion
- Empty
- Display

Circular Queue Operation: Create

Create *circular queue* operation **creates an empty queue**.

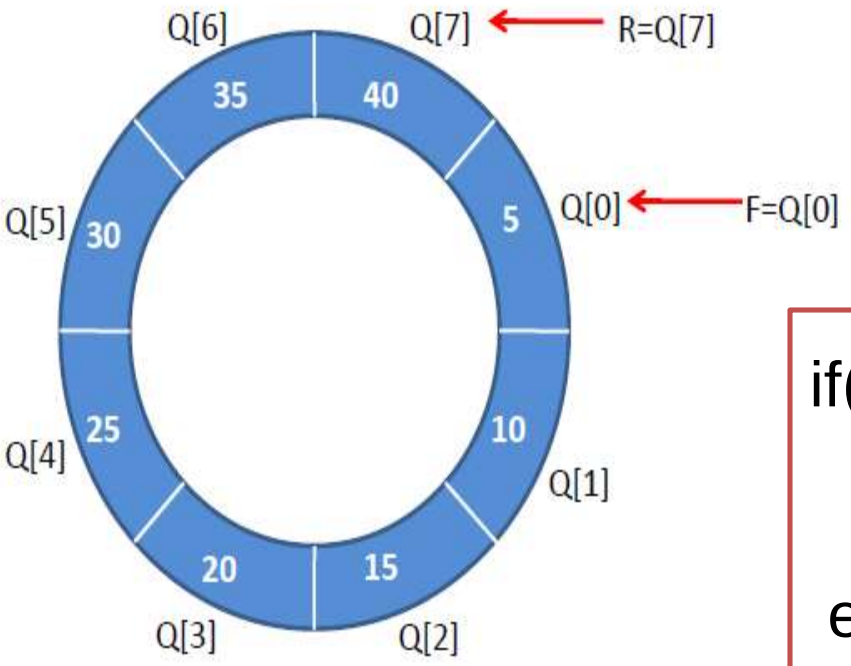
int Q[array size];

For an empty queue : front = -1; and rear=-1



Circular Queue Operation: Full : Scenario 1

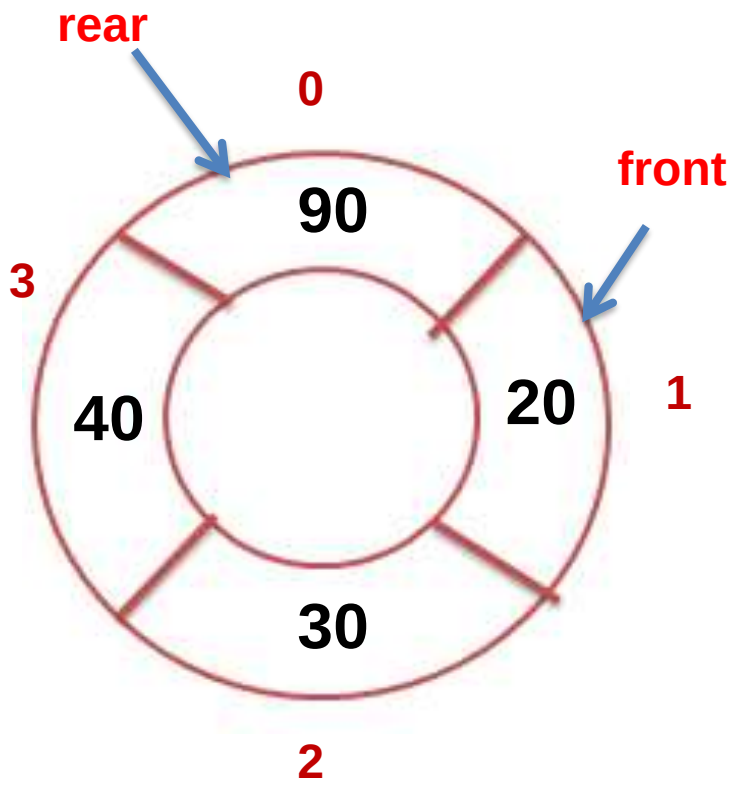
rear = max-1 && front = 0



Queue is full(Overflow)

```
if(   rear = max-1 && front = 0)
            return 1;
 else
            return 0;
```

Circular Queue Operation: Full : Scenario 2



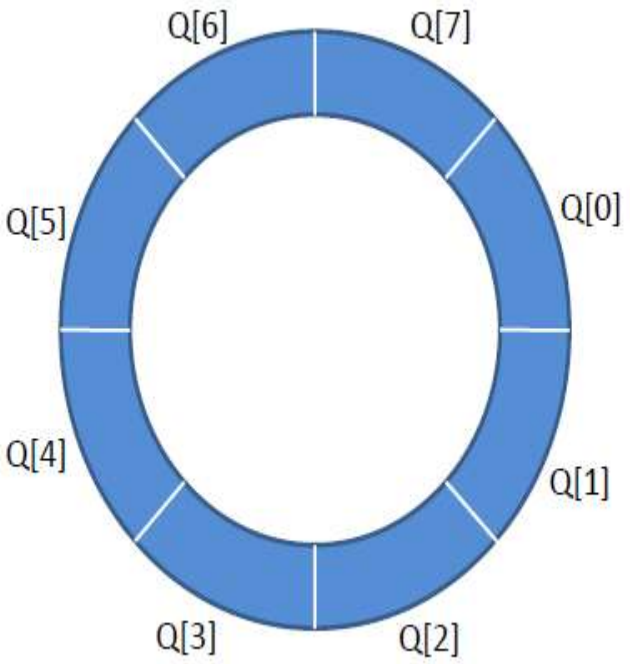
rear = front - 1

Queue is Full

```
if(   (rear = max-1 && front = 0) ||
          (rear= front -1) )
                    return 1;
 else
                    return 0;
```

Circular Queue Operation: Empty

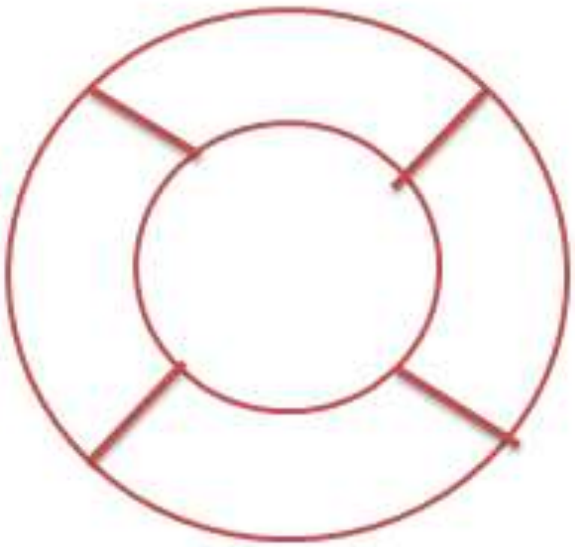
Empty queue
front = -1; and rear = - 1



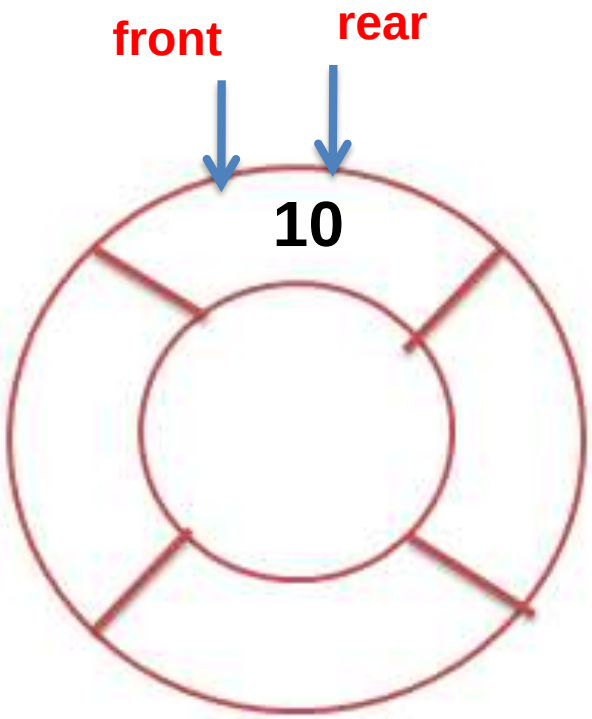

```
if(front = -1 && rear=-1)
        return 1;
 else
        return 0;
```

Insertion in Circular Queue : Scenario 1

Empty Queue



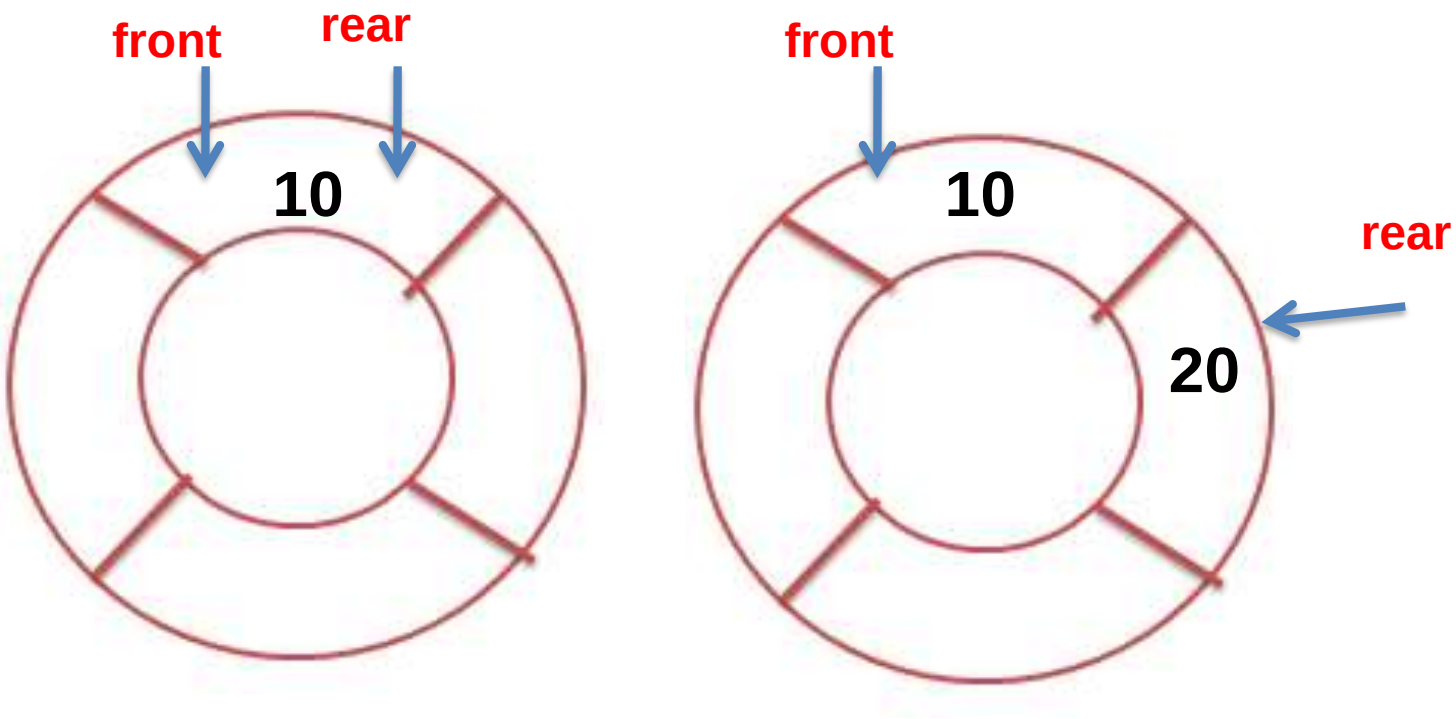
rear = -1 && front = -1



Enqueue(10)

rear = 0 && front = 0

Insertion in Circular Queue : Scenario 2

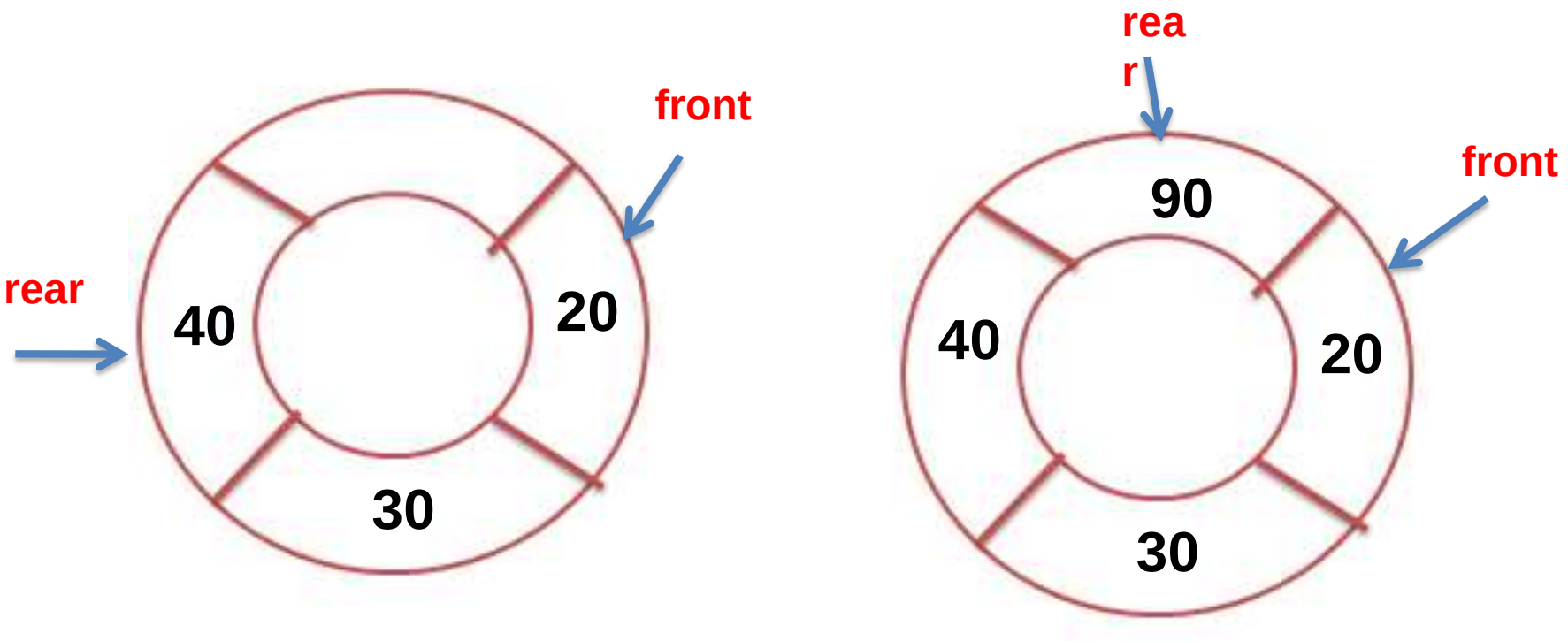


front = 0 && rear! = max-1

rear = rear + 1

Enqueue(20)

Insertion in Circular Queue : Scenario 3



IF rear= max-1 && front != 0

rear = 0

Enqueue(90)

Insertion in Circular Queue: Algorithm

Step 1: If (rear= max-1 && front=0) II (rear = front -1) then

print “overflow”

[End of If]

Goto Step 4

Step 2: If front= -1 && rear = -1

Set front = rear = 0

else if rear=max-1 && front != 0

Set rear = 0

else

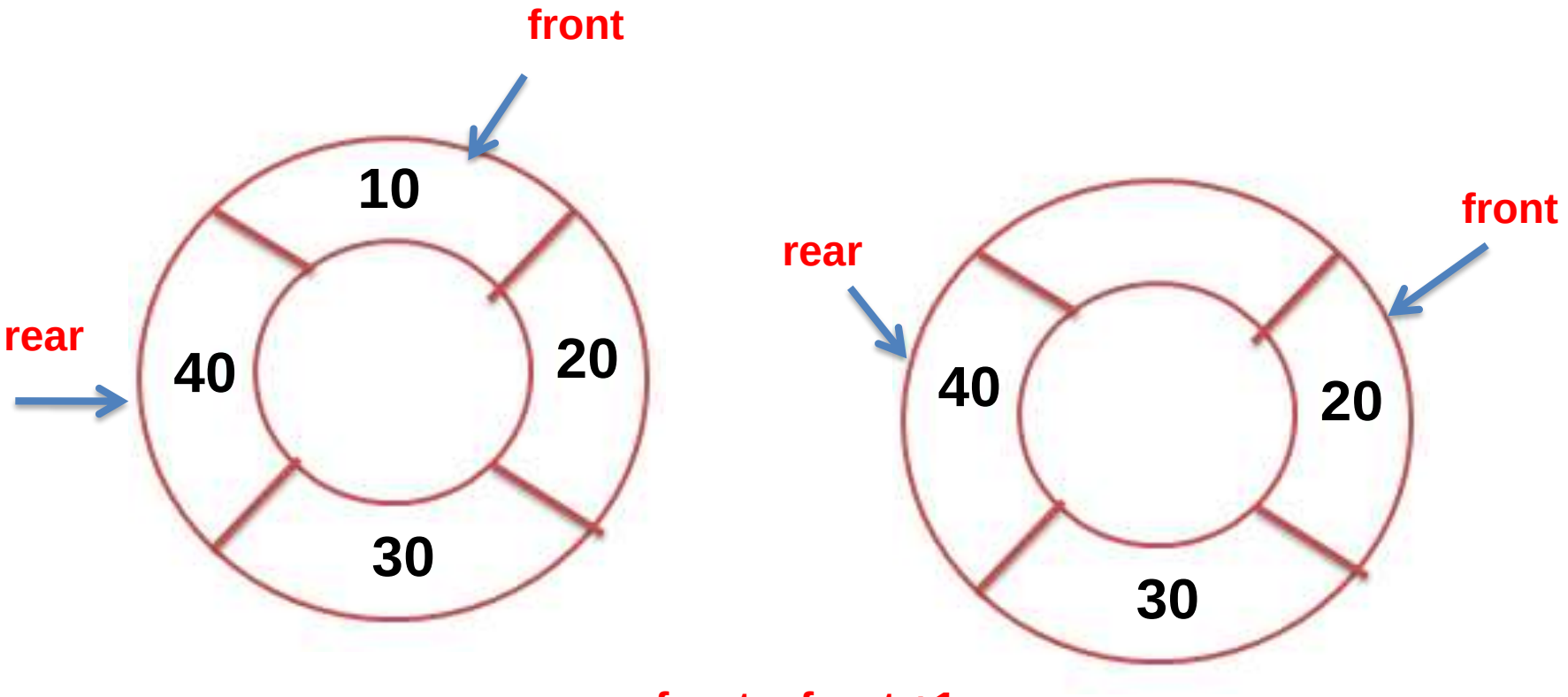
Set rear = rear+1

[End of If]

Step 3: Set **Queue[rear]=value**

Step 4:End

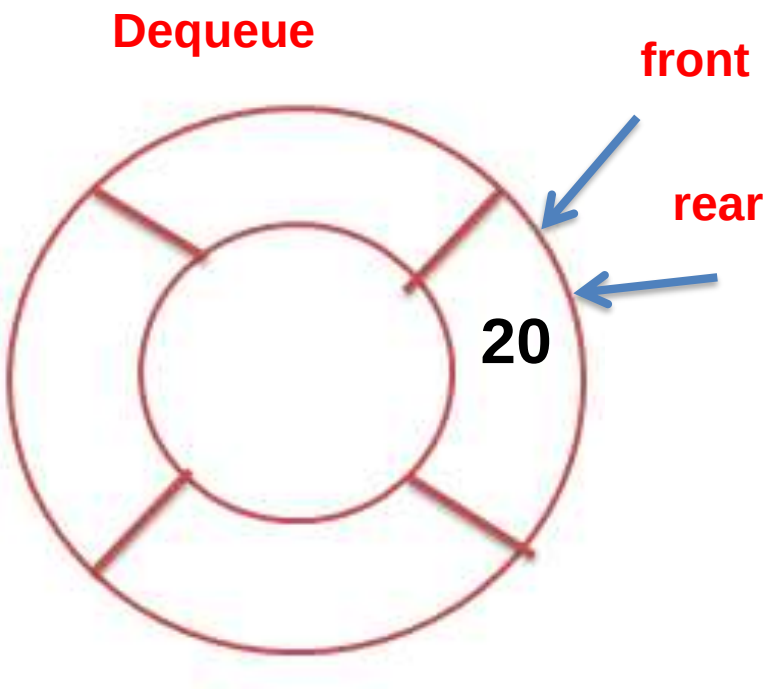
Deletion from Circular Queue : Scenario 1



front = front +1

Dequeue( )

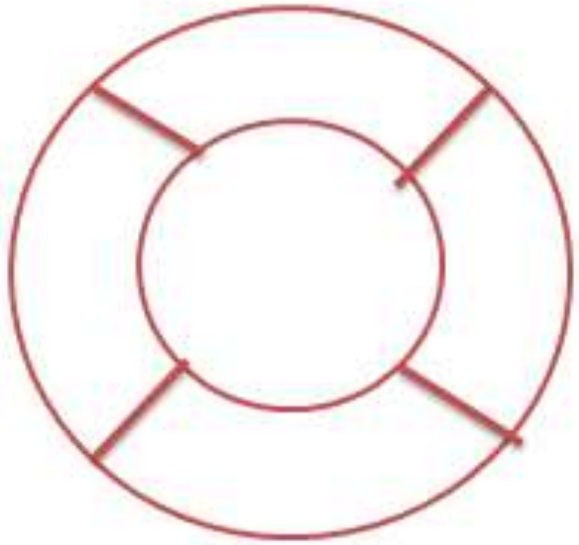
Deletion from Circular Queue : Scenario 2



front = rear

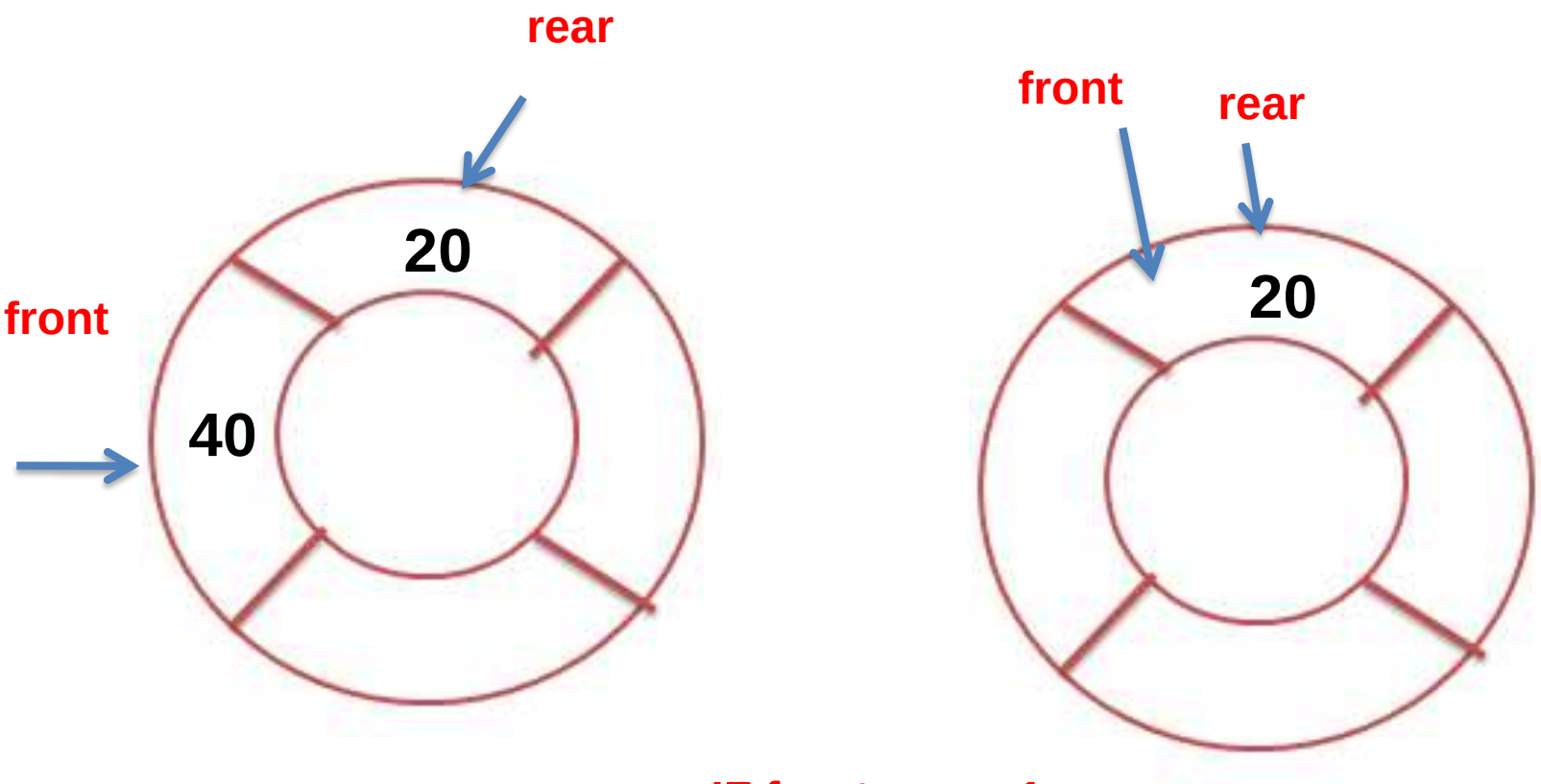
After Dequeue operation

queue becomes empty



front = rear = -1

Deletion from Circular Queue : Scenario 3



IF front= max-1

front = 0

Dequeue

Deletion in Circular Queue: Algorithm

Step 1: If front= -1 && rear=-1 then

print “Underflow”

[End of If]

Goto Step 4

Step 2: Set **value = 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]

Step 4:End

Priority Queue

- In priority queue, each element is assigned a priority.
- Priority of an element determines the order in which the elements will be processed.
- Rules:

  1. An element with higher priority will processed before an element with a lower priority.

  2. Two elements with the same priority are processed on a First Come First Serve basis.

Types of Priority Queue

1. Ascending Priority Queue

In this type of priority queue, elements can be inserted into any order but only the smallest element can be removed.

2. Descending Priority Queue

In this type of priority queue, elements can be inserted into any order but only the largest element can be removed.

Priority Queue Example

Initial Queue = { }

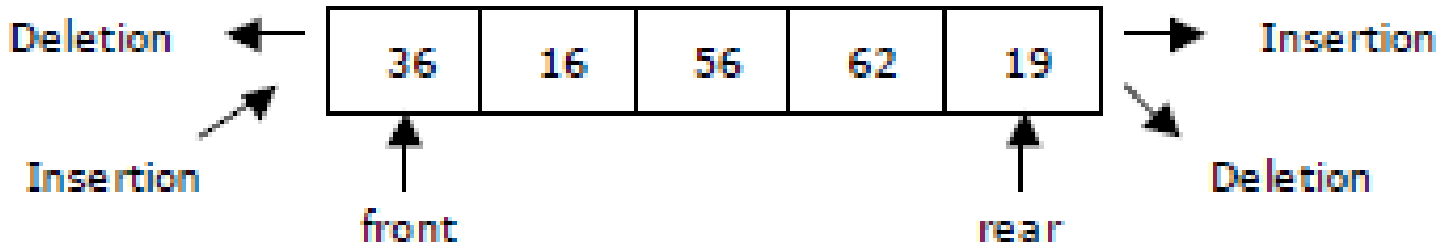
Operation	Return value	Queue Content
insert (C)		C
insert (O)		C O
insert (D)		C O D
remove max	O	C D
insert (I)		C D I
insert (N)		C D I N
remove max	N	C D I
insert (G)		C D I G

Applications of Queue

- Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
- Queues are 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.
- Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.
- Computer systems must often provide a “holding area” for messages between two processes, two programs, or even two systems. This holding area is usually called a “buffer” and is often implemented as a queue.
- In shared resources
- Keyboard buffer
- Round robin scheduling
- Job Scheduling

Double Ended Queue (Deque)

The word *deque* is an acronym derived from *double-ended queue*.



There are two variations of deque. They are:

- Input restricted deque (IRD)
- Output restricted deque (ORD)

Double Ended Queue (Deque)

There are two variations of deque. They are:

- **Input restricted deque (IRD)**

In this, insertions can be done only at one of the ends, while deletions can be done from both ends.

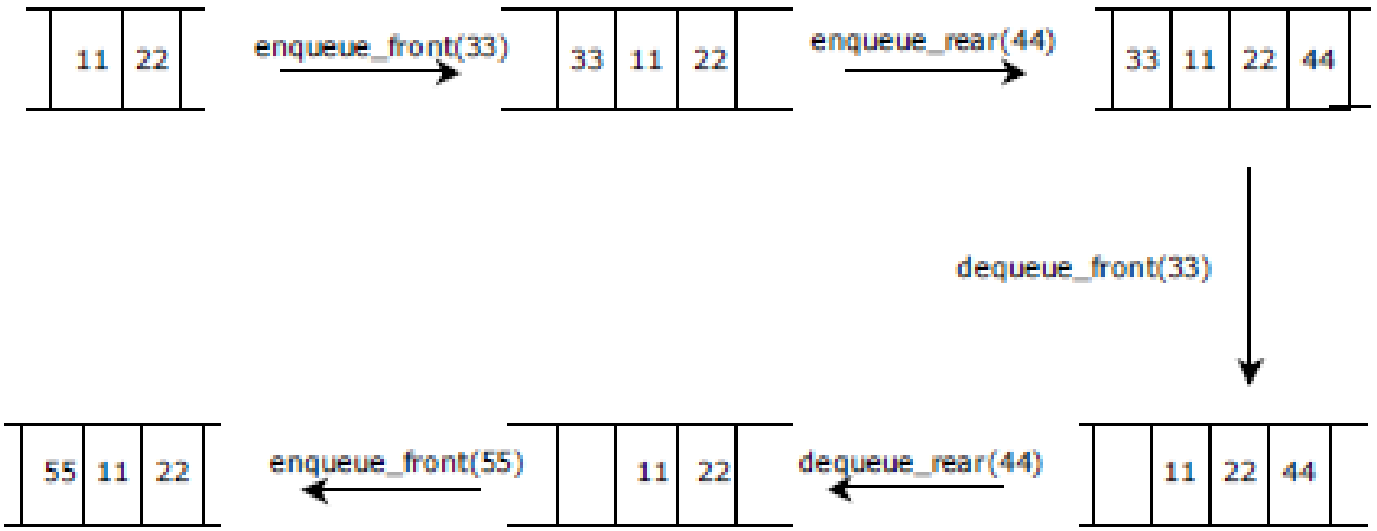
- **Output restricted deque (ORD)**

Deletions can be done only at one of the ends, while insertions can be done on both ends.

Operations on Deque

A deque provides four operations. Figure shows the basic operations on a deque.

- enqueue_front: insert an element at front.
- dequeue_front: delete an element at front.
- enqueue_rear: insert element at rear.
- dequeue_rear: delete element at rear.



Thank You