(2 * 3) + (2 / 4) – (4 + 3)

Find Prefix **- + * 2 3 / 2 4 + 4 3**
Evaluate Prefix

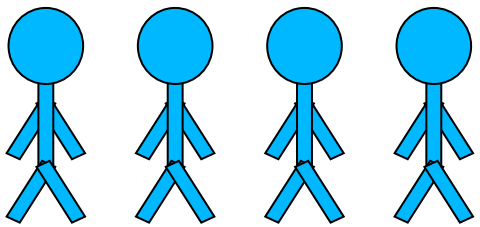
Find Postfix **2 3 * 2 4 / + 4 3 + -**
Evaluate Postfix

Queues

Introduction to Queues

- A queue is a waiting line – seen in daily life
  - A line of people waiting for a bank teller
  - A line of cars at a toll both
- What other kinds of queues can you think of

The queue has a **front** and a **rear**.



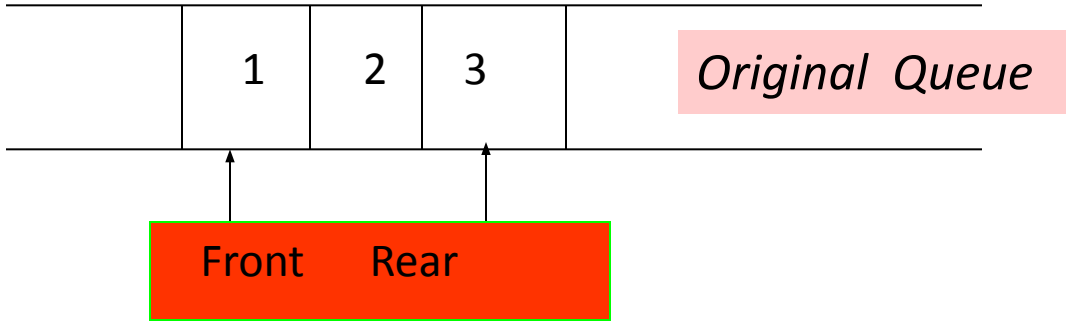
Front

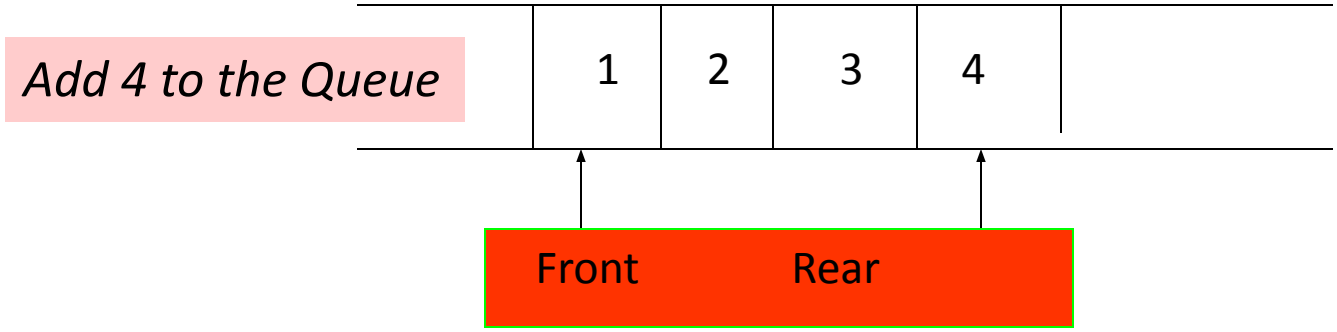
Rear

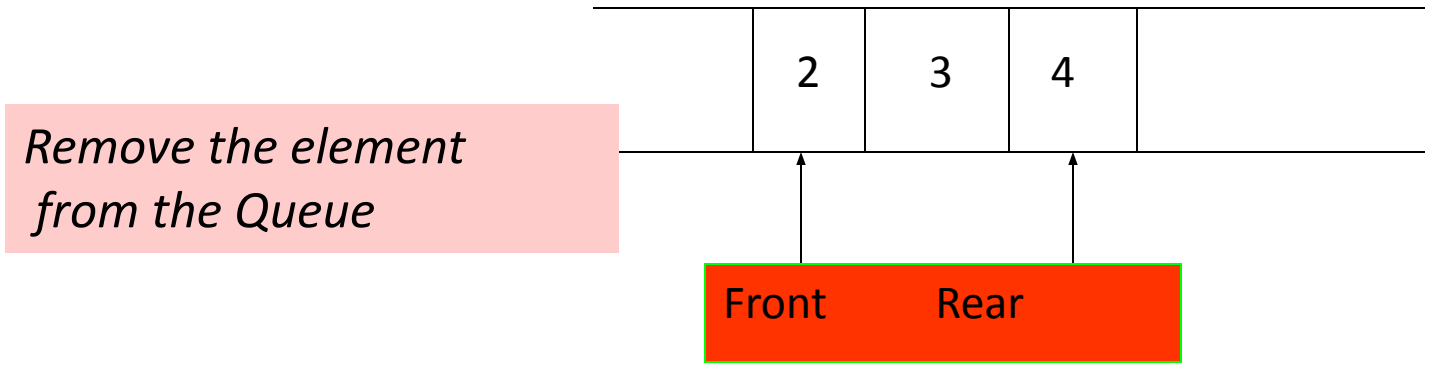
In Queue

a. Items can be removed only at the front

b. Items can be added only at the other end, the back



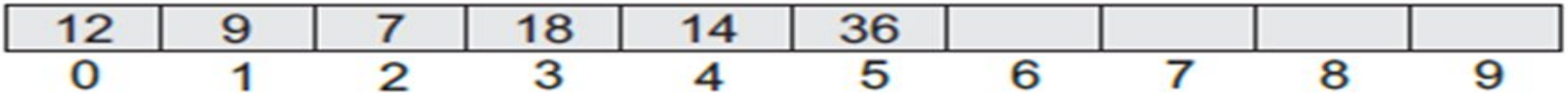




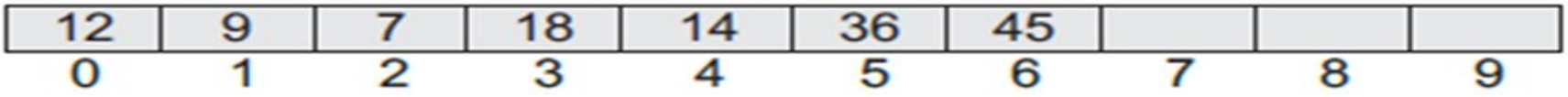
The Queue As an ADT

1. A queue is a sequence of data elements
2. Basic operations
   a. **Enqueue** (add element to back)
   b. **Dequeue** (remove element from front)

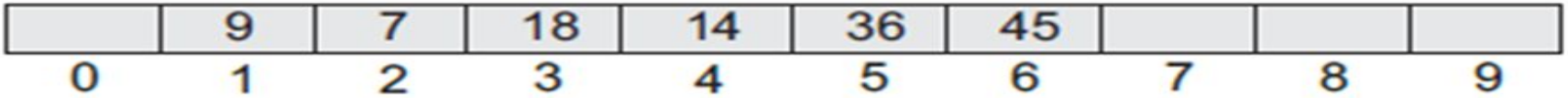
Enqueue & **Dequeue** operations



Queue



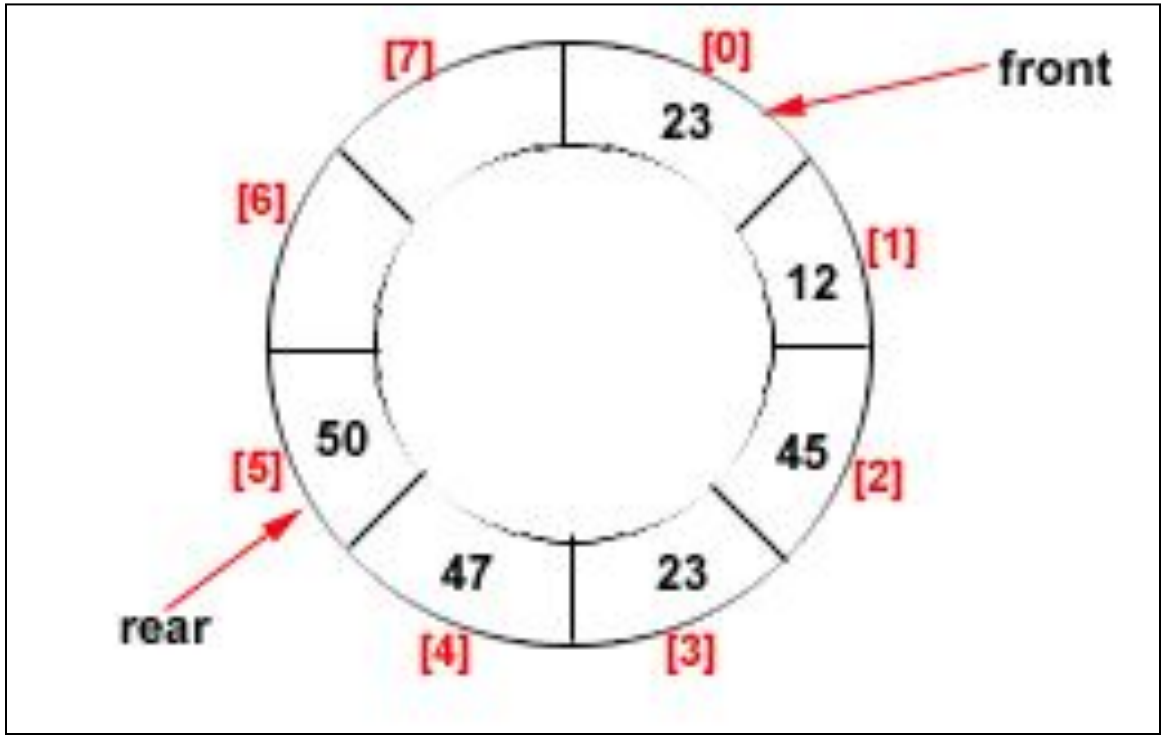
Queue after insertion of a new element



Queue after deletion of an element

Types Of Queues

- Linear Queue
- Circular Queue
- Double Ended Queue (Deque)
- Priority Queue



Double Ended Queue

Double ended queues, called **deques** for short, are a generalized form of the queue. It is exactly like a queue except that elements can be added to or removed from the **head** *or* the **tail**.

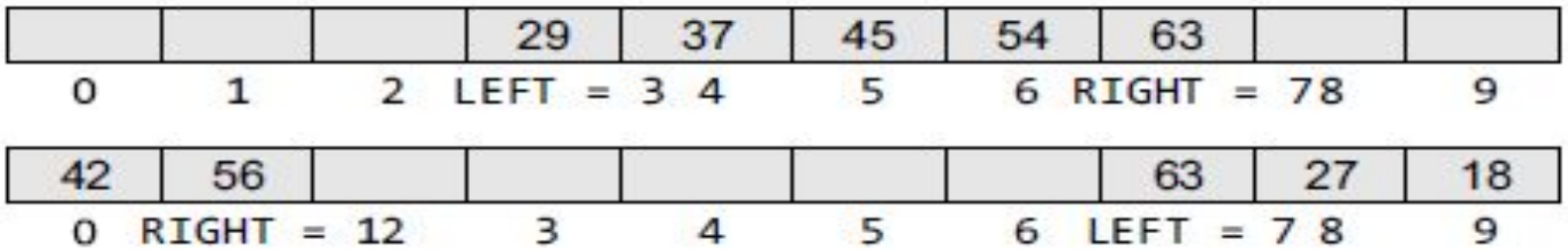
However, no element can be added and deleted from the middle.

In the computer's memory, a deque is implemented using either a **circular list** or a **circular doubly linked list**.

In a deque, 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, Dequeue[N–1] is followed by Dequeue[0].

There are two variants of a double-ended queue. They include :

- *Input restricted deque* In this, insertions can be done only at one of the ends, while deletions can be done from both ends.
- *Output restricted deque* In this deletions can be done only at one of the ends, while insertions can be done on both ends.



- deque is **useful for priority queuing**.
- A deque can model a station where cars can enter and leave on the left or right side of a line, but only the cars at the ends can move in and out.
- common application of the deque is storing a software application's list of undo operations.

Array Implementation -Dequeue

When an item is taken from the queue, it always comes from the front.

This a **dequeue** operation.

```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        SET VAL = QUEUE[FRONT]

        IF FRONT =  REAR
            SET FRONT = REAR = -1

        ELSE
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

Array Implementation -Enqueue

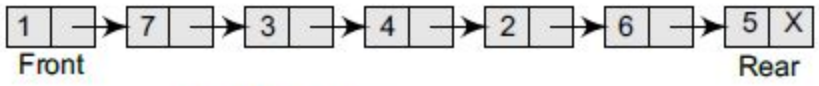
When an item is inserted into the queue, it always goes at the end (rear).

This a **enqueue** operation.

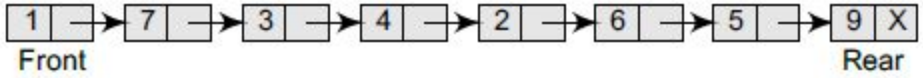
```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```

LINKED REPRESENTATION OF QUEUEs

Enqueue:



Linked queue



Linked queue after inserting a new node

```
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
```

Dequeue

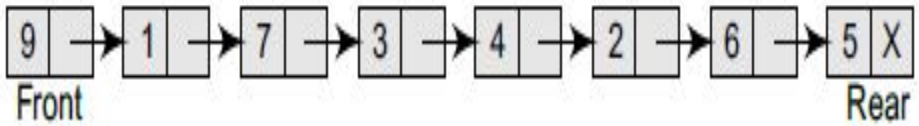


Figure 8.10 Linked queue

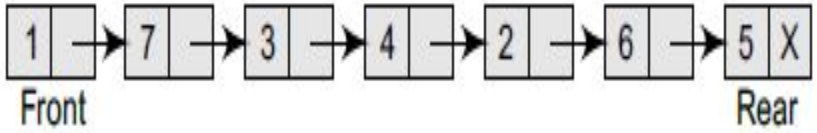
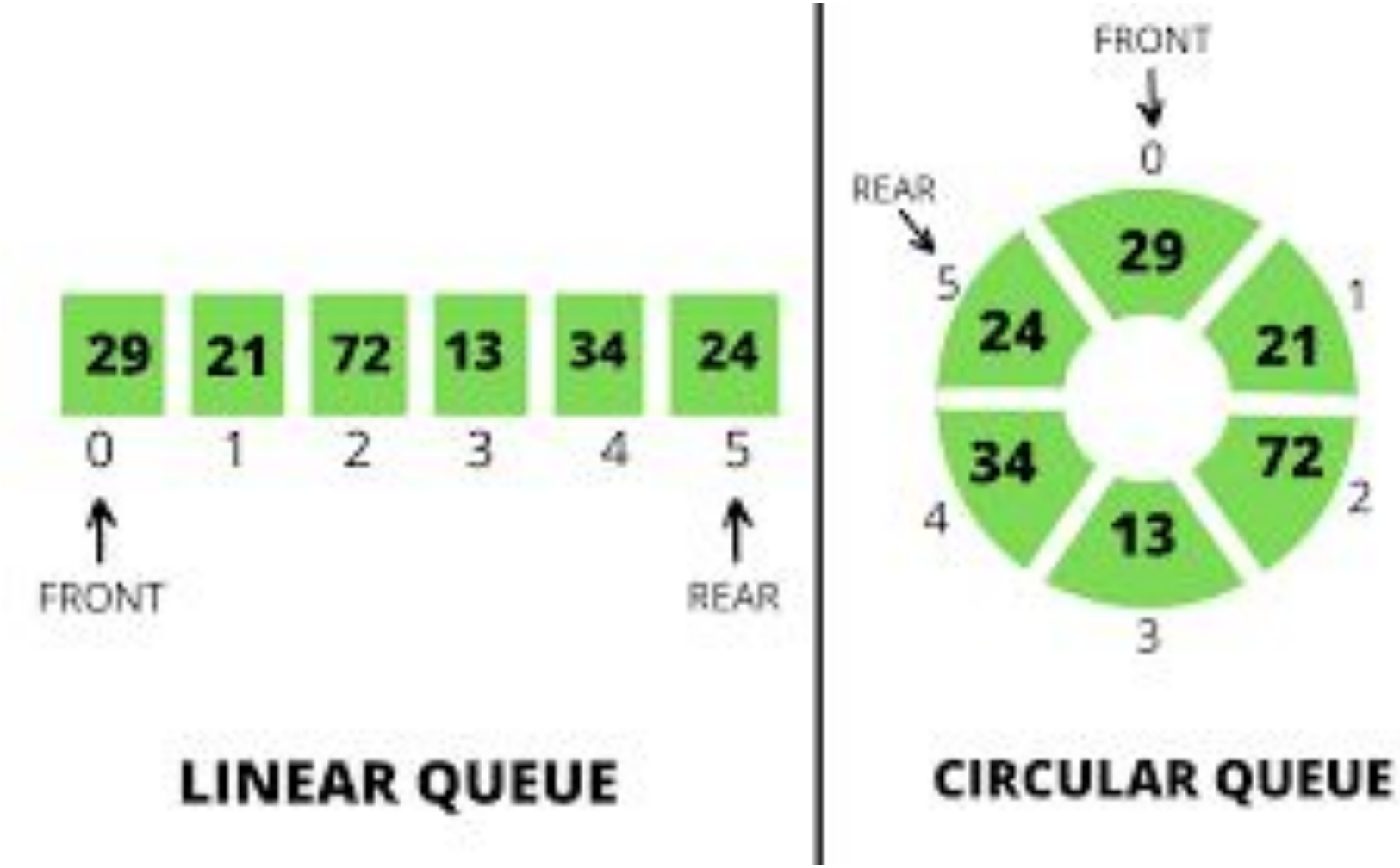


Figure 8.11 Linked queue after deletion of an element

```
IF FRONT = NULL
    Write "Underflow"
    Go to Step 5
[END OF IF]
SET PTR = FRONT
IF  FRONT = REAR
    SET FRONT = REAR = NULL
ELSE
SET FRONT = FRONT -> NEXT
FREE PTR
END
```

Circular Queue



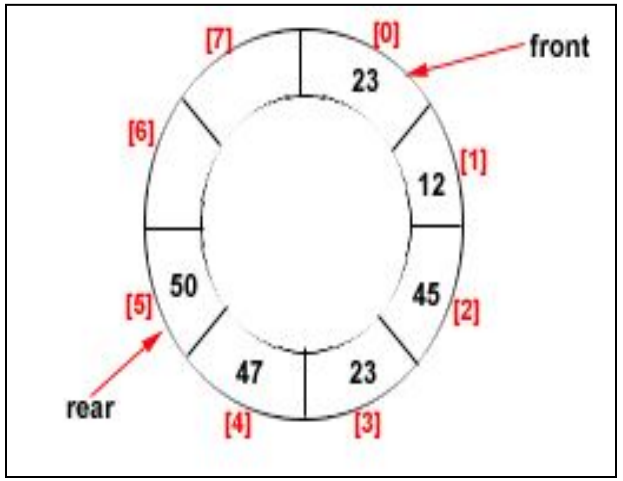
Drawback of Linear Queue

• Once the queue is full, even though few elements from the front are deleted and some occupied space is relieved, it is not possible to add anymore new elements, as the **rear has already reached the Queue's rear most position**.

Circular Queue

• This queue is not linear but circular.:

• In circular queue, once the **Queue is full** the "First" index of the Queue becomes the "Rear" most index, if and only if the "Front" element has moved forward. otherwise it will be a "Queue overflow" state.



Circular Queue having Rear = 5 and Front = 0

Algorithms for Insert Operations in Circular Queue

For Insert Operation

Insert-Circular-Q(CQueue, Rear, Front, N, Item)

Here, **CQueue** is a circular queue.

Rear represents the location in which the data element is to be inserted and

Front represents the location from which the data element is to be removed.

N is the maximum size of CQueue and

Item is the new item to be added.

Initailly Rear = -1 and Front = -1.

1. If Front = -1 and Rear = -1 then Set Front = Rear = 0 and go to step 5.
2. Else If Front =0 and Rear = N-1 or Front = Rear + 1
   then Print: “Circular Queue Overflow” and Return.
3. Else If Rear = N -1 then Set Rear := 0 and go to step 5.
4. Else Rear = Rear + 1
5. CQueue [Rear] := Item
6. Return

For Delete Operation

Delete-Circular-Q(CQueue, Front, Rear, Item)

CQueue is the place where data are stored.

Rear represents the location in which the data element is to be inserted and

Front represents the location from which the data element is to be removed. Front element is assigned to **Item**.

Initially, Front = -1.

*..Delete without Insertion

1. If Front = -1 then Print: “Circular Queue Underflow” and Return.
2. Set Item := CQueue [Front]
3. If Front = N – 1 then Set Front = 0 and Return.
4. If Front = Rear then Set Front = Rear = -1 and Return.
5. Set Front := Front + 1
6. Return.

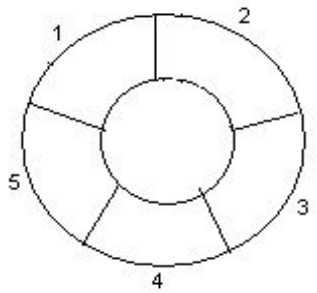
Example- ENQUEUE
Circular queue with N = 5.

(If Index starts with 0)

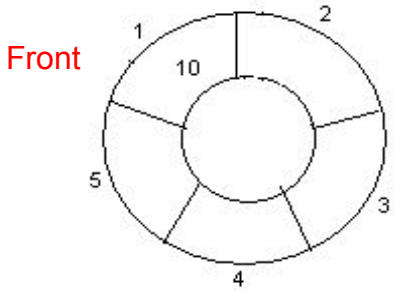
Initailly Rear = -1 and Front = -1.

1. If Front = -1 and Rear = -1 then Set Front :=0 and go to step 4.
2. If Front =0 and Rear = N-1 or Front = Rear + 1
   then Print: "Circular Queue Overflow" and Return.
3. If Rear = N -1 then Set Rear := 0 and go to step 4.
4. Set Rear:=Rear + 1 and CQueue [Rear] := Item.
5. Return

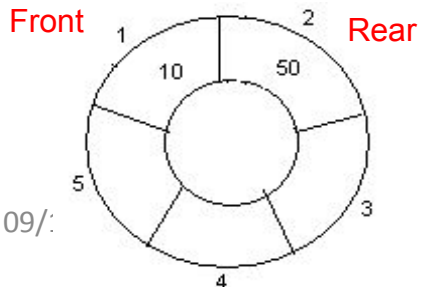
1. Initially, Rear = -1, Front =-1



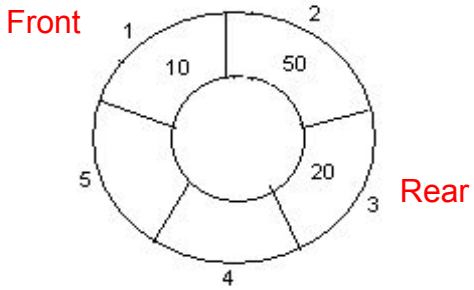
2. Insert 10, Rear = 0, Front = 0.



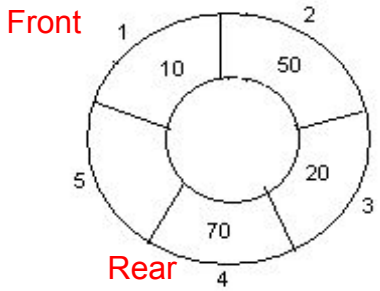
3. Insert 50, Rear = 1, Front = 0.



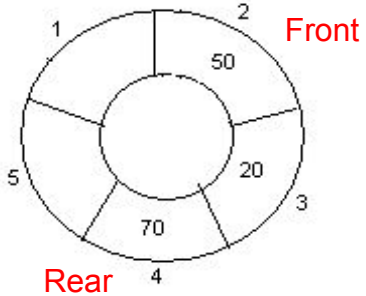
4. Insert 20, Rear = 2, Front = 0.



5. Insert 70, Rear = 3, Front = 0.



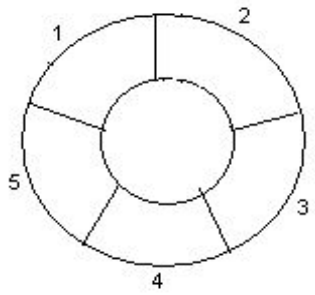
6. Delete front, Rear = 3, Front =1.



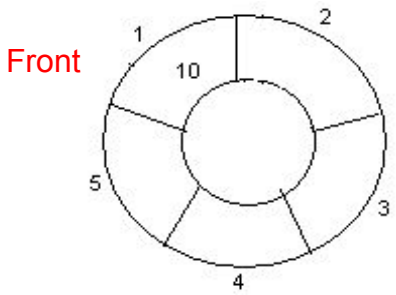
Example- ENQUEUE
Circular queue with N = 5.

(Assume Index starts with 1)

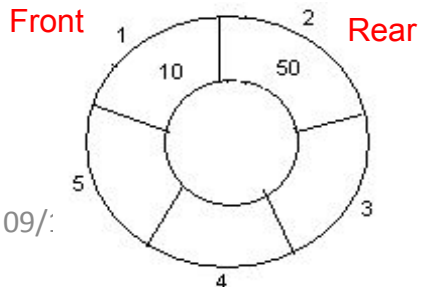
1. Initially, Rear = 0, Front = 0.



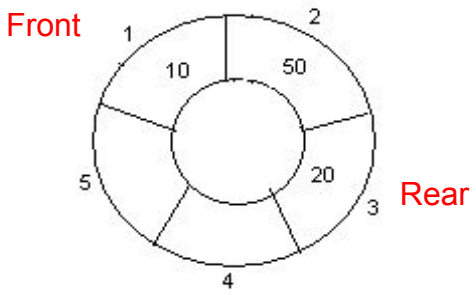
2. Insert 10, Rear = 1, Front = 1.



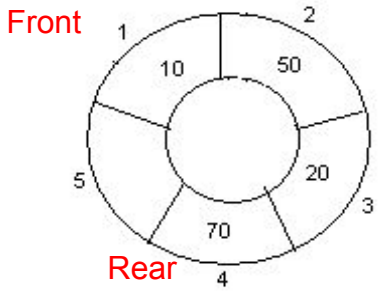
3. Insert 50, Rear = 2, Front = 1.



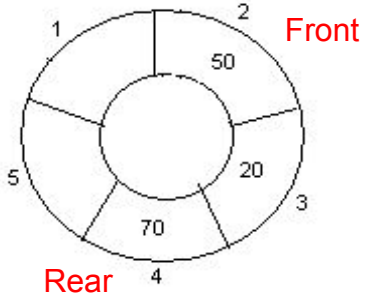
4. Insert 20, Rear = 3, Front = 1.



5. Insert 70, Rear = 4, Front = 1.

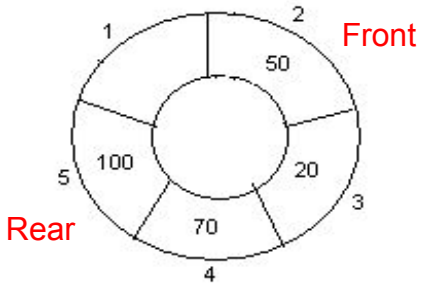


6. Delete , Rear = 4, Front = 2.

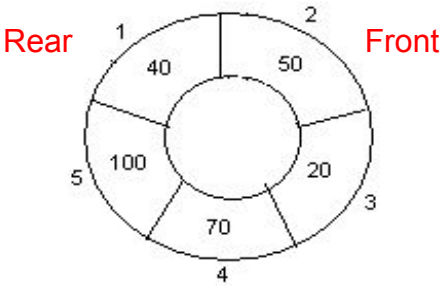


ENQUEUE/DEQUEUE
Circular queue with N = 5.

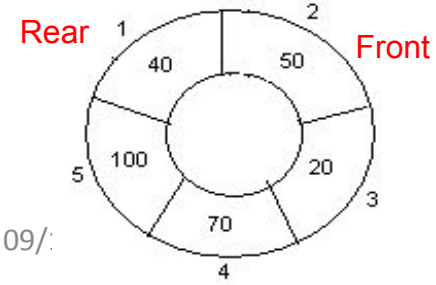
7. Insert 100, Rear = 5, Front = 2.



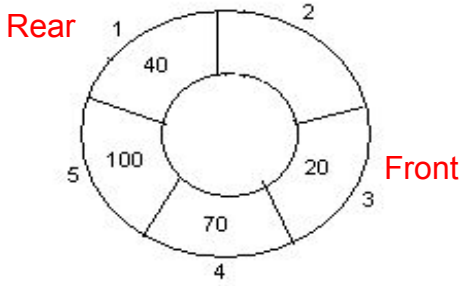
8. Insert 40, Rear = 1, Front = 2.



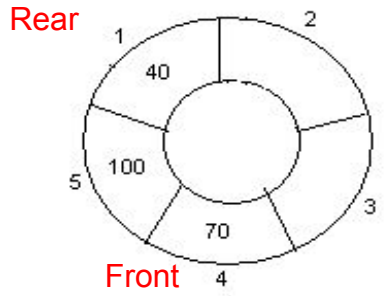
9. Insert 140, Rear = 1, Front = 2.
As Front = Rear + 1, so Queue overflow.



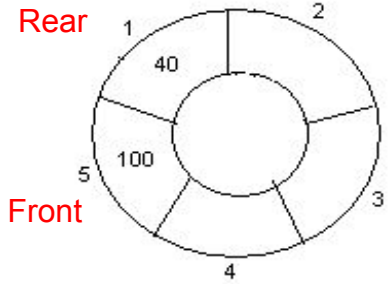
10. Delete, Rear = 1, Front = 3.



11. Delete, Rear = 1, Front = 4.



12. Delete, Rear = 1, Front = 5.



Example- ENQUEUE / DEQUEUE
Circular queue with N = 5.

(Index starts with 0)

1. Initially empty Queue
2. Insert 10,
3. Insert 50,
4. Insert 20,
5. Insert 70,
6. Delete front,.
7. Insert 100.
8. Insert 40.
9. Insert 140.
10. Delete front,
11. Delete front.
12. Delete front.

Initially, Front = -1.

1. If Front = -1 then Print: “Circular Queue Underflow” and Return.
2. Set Item := CQueue [Front]
3. If Front = N – 1 then Set Front = 0 and Return.
4. If Front = Rear then Set Front = Rear = -1 and Return.
5. Set Front := Front + 1
6. Return.

Initailly Rear = -1 and Front = -1.

1. If Front = -1 and Rear = -1 then Set Front :=0 and go to step 4.
2. If Front =0 and Rear = N-1 or Front = Rear + 1
   then Print: “Circular Queue Overflow” and Return.
3. If Rear = N -1 then Set Rear := 0 and go to step 4.
4. Set Rear:=Rear + 1 and CQueue [Rear] := Item.
5. Return

Example- ENQUEUE / DEQUEUE
Circular queue with N = 5.

(Index starts with 0)

1. Initially, Rear = -1, Front =-1

2. Insert 10, Rear = 0, Front = 0.

3. Insert 50, Rear = 1, Front = 0.

Rear

4. Insert 20, Rear = 2, Front = 0.

5. Insert 70, Rear = 3, Front = 0.

6. Delete front, Rear = 3, Front =1.

7. Insert 100, Rear = 4, Front = 1.

8. Insert 40, Rear = 0, Front = 1.

9. Insert 140, Rear = 0, Front = 1.
   As Front = Rear + 1, so Queue overflow.

10. Delete front, Rear = 0, Front = 2.

11. Delete front, Rear = 0, Front = 3.

12. Delete front, Rear = 0, Front = 4.

Double Ended Queue

Double ended queues, called **deques** for short, are a generalized form of the queue. It is exactly like a queue except that elements can be added to or removed from the **head** *or* the **tail**.

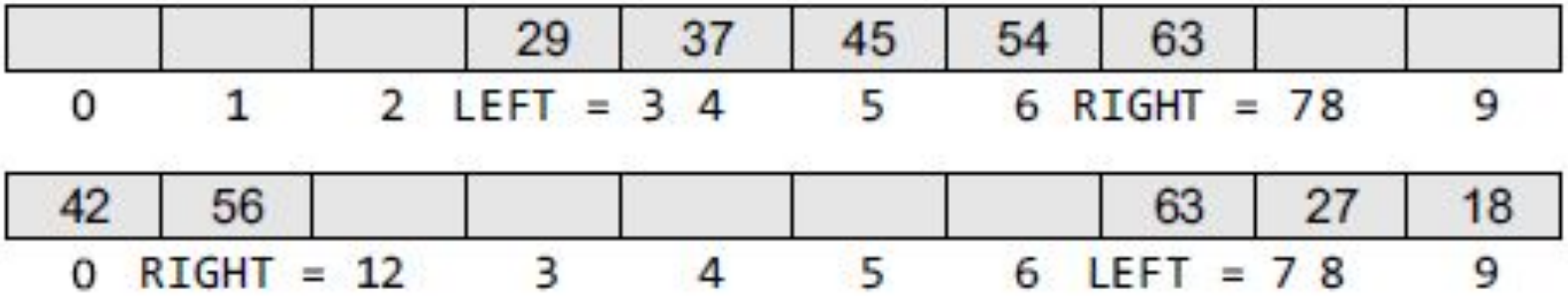
However, no element can be added and deleted from the middle.

In the computer's memory, a deque is implemented using either a circular array or a circular doubly linked list.

In a deque, 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, Dequeue[N–1] is followed by Dequeue[0].

There are two variants of a double-ended queue. They include :

- *Input restricted deque* In this, insertions can be done only at one of the ends, while deletions can be done from both ends.
- *Output restricted deque* In this deletions can be done only at one of the ends, while insertions can be done on both ends.



Priority Queues

A priority queue is 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.

The general rules of processing the elements of a priority queue are

- 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.