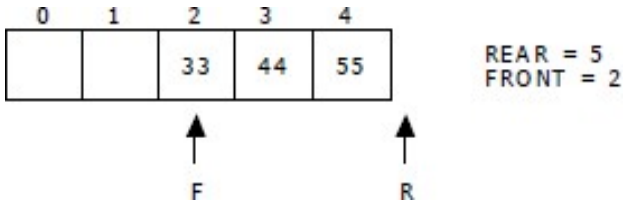
CIRCULAR QUEUE

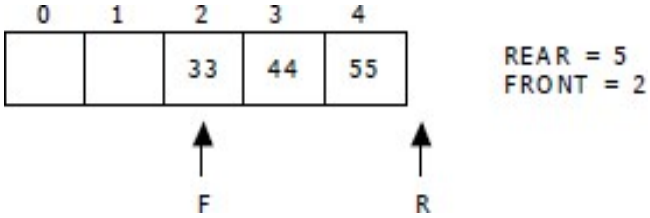
A more efficient queue representation is obtained by regarding the array Q[MAX] as circular. Any number of items could be placed on the queue. This implementation of a queue is called a circular queue because it uses its storage array as if it were a circle instead of a linear list. There are two problems associated with linear queue. They are:

- Time consuming: linear time to be spent in shifting the elements to the beginning of the queue.
- Signaling queue full: even if the queue is having vacant position.

For example, let us consider a linear queue status as follows:



Next insert another element, say 66 to the queue. We cannot insert 66 to the queue as the rear crossed the maximum size of the queue (i.e., 5). There will be queue full signal. The queue status is as follows:

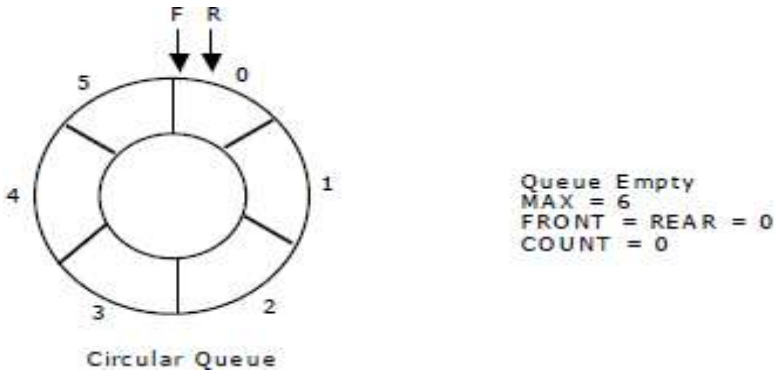


This difficulty can be overcome if we treat queue position with index zero as a position that comes after position with index four then we treat the queue as a **circular queue.**

In circular queue if we reach the end for inserting elements to it, it is possible to insert new elements if the slots at the beginning of the circular queue are empty.

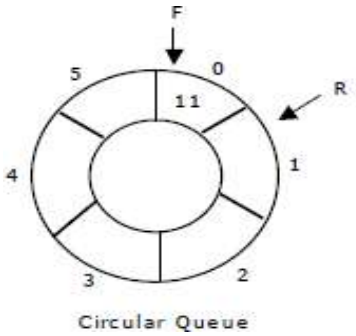
Representation of Circular Queue:

Let us consider a circular queue, which can hold maximum (MAX) of six elements. Initially the queue is empty.



Circular Queue

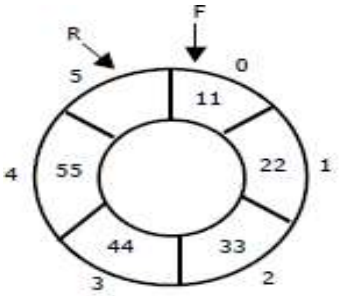
Now, insert 11 to the circular queue. Then circular queue status will be:



Circular Queue

FRONT = 0
REAR = (REAR + 1) % 6 = 1
COUNT = 1

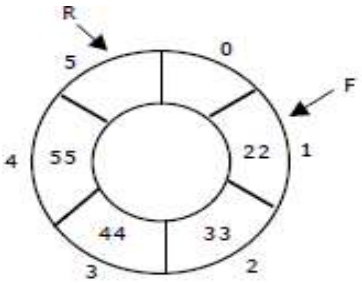
Insert new elements 22, 33, 44 and 55 into the circular queue. The circular queue status is:



Circular Queue

FRONT = 0
REAR = (REAR + 1) % 6 = 5
COUNT = 5

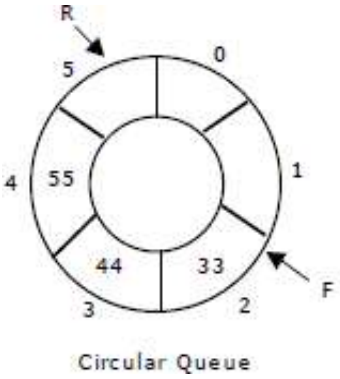
Now, delete an element. The element deleted is the element at the front of the circular queue. So, 11 is deleted. The circular queue status is as follows:



Circular Queue

FRONT = (FRONT + 1) % 6 = 1
REAR = 5
COUNT = COUNT - 1 = 4

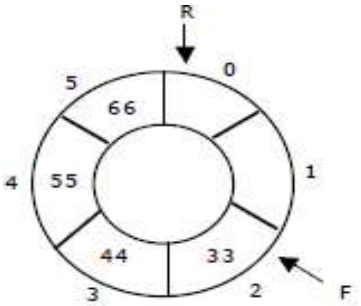
Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 22 is deleted. The circular queue status is as follows:



Circular Queue

FRONT = (FRONT + 1) % 6 = 2
REAR = 5
COUNT = COUNT - 1 = 3

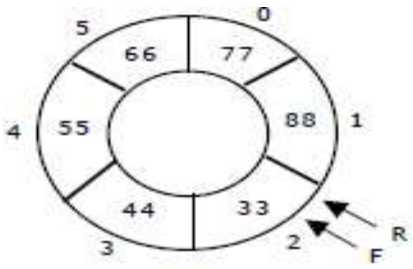
Again, insert another element 66 to the circular queue. The status of the circular queue is:



Circular Queue

FRONT = 2
REAR = (REAR + 1) % 6 = 0
COUNT = COUNT + 1 = 4

Now, insert new elements 77 and 88 into the circular queue. The circular queue status is:



Circular Queue

FRONT = 2, REAR = 2
REAR = REAR % 6 = 2
COUNT = 6

Now, if we insert an element to the circular queue, as COUNT = MAX we cannot add the element to circular queue. So, the circular queue is *full*.

Operations on Circular queue:

a.enqueue() or insertion(): This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

```
void insertCQ()
{
     int  data;
     if(count ==MAX)
    {
       printf("\n Circular Queue is Full");
     }
    else
    {
       printf("\n Enter data: ");
       scanf("%d", &data);
       CQ[rear] = data;
       rear = (rear + 1) % MAX;
       count ++;
printf("\n Data Inserted in the Circular
Queue ");
    }
}
```

Algorithm: procedure of insertCQ():

Step-1:START
Step-2: if count==MAX then
        Write “Circular queue is full”
Step-3:otherwise
      3.1: read the data element
      3.2: CQ[rear]=data
      3.3 : rear=(rear+1)%MAX
      3.4 : count=count+1
Step-4:STOP

b.dequeue() or deletion(): This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

```
void deleteCQ()
{
if(count ==0)
{
printf("\n\nCircular Queue is Empty..");
}
else
{
printf("\n Deleted element from Circular
Queue is %d ", CQ[front]);
front = (front + 1) % MAX;
count --;
}
}
```

Algorithm: procedure of deleteCQ():

Step-1:START
Step-2: if count==0 then
Write “Circular queue is empty”
Step-3:otherwise
3.1: print the deleted element
3.2: front=(front+1)%MAX
3.3: count=count-1
Step-4:STOP

c.dispaly(): This function is used to display the list of elements in the circular queue.

```
void displayCQ()
{
int i, j;
if(count ==0)
{
printf("\n\n\t Circular Queue is Empty ");
}
else
{
printf("\n Elements in Circular Queue are:
");
j = count;
for(i = front; j != 0; j--)
{
printf("%d\t", CQ[i]);
i = (i + 1) % MAX;
}
}
}
```

Algorithm: procedure of displayCQ():

Step-1:START
Step-2: if count==0 then
Write “Circular queue is empty”
Step-3:otherwise
3.1: print the list of elements
3.2: for i=front to j!=0
3.3: print CQ[i]
3.4: i=(i+1)%MAX
Step-4:STOP

Deque:

In the preceding section we saw that a queue in which we insert items at one end and from which we remove items at the other end. In this section we examine an extension of the queue, which provides a means to insert and remove items at both ends of the queue. This data structure is a ***deque***. The word ***deque*** is an acronym derived from ***double-ended queue***. Below figure shows the representation of a deque.

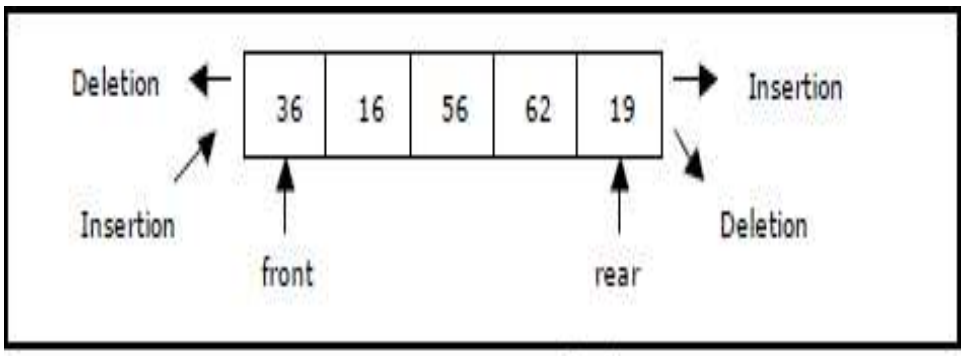


Figure Representation of a deque.

deque provides four operations. Below 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.

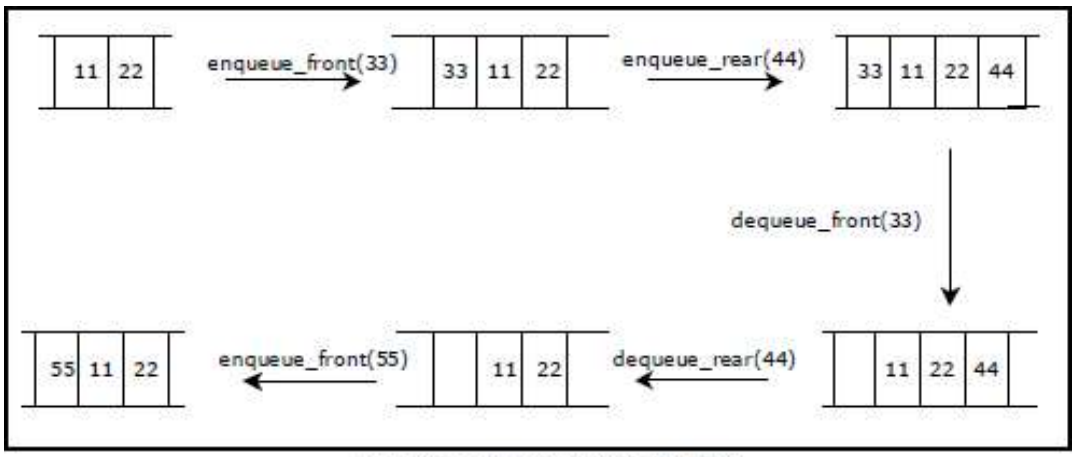


Figure . Basic operations on deque

There are two variations of deque. They are:

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

An Input restricted deque is a deque, which allows insertions at one end but allows deletions at both ends of the list.

An output restricted deque is a deque, which allows deletions at one end but allows insertions at both ends of the list.

Priority Queue:

A **priority queue** is a collection of elements such that each element has been assigned a priority. We can insert an element in priority queue at the rare position. We can delete an element from the priority queue based on the elements priority and such that the order in which elements are deleted and processed comes from the following rules:

1. An element of higher priority is processed before any element of lower priority.
2. Two elements with same priority are processed according to the order in which they were added to the queue. It follows FIFO or FCFS(First Comes First serve) rules.

We always remove an element with the highest priority, which is given by the minimal integer priority assigned.

A prototype of a priority queue is time sharing system: programs of high priority are processed first, and programs with the same priority form a standard queue. An efficient implementation for the Priority Queue is to use heap, which in turn can be used for sorting purpose called heap sort

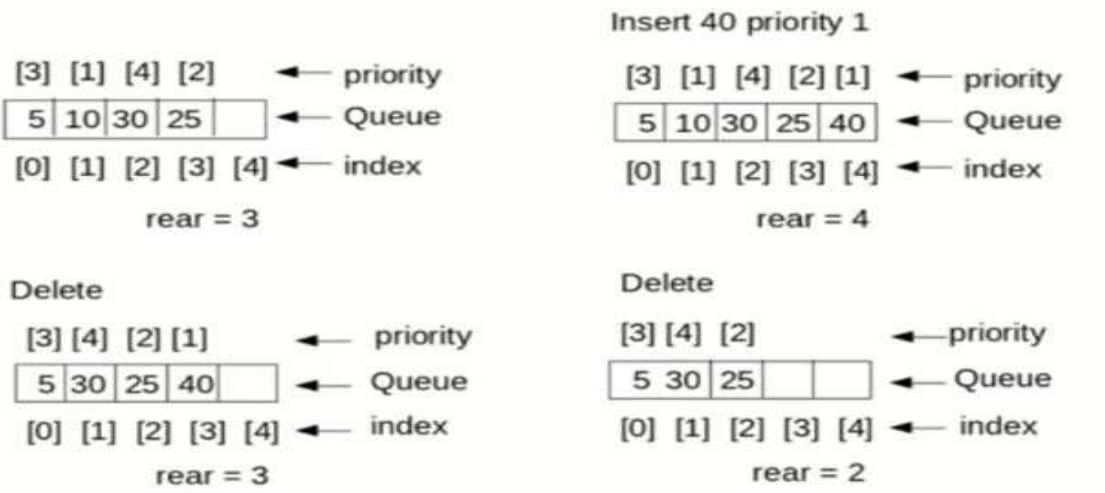
Priority queues are two types:

1. Ascending order priority queue
2. Descending order priority queue

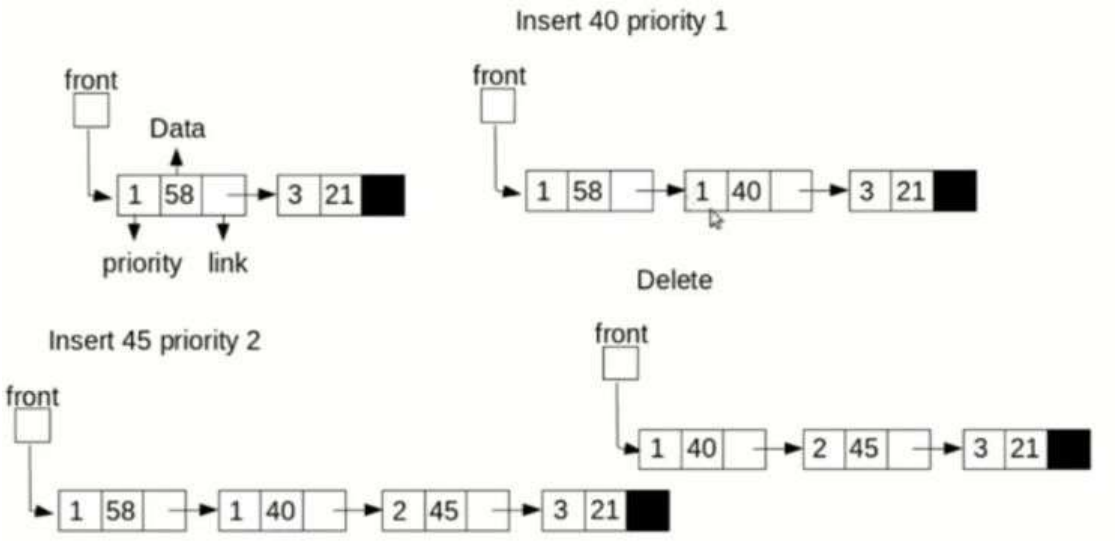
1. Ascending order priority queue: It is Lower priority number to high priority number.
Examples: order is 1,2,3,4,5,6,7,8,9,10
2. Descending order priority queue: It is high priority number to lowest priority number.
Examples: Order is 10,9,8,7,6,5,4,3,2,1
Implementation of Priority Queue:
Implementation of priority queues are two types:

1. Through Queue(Using Array)
2. Through Sorted List(Using Linked List)

1. Through Queue (Using Array): In this case element is simply added at the rear end as usual. For deletion, the element with highest priority is searched and then deleted.

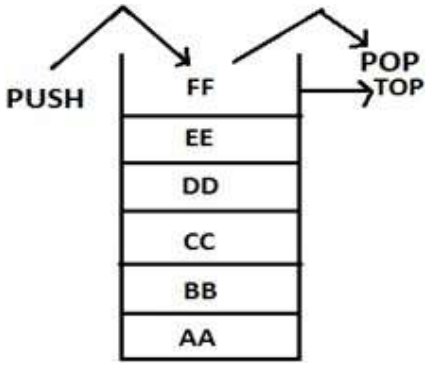


2. Through sorted List (Using Linked List): In this case insertion is costly because the element insert at the proper place in the list based on the priority. Here deletion is easy since the element with highest priority will always be in the beginning of the list.



1. Difference between stacks and Queues?

stacks	Queues
1.A stack is a linear list of elements in which the element may be inserted or deleted at one end. 2. In stacks, elements which are inserted last is the first element to be deleted. 3.Stacks are called LIFO (Last In First Out)list 4.In stack elements are removed in reverse order in which thy are inserted. 5.suppose the elements a,b,c,d,e are inserted in the stack, the deletion of elements will be e,d,c,b,a. 6.In stack there is only one pointer to insert and delete called "Top". 7.Initially top=-1 indicates a stack is empty. 8.Stack is full represented by the condition TOP=MAX-1(if array index starts from '0'). 9.To push an element into a stack, Top is incremented by one 10.To POP an element from stack,top is decremented by one.	1.A Queue is a linerar list of elements in which the elements are added at one end and deletes the elements at another end. 2. . In Queue the element which is inserted first is the element deleted first. 3. Queues are called FIFO (First In First Out)list. 4. In Queue elements are removed in the same order in which thy are inserted. 5. Suppose the elements a,b,c,d,e are inserted in the Queue, the deletion of elements will be in the same order in which thy are inserted. 6. In Queue there are two pointers one for insertion called "Rear" and another for deletion called "Front". 7. Initially Rear=Front=-1 indicates a Queue is empty. 8.Queue is full represented by the condition Rear=Max-1. 9.To insert an element into Queue, Rear is incremented by one. 10.To delete an element from Queue, Front is

11.The conceptual view of Stack is as follows:  	incremented by one. 11.The conceptual view of Queue is as follows: