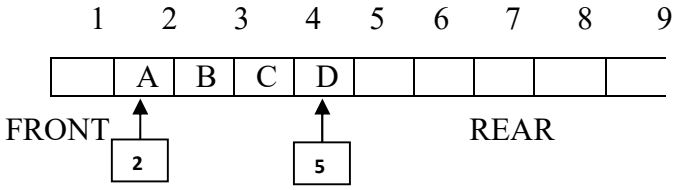
GIVE CONCEPT OF QUEUE, AND COMPARE QUEUE AND STACK. [7]

➔ QUEUE :-

A queue is a linear list of elements in which deletion can take place only at one end called the **FRONT**, and insertion can take place only at the other end called **REAR.**



Queues are also called **FIFO [First In First Out]** list (or) **FCFS [First Come First Serve]** list, i.e., Data items in such a list is processed in the same order as it was received that is on **FIFO** (or) **FCFS** basis. The first element in a queue will be the first element to come out of the queue.

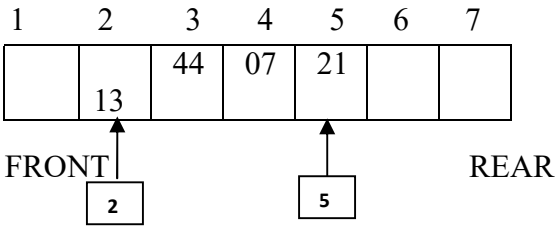
➔ Memory Representation Of Queue :-

Queues can be represented in memory by: linked list as well as arrays

Array representation of queue:

Queues will be maintained by a linear array called QUEUE and two pointer variable FRONT & REAR.
FRONT : Containing the location of front element of the queue.
REAR : Containing the location of rear element of the queue.

The condition **FRONT=NULL** indicate that queue is empty.

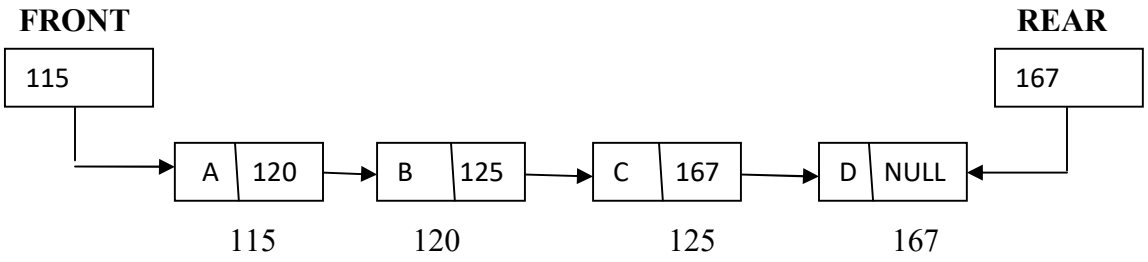


Linked List Representation Of Queue:

In this representation we do not bother on the size of the queue. Dynamically we create a node whenever it is required. hence linked list representation of queue is more efficient than array representation.

Each node in this consists of two parts information and link part. FRONT pointer points to the first node and REAR pointer points to the last node of the linked list.

Example:



➔ **Types of queues :-**

1. Ordinary queue / Simple queue
2. Priority queue
3. Circular queue
4. Dequeue
   4.1 input restricted dequeue
   4.2 output restricted dequeue

Operations on queues:

The following operations can be performed on queues

- Insert
- Delete
- **Front:** Get the front item from queue.
- **Rear:** Get the last item from queue.

1) INSERTION:

QINSERT [QUEUE, REAR, FRONT, ITEM, N]
Given FRONT & REAR pointers to the first and last elements of queue, this consists of N elements in a linear array QUEUE. This algorithm inserts a new value stored in ITEM in QUEUE.

Step 1: [queue is full]
If REAR>=N, then
Write "overflow"
Exit

Step 2: REAR=REAR+1 [Incrementation]

Step 3: [INSERT AN ITEM]
QUEUE [REAR] =ITEM

STEP 4: [when queue is empty in the beginning]
IF FRONT=NULL
THEN FRONT=FRONT+1

Step 5: Exit

2) DELETION:

QDELETE [QUEUE, REAR, FRONT, ITEM, N]
Given FRONT & REAR pointers to the first and last elements of queue, which consists of N elements in a linear array QUEUE . This algorithm deletes a value from QUEUE .ITEM is the temporary variable.

Step 1: [If queue is empty]
If FRONT=NULL, then
Write "Underflow"
Exit

Step 2: ITEM=QUEUE [FRONT]
[Deleting the element]

Step 3: If FRONT=REAR
FRONT=REAR=0 [when queue contains only one element]
ELSE
FRONT=FRONT+1

STEP 4: Exit

A Program That Implements The Queue Using An Array Is Given As Follows

Example

```
#include <iostream>
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert() {
  int val;
  if (rear == n - 1)
  cout<<"Queue Overflow"<<endl;
  else {
    if (front == - 1)
    front = 0;
    cout<<"Insert the element in queue : "<<endl;
    cin>>val;
    rear++;
    queue[rear] = val;
  }
}
void Delete() {
  if (front == - 1 || front > rear) {
    cout<<"Queue Underflow ";
    return ;
  } else {
    cout<<"Element deleted from queue is : "<< queue[front] <<endl;
    front++;;
  }
```

```
}
void Display() {
  if (front == - 1)
  cout<<"Queue is empty"<<endl;
  else {
    cout<<"Queue elements are : ";
    for (int i = front; i <= rear; i++)
    cout<<queue[i]<<" ";
      cout<<endl;
  }
}
int main() {
  int ch;
  cout<<"1) Insert element to queue"<<endl;
  cout<<"2) Delete element from queue"<<endl;
  cout<<"3) Display all the elements of queue"<<endl;
  cout<<"4) Exit"<<endl;
  do {
    cout<<"Enter your choice : "<<endl;
    cin<<ch;
    switch (ch) {
      case 1: Insert();
      break;
      case 2: Delete();
      break;
```

```
        case 3: Display();
        break;
        case 4: cout<<"Exit"<<endl;
        break;
        default: cout<<"Invalid choice"<<endl;
      }
   } while(ch!=4);
   return 0;
}
```

The output of the above program is as follows

```
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice : 1
Insert the element in queue : 4
Enter your choice : 1
Insert the element in queue : 3
Enter your choice : 1
Insert the element in queue : 5
Enter your choice : 2
Element deleted from queue is : 4
Enter your choice : 3
Queue elements are : 3 5
Enter your choice : 7
Invalid choice
Enter your choice : 4
Exit
```

➔ Comparison Between QUEUE and STACK :-

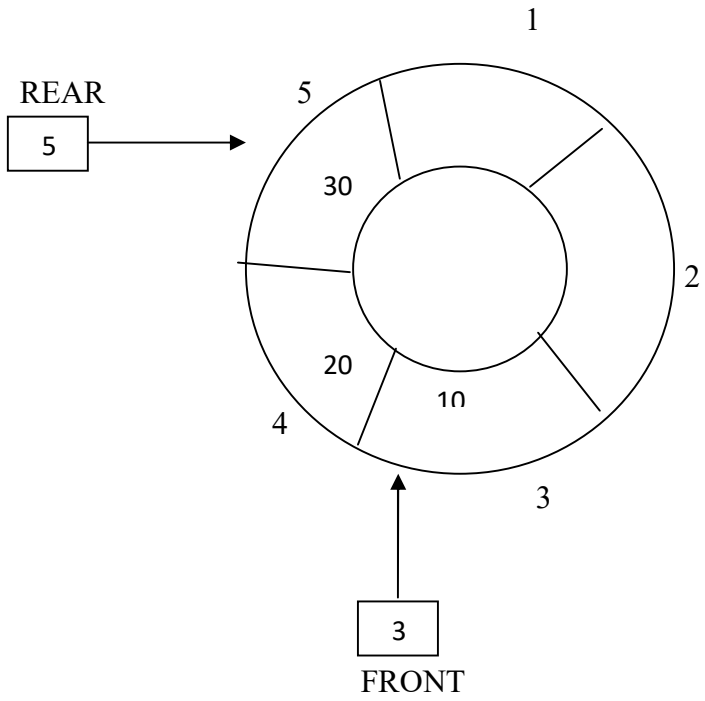
QUEUE	STACK
1) In QUEUE insertion and deletion operations are performed at different end.	1) In STACK insertion and deletion operations are performed at same end.
2) In QUEUE an element which is instead first is first to delete. So it is called FIFO [First In First Out].	2) In STACK an element which is instead last is first to delete. So it is called LIFO [Last In First Out].
3) In QUEUE two pointers are used called FRONT and REAR.	3) In STACK only one pointer is used called TOP.
4) In queue there is wastage of memory space.	4) In stack there is no wastage of memory space.
5) Students standing in a line at Fee counter are an example of QUEUE.	5) Plate counter at marriage reception is an example of STACK.

EXPLAIN CIRCULAR QUEUE.DESCRIBE USING ALGORITHM. [5]

➔ CIRCULAR QUEUE :-

Consider an array Q that contains N elements in which Q [0] comes after Q [N-1] in the array. When this method is used to construct a queue then queue is called CIRCULAR QUEUE.

In other words, a queue is called circular when the last room comes just before the first room. Below fig. Represents circular queue.



Above fig. represents CIRCULAR QUEUE, whose size is 5 & there are 3 elements present in the queue .i.e., Q [3] = 10, Q [4] = 20, Q [5] = 30

FRONT = 3
REAR = 5

➔ Operations performed on circular queue:

- Insertion
- Deletion

The following algorithm is use to insert the element in the circular queue.

→ **Assumptions :-**

QUEUE : A linear array used to represent queue DS.
REAR : A pointer variable containing the location of the last element of the queue.
FRONT : A pointer variable containing the location of the first element of the queue.
ITEM : A data value, which is to be inserted in queue.
N : Represent the maximum size of the queue.

→ **Algorithm :-**

Circular Q-INSERT [QUEUE, REAR, FRONT, ITEM, N]

Step 1 : [check overflow condition **]**
if (FRONT =1 && REAR = N) OR (REAR = FRONT - 1) THEN

```
        Write  "queue is full"
        EXIT.

Step 2 : if  FRONT = NULL
                FRONT = 1     [ When QUEUE is empty initially ]
                REAR = 1
         else if  ( REAR = N  and  FRONT ! = 1 )
                REAR = 1
         else
                REAR = REAR + 1
          [ END IF ]

Step 3 : [ INSERT an item into REAR position ]
         QUEUE [ REAR ] = ITEM

Step 4 : EXIT
```

The following algorithm is use to delete the element in the circular queue.

→ **Assumptions :-**

QUEUE : A linear array used to represent queue DS.
REAR : A pointer variable containing the location of the last element of the queue.
FRONT : A pointer variable containing the location of the first element of the queue.
ITEM : A data value, which is to be inserted in queue.
N : Represent the maximum size of the queue.

→ **Algorithm :-**

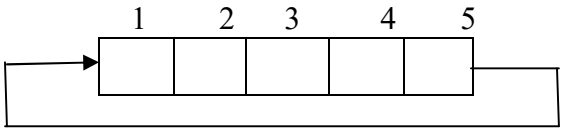
Circular Q-DELETE [QUEUE, REAR, FRONT, ITEM, N]

```
Step 1 : [ check underflow condition ]
         if  FRONT = NULL
         Write  "queue is empty"
         EXIT.
         [ END If ]

Step 2 : [ DELETE FRONT element from QUEUE ]
         ITEM = QUEUE [ FRONT ]

Step 3 : [ if QUEUE has only one element ]
         If  FRONT = REAR
                FRONT = NULL
                REAR = NULL
         else if
                FRONT = N
```

```
            FRONT = 1
      else
            FRONT  =  FRONT + 1
      [ END IF ]
```

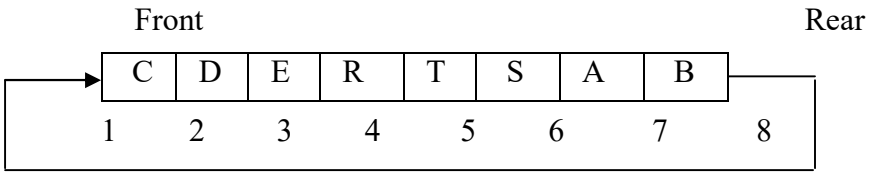
Step 4 : EXIT

WHAT WILL BE POSITION OF FRONT AND REAR IN EMPTY CIRCULAR QUEUE? [2]

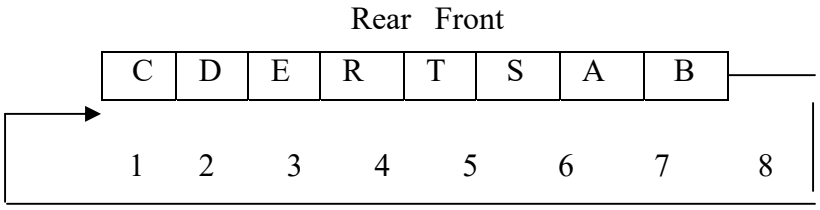


When circular queue is empty then the position of FRONT and REAR is **NULL.**

WHAT WILL BE POSITION OF FRONT AND REAR IN FULL CIRCULAR QUEUE ?[2]



OR



When the circular queue is full then the positions are

- **FRONT=1 and the position of REAR=N.**
- **OR REAR=FRONT-1**

/**** Program to Implement Circular Queue using Array ****/

```
#include<stdio.h>
#define SIZE 5
void insert();
void delet();
void display();
int queue[SIZE], rear=-1, front=-1, item;
void  main()
{
int ch;
do
{
printf("\n\n1.\tInsert\n2.\tDelete\n3.\tDisplay\n4.\tExit\n"); printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1:
insert();
break;
case 2:
delet();
break;
case 3:
display();
break;
case 4:
exit(0);
default:
printf("\n\nInvalid choice. Pleasr try again...\n");
}
} while(1);
getch();
}
void insert()
{
if((front==0 && rear==SIZE-1) || (front==rear+1))
printf("\n\nQueue is full.");
else
{
printf("\n\nEnter ITEM: ");
scanf("%d", &item);
if(rear == -1)
{
```

```
rear = 0;
front = 0;
}
else if(rear == SIZE-1)
 rear = 0;
else
rear++;
 queue[rear] = item;
 printf("\n\nItem inserted: %d\n", item);
 }
 }
 void delet()
 {
 if(front == -1)
 printf("\n\nQueue is empty.\n");
else
 {
 item = queue[front];
 if(front == rear)
 {
 front = -1;
 rear = -1;
 }
else if(front == SIZE-1)
 front = 0;
 else
 front++;
 printf("\n\nITEM deleted: %d", item);
 }
 }
 void display()
 {
 int i;
 if(front == -1)
 printf("\n\nQueue is empty.\n");
 else
{
 printf("\n\n");
 for(i=front; i<=rear; i++)
 printf("\t%d",queue[i]);
 }
 }
```

GIVE APPICATION OF QUEUE. [5]

➔ **APPLICATION OF QUEUE** :-

QUEUE is used when things do not have to be processed immediately, but have to be processed in First In First Out order. This property of queue makes it also useful in following kind of scenarios :

1) Used in time sharing system.
2) Used in network communication system.
3) Circular queues are used in Operating systems
4) Call center phone systems will use a queue to hold people in line until a service provider is free.
5) Buffers on MP3 players and portable CD players, iPod playlist, are maintained by queue.
6) Used for handling interrupts in programming a real time system

WRITE CONDITION OF OVERFLOW AND UNDERFLOW IN DEQUEUE. [2]

➔ **CONDITION OF OVERFLOW :-**

When the DEQUEUE is full then the value of FRONT is 1 and the value of REAR is N, where N=total element

➔ **CONDITION OF UNDERFLOW :-**

When the DEQUEUE is empty then the value of FRONT and REAR=NULL.

WHAT IS DOUBLE ENDED QUEUE? EXPLAIN INPUT RESTRICTED AND OUTPUT RESTRICTED DEQUEUE? WRITE ALGORITHM OF INPUT RESTRICTED DEQUEUE. [7]

➔ **DEQUES :-**

A deque is a linear list in which elements can be added or removed at either end but not in the middle. The term deque is a contraction of the name **Double-ended queue. Are made to or from either end of** Deque is linear list in which insertions & deletion are made to or from either end of structure such list is called DEQUEUE.

Deque is maintained by circular array; DEQUE with pointers LEFT and RIGHT, which point to the two ends of the deque. We assume that the elements extend from left end to the right end in the array. The term **"circular"** comes from the fact that we assume the DEQUE [1] comes after DEQUE [N] in the array. The condition LEFT = NULL indicates that DEQUE is empty.

There are two variations of a DEQUE namely
1) Input restricted DEQUE
2) Output restricted DEQUE

Which are intermediate between a deque and a queue

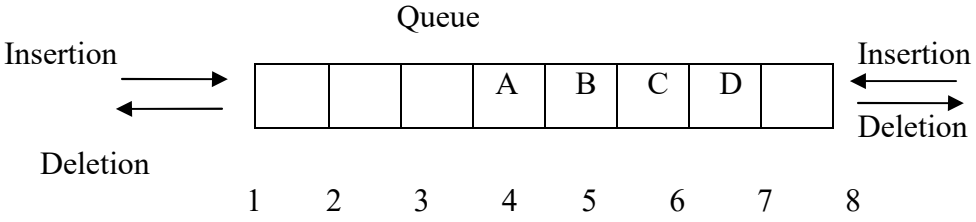
1) <u>INPUT restricted DEQUE</u> :-

An input restricted DEQUE which allows insertion at only one end of the list but allows deletions at both ends of the list.

2) <u>OUTPUT restricted DEQUE</u> :-

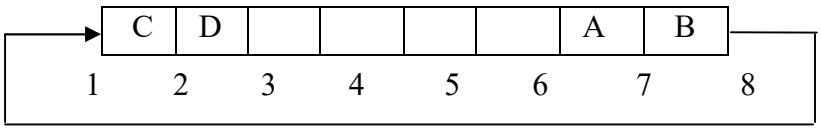
An output restricted DEQUE is a deque which allows deletions at only one end of the list that allows insertion at both ends of the list.

<u>Memory representation of DEQUES</u> :-



LEFT = 4
RIGHT = 7

In a Deque at both ends we can do insertion and deletions

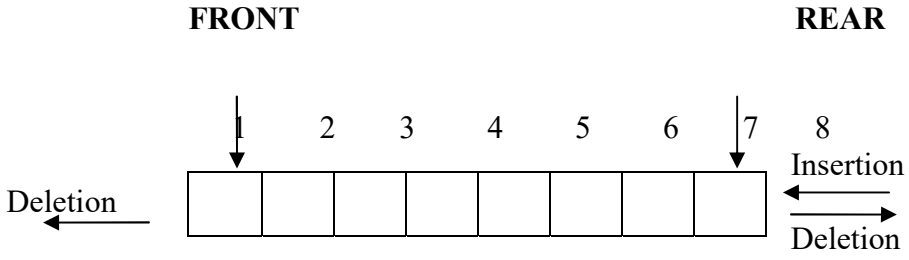


LEFT = 7
RIGHT = 2

Circular queue is same as Deque after last element it will point to first element.
i.e., A [N] → A [1]

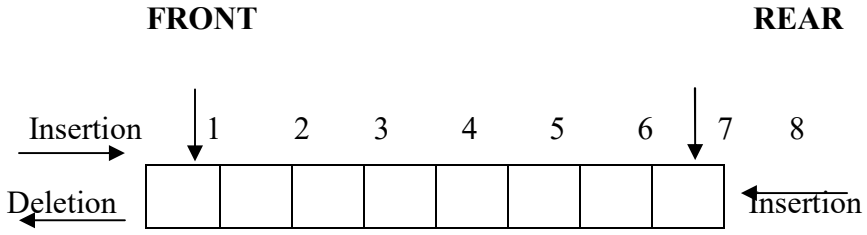
INPUT restricted Deque :-

Example :-



OUTPUT Restricted Deque :-

Example :-



In output restricted deque insert

➔ **Operations perform on dequeue :-**

1) Initialize() : make the queue empty
2) Empty() : determine if queue is empty
3) Full() : determine if queue is full
4) EnqueueF() : insert an element at the front end of the queue
5) EnqueueR() : insert an element at the rear end of the queue
6) dequeueR() :delete the rear element
7) dequeueF() : delete the front element
8) printf() : print elements of the queue

OR

Operations on Deque:
Mainly the following four basic operations are performed on queue:

insetFront(): Adds an item at the front of Deque.
insertLast(): Adds an item at the rear of Deque.
deleteFront(): Deletes an item from front of Deque.
deleteLast(): Deletes an item from rear of Deque.

In addition to above operations, following operations are also supported
getFront(): Gets the front item from queue.
getRear(): Gets the last item from queue.
isEmpty(): Checks whether Deque is empty or not.
isFull(): Checks whether Deque is full or not.

Application of Deque:

used in a Steal job scheduling algorithm

ALGORITHM:

Algorithm represents insertions at right end of dequeue:
Assumptions:

FRONT: Points to first element of a deque
REAR: Points to last element of a deque
DQ: is an array that represents deque
SIZE: indicates the size of deque
VALUE:is a variable that contain new element

Algorithm:DQ_Rins(VALUE,FRONT,REAR,DQ,SIZE)

(a)[Check rear end of deque]
If REAR>=SIZE-1 then
Print "overflow at right end"
Exit
(b)[Increment REAR variable]
REAR=REAR+1
(C)[Put new element in deque]
DQ[REAR]=VALUE

(d)[set the value of FRONT variable]
IF FRONT== -1 then
FRONT=0

(e)[Exit from the function]
Exit

Insertion at left end of deque:

DQ_Lins(VALUE,FRONT,REAR,DQ,SIZE)

Step 1: if front==0 then
Print "overflow at left end"
Exit

Step 2: if front ==-1 then
Rear=front=size-1
Else
Front=front-1

Step 3: DQ[front]=value

Step 4: exit

Deletion at right end of deque:

DQ_Rdel(VALUE,FRONT,REAR,DQ,SIZE)

Step 1: if rear ==-1 then
Print "underflow at right end"
exit

Step 2: value=DQ[rear]

Step 3: if (front=rear) then
Front=rear=-1
Else
Rear=rear-1

Step 4: Exit

Deletion at left end of deque:

DQ_Ldel(VALUE,FRONT,REAR,DQ,SIZE)

Step 1 : if front==-1 then
Print "deque is underflow"
Exit

Step 2: value=DQ[front]

Step 3: if front==rear then
Front=rear=-1
Else
Front=front+1

Step 4: exit