**Experiment No 8:**

**A program to implement Queue.**

**Aim:** A program for inserting and deleting an element in a circular queue.

**Theory:**

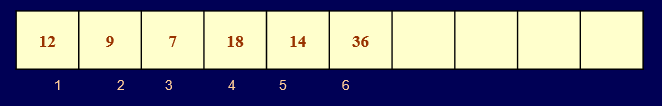
Queue is an important data structure which stores its elements in an ordered manner. Take for example the analogies given below.

People waiting for bus. The first person standing in the line will be the first one to get into the bus.

A queue is a FIFO (First In First Out) data structure in which the element that was inserted first is the first one to be taken out. The elements in a queue are added at one end called the rear and removed from the other one end called front.

**Array Representation Of Queue**

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

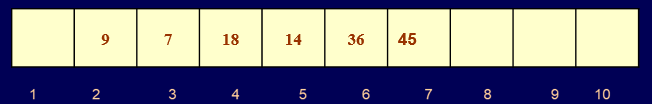
****

Here, front = 1 and rear = 6. If we want to add one more value in the list say with value 45, then rear would be incremented by 1 and the value would be stored at the position pointed by rear. The queue after addition would be as shown in figure

****

Here, front = 1 and rear = 7. Every time a new element has to be added, we will repeat the same procedure.

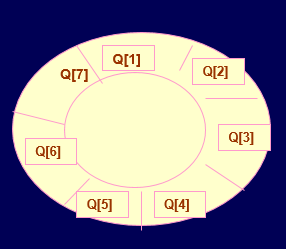
Now, if we want to delete an element from the queue, then the value of front will be incremented. Deletions are done from only this end of the queue. The queue after deletion will be as shown in figure

****

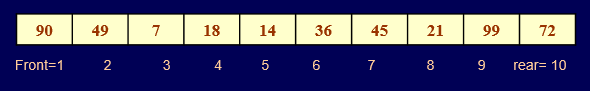
Here, front = 2 and rear = 7.

However, before inserting an element in the queue we must check for overflow conditions. An overflow will occur when we will try to insert an element into a queue that is already full. When Rear = MAX, where MAX is the size of the queue that is, MAX specifies the maximum number of elements that the queue can hold.

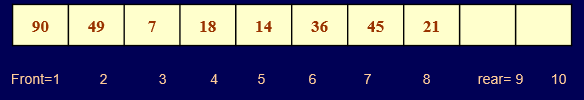
Similarly, before deleting an element from the queue, we must check for underflow condition. An underflow condition occurs when we try to delete an element from a queue that is already empty. If front = NULL and rear = NULL, this means there is no element in the queue.

****

The circular queue will be full, only when front=0 and rear = Max . A circular queue is implemented in the same manner as the linear queue is implemented. The only difference will be in the code that performs insertion and deletion operations. For insertion we will now have to check for three conditions which are as follows:

If front=0 and rear= MAX , then print that the circular queue is full. 

**If rear != MAX, then the value will be inserted and rear will be incremented as illustrated in figure**

****

**Algorithm:**

**Algorithm For insertion:**

QINSERT(QUEUE,N,FRONT,REAR,ITM)

These procedure inserts an element ITEM into a queue.

1.[Queue already filled?]

If FRONT=1 and REAR=N, or if FRONT=REAR+1, then:

Write: OVERFLOW, and Return

2.[Find new value of REAR.]

If Front:=NULL, then:[Queue initially empty.]

Set FRONT:=1 and REAR:=1

Else if REAR=N,then:

Set REAR:=1.

Else:

Set REAR=REAR+1.

[End of If Structure]

3.Set QUEUE[REAR]:=ITEM.[This inserts new element.]

4.Return

**Algorithm For Deletion:**

**QDELETE(QUEUE,N,FRONT,REAR,ITEM)**

This procedure deletes an element from queue and assigns it to the variable ITEM.

1.[Queue already empty?]

If FRONT:=NULL, then: Write: UNDERFLOW, and Return.

2.Set ITEM:=QUEUE[FRONT.]

3.[Find new value of FRONT.]

If FRONT=REAR , then:[Queue has only one element to start.]

Set FRONT:=NULL and REAR:=NULL.

Else if FRONT=N, then:

Set FRONT:=1.

Else:

Set FRONT:=FRONT+1.

[End of IF structure.]

4.Return.

**PROGRAM:**

**OUTPUT**

**CONCLUSION:**