As we have discussed earlier that there arises a problem when the rear reaches the end of the array we are unable to add more elements to the queue, although there are free spaces in the queue. To remove this problem all we need is to reset the rear pointer to 0 as soon as it exceeds the maximum number of elements in the queue.

5.7.1 A Circular Queue

A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full. In other words if we have a queue Q of say n elements, then after inserting an element last (*i.e.*, in the *n*-1th) location of the array the next element will be inserted at the very first location (*i.e.*, location with subscript 0) of the array. It is possible to insert new elements, if and only if those locations (slots) are empty. We can say that a circular queue is one in which the first element comes just after the last element. It can be viewed as a mesh or loop of wire, in which the two ends of the wire are connected together. Fig 5.5 shows a empty circular queue Q[5] which can accommodate five elements.

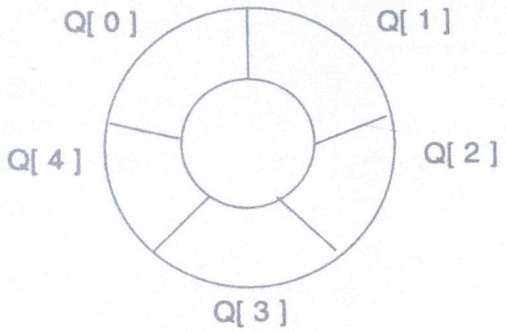


Fig. 5.5. Circular Queue

A circular queue overcomes the problem of unutilized space in linear queues implemented as arrays. A circular queue also have a Front and Rear to keep the track of the elements to be deleted and inserted and therefore to maintain the unique characteristic of the queue. The following assumptions are made :

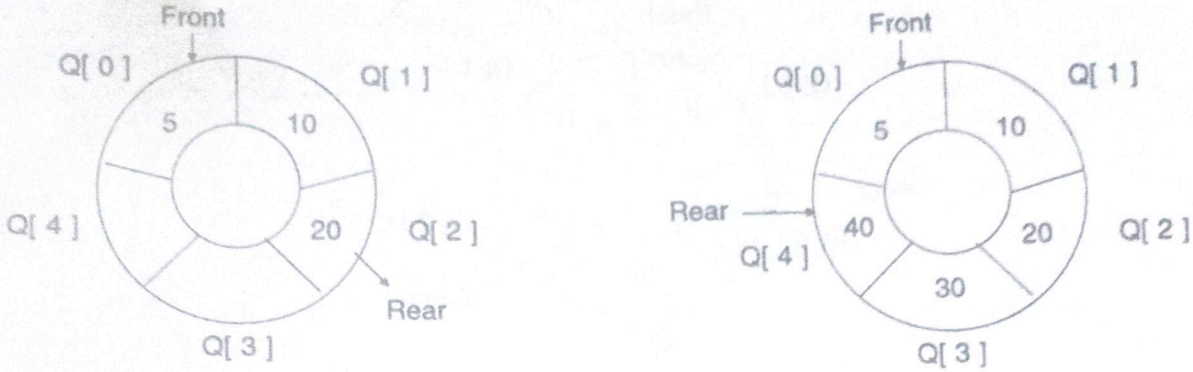
1. Front will always be pointing to the first element (as in the linear queue).
2. If Front = Rear the queue will be empty.
3. Each time a new element is inserted into the queue the Rear is incremented by one.

   Rear = Rear + 1
4. Each time an element is deleted from the queue the value of Front is incremented by one.

   Front = Front + 1

Insertion: Consider the circular queue shown above, the insertion in this queue will be same as with linear queue, we only have to keep track of Front and Rear with some extra logic. A Circular Queue is shown in the figure 5.6.*a*.

If more elements are added to the queue, it looks like shown in figure 5.6.*b*.



(*a*) Insertion in a Queue. (*b*)

Fig. 5.6.

Now the queue will be full. If we now try to add another element to the queue, as the new element is inserted from the Rear end, the position of the element to be inserted will be calculated by the relation:

```
Rear = (Rear + 1) % MAXSIZE
Queue[Rear] = Value
```

For the current queue (Fig 5.6.*b*.) the value of Rear is 4, and value of MAXSIZE is 5, hence

```
Rear = (Rear + 1) % MAXSIZE
Rear = (4 + 1) % 5
     = 0.
```

Note that Front is also pointing to Q[0] (Front = 0) and Rear also comes out to be 0 (*i.e.*, Rear is also pointing to Q[0]). Since Rear = Front, the Queue Overflow condition is satisfied, and our trial to add new element will flash the message "Queue Overflow" which avoids us to add new element.

Referring to the figure Fig 5.6(*c*). Consider the situation when we add an element to the queue. The Rear is calculated as follows :

```
Rear = (Rear + 1) % 5
Rear = (5 + 1) % 5
     = 0.
```

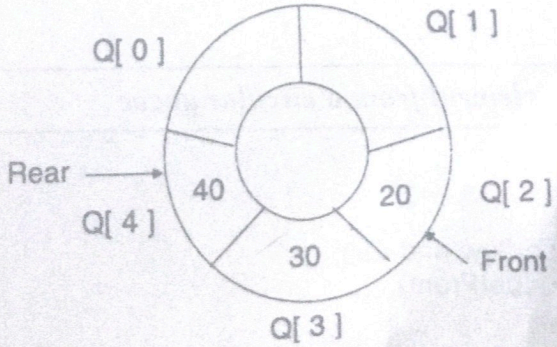


Fig. 5.6. (*c*)

The new element will be added to Q[Rear] or Q[0] location of the array and Rear is incremented by one (*i.e.*, Rear = Rear + 1 = 0 + 1) The next element will be added to location Q[1] of the array.

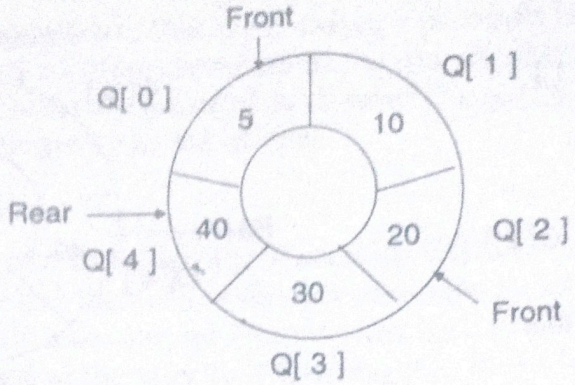


Fig. 5.7.

Deletion. The deletion method for a circular queue also requires some modification as compared to linear queues. The deletion is explained using Fig 5.7.

In the above figure Fig 5.7 the queue is full. Now if we delete one element from the queue, it will be deleted from the Front end. After deleting the front element, the front should be modified according to position of Front. That is if Front indicates to the last element of the circular queue then after deleting that element the Front should be again reset to 0 (Front = 0). Otherwise after every deletion the new position which Front should indicate will be given as :

Front = (Front + 1) % MAXSIZE

For example

Algorithms For Addition And Deletion In A Circular Queue (using Arrays)

Algorithm for adding an element in a circular queue

Algorithm

```
1.  If (Front = = (Rear + 1) % MAXSIZE)
    Write Queue Overflow and Exit.
    Else : Take the value
                If (Front = = –1)
                Set Front = Rear = 0
                Rear = ((Rear + 1) % MAXSIZE)
    [Assign Value] Queue [Rear] = value.
    [End if]
2.  Exit.
```

Algorithm for deleting an element from a circular queue

Algorithm

```
1.  If (Front = –1)
    Write Queue underflow and Exit.
    else : value = Queue(Front)
    if (Front = = Rear)
    Set Front = –1
    Rear = –1
    Else : Front = (Front + 1) % MAXSIZE
    [End If Structure]
2.  Exit.
```

EXAMPLE 5.2

```
#include <stdio.h>
#include <conio.h>
#define MAXSIZE 5
int cq[10] ;
int front = –1, rear = 0 ;
int choice ;
char ch ;
void main( )
{
    do
    {
        clrscr( ) ;
        printf ("--------1. Insert-------\n") ;
        printf ("------- 2. Delete--------\n") ;
        printf ("------- 3. Display--------\n") ;
        printf (\"-------4. exit-----------\n") ;
        printf ("Enter your choice\n") ;
        scanf ("%d", &choice) ;
        switch (choice)
        {
            case 1 : cqinsert( ) ;
                break ;
            case 2 : cqdelete( ) ;
                break ;
            case 3 : cqdisplay( ) ;
                break ;
            case 4 : return ;
        }
        fflush (stdin) ;
    }
    while (choice != 4) ;
}
cqinsert( )
{
    int num ;
    if (front = = (rear + 1) % MAXSIZE)
    {
        printf ("Queue is full\n") ;
        return ;
    }
    else
    {
        printf ("Enter the element to be inserted\n") ;
        scanf ("%d", &num) ;
```

```
            if (front = = -1)
            front = rear = 0 ;
            else
            rear = (rear + 1) % MAXSIZE ;
            cq[rear] = num ;
    }
    return ;
}
int cqdelete( )
{
    int num ;
        if (front = = -1)
        {
            printf ("Queue is Empty\n") ;
            return ;
        }
        else
        {
            num = cq[front] ;
            printf ("Deleted element is = %d\n", cq[front]) ;
            if (front = = rear)
                front = rear = -1 ;
                else
                front = (front + 1) % MAXSIZE ;
        }
        return (num) ;
}
cqdisplay( )
{
    int i ;
    if (front = = -1)
    {
        printf ("Queue is empty\n") ;
        return ;
    }
    else
    {
        printf ("\n The status of the queue\n") ;
        for (i = front ; i <= rear ; i + +)
    {
        printf ("%d", cq[i]) ;
    }
}
if (front > rear)
    {
        for (i = front ; i < MAXSIZE ; i + +)
```

```
            {
                printf ("%d", cq[i]) ;
            }
            for (i = 0 ; i <= rear ; i + +)
            {
                printf ("%d", cq[i]) ;
            }
        }
        printf("\n") ;
    }
```

Output of the program

```
--------1.Insert-------
------- 2. Delete--------
------- 3. Display--------
-------4.exit------------
Enter your choice
      1
Enter the element to be inserted
      5
Do you want to continue (y/n)
      y
--------1.Insert-------
------- 2. Delete--------
------- 3. Display--------
-------4.exit------------
Enter your choice
      1
Enter the element to be inserted
      6
Do you want to continue (y/n)
      y
--------1.Insert-------
------- 2. Delete--------
------- 3. Display--------
-------4.exit------------
Enter your choice
      1
Enter the element to be inserted
      7
Do you want to continue (y/n)
      y
--------1.Insert-------
------- 2. Delete--------
------- 3. Display--------
```