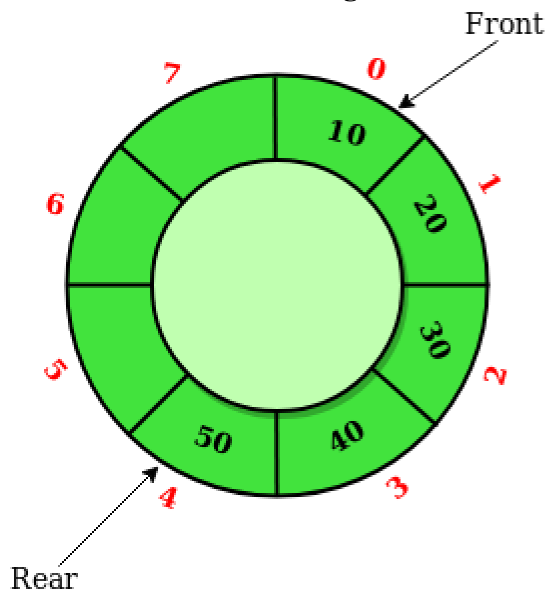
What is Circular Queue?

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **'Ring Buffer'**.



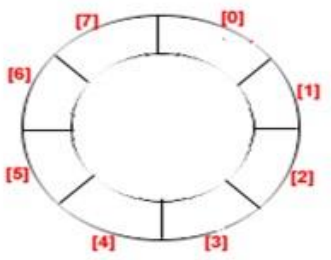
Note that the container of items is an array. Array is stored in main memory. Main memory is linear. So this circularity is only logical. There can not be physical circularity in main memory.

In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.

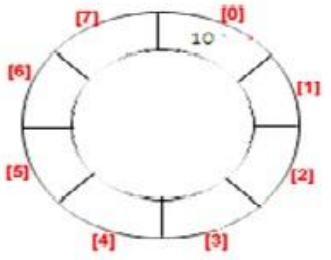
Applications:

1. **Memory Management:** The unused memory locations in the case of ordinary queues can be utilized in circular queues.
2. **Traffic system:** In computer controlled traffic system, circular queues are used to switch on the traffic lights one by one repeatedly as per the time set.
3. **CPU Scheduling:** Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.

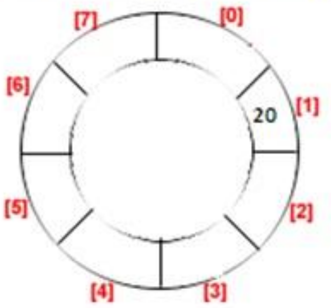
1) Initially: **Front = 0** and **rear = -1**



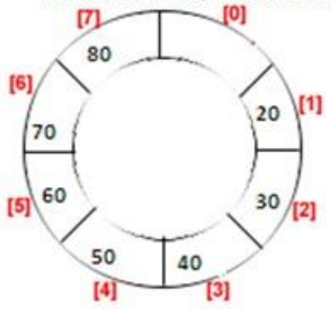
2) Add item 10 then **front = 0** and **rear =0**.



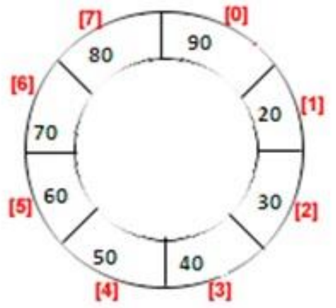
3) Now delete one item then **front = 1** and **rear =1**.

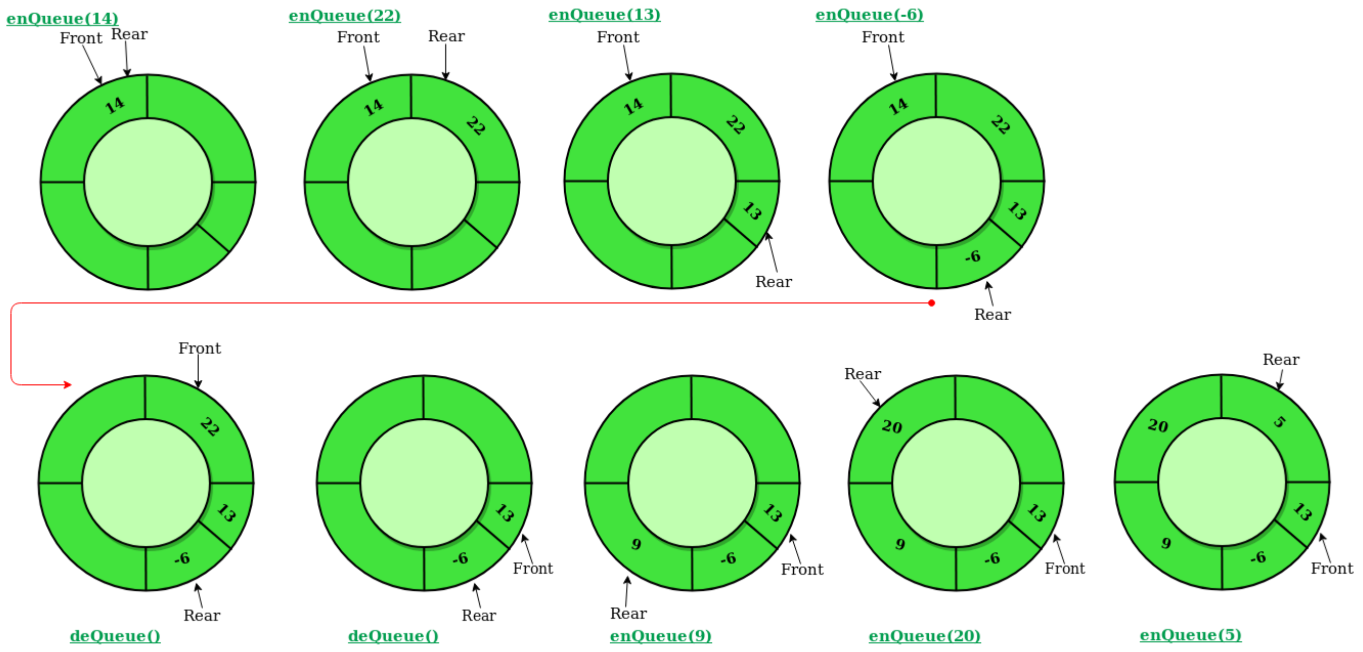


4) Like this now insert 30, 40, and 50,50,70,80 respectability then **front =1** and **rear = 7.**



5) Now in case of linear queue, we can not access 0 block for insertion but in circular queue next item will be inserted of 0 block then **front =0** and **rear = 0.**





See the logical circularity of the queue. Addition causes the increment in REAR. It means that when REAR reaches MAX-1 position then Increment in REAR causes REAR to reach at first position that is 0.

```
if( rear  == MAX -1 )
    rear  = 0;
else
    rear = rear + 1;
```

As we know that, Deletion causes the increment in FRONT. It means that when FRONT reaches the MAX-1 position, then increment in FRONT, causes FRONT to reach at first position that is 0.

```
if( front  == MAX -1 )
    front  = 0;
else
    front = front + 1;
```

Condition to check FULL Circular Queue

```
if( ( front == MAX-1) || ( front ==0 && rear == MAX -1 ) )
```

Condition to check EMPTY Circular Queue

```
if( ( front == MAX-1) || ( front ==0 && rear == MAX -1 ) )
```

Operation of Circular Queue using count

1. Initialize Operation.
2. **Is_Full** check.
3. **Is_Empty** check.
4. Addition or Insertion operation.
5. Deletion operation.

Algorithms:

1. **INIT**(QUEUE,FRONT,REAR,COUNT)
2. **INSERT-ITEM**(QUEUE, FRONT, REAR, MAX, COUNT, ITEM)
3. **REMOVE-ITEM**(QUEUE, FRONT, REAR, COUNT, ITEM)
4. **FULL-CHECK**(QUEUE,FRONT,REAR,MAX,COUNT,FULL)
5. **EMPTY-CHECK**(QUEUE,FRONT,REAR,MAX,COUNT,EMPTY)

INIT(QUEUE,FORNT,REAR,COUNT)

This algorithm is used to initialize circular queue.

```
1. FRONT := 1;
2. REAR  := 0;
3. COUNT :=  0;
4. Return;
```

INSERT-ITEM(QUEUE, FRONT, REAR, MAX, COUNT, ITEM)

This algorithm is used to insert or add item into circular queue.

```
1.  If ( COUNT = MAX ) then
    a. Display  “Queue overflow”;
    b. Return;
2.  Otherwise
    a. If ( REAR = MAX ) then
       i.  REAR := 1;
    b. Otherwise
       i.  REAR := REAR + 1;
    c. QUEUE(REAR) := ITEM;
    d. COUNT := COUNT + 1;
3.  Return;
```

REMOVE-ITEM(QUEUE, FRONT, REAR, COUNT, ITEM)

This algorithm is used to remove or delete item
from circular queue.

```
1.  If ( COUNT = 0 ) then
    a. Display “Queue underflow”;
    b. Return;
2.  Otherwise
    a. ITEM := QUEUE(FRONT)l
    b. If ( FRONT =MAX ) then
       i.  FRONT := 1;
    c. Otherwise
       i.  FRONT := FRONT + 1;
    d. COUNT := COUNT + 1;
3.  Return;
```

EMPTY-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,EMPTY)

This is used to check queue is empty or not.

```
1.  If( COUNT = 0 ) then
    a. EMPTY := true;
2.  Otherwise
    a. EMPTY := false;
```

```
3.  Return ;
```

FULL-CHECK(QUEUE,FRONT,REAR,MAX,COUNT,FULL)
This algorithm is used to check queue is full or not.

```
1.  If ( COUNT = MAX ) then
    a. FULL := true;
2.  Otherwise
    a. FULL := false;
3.  Return ;
```

Task:

1. Implement Circular Queue using Arrays.

2. Implement Circular Queue using linklist.