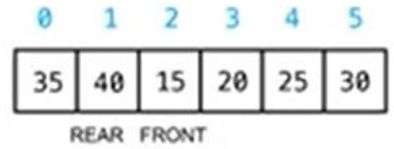
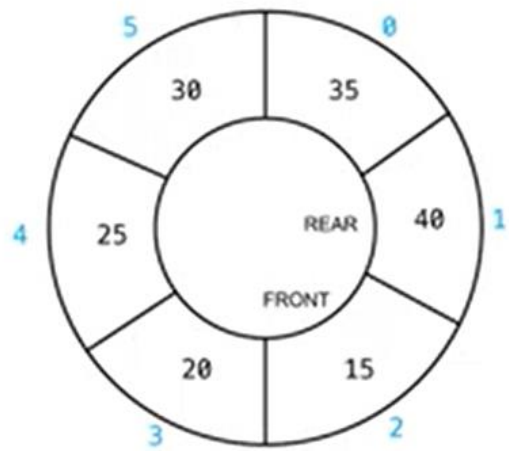
Circular Queue

- A circular queue solved the limitations of the normal queue. Thus making it a better pick than the normal queue. It also follows the first come first serve algorithm. Circular Queue is also called ***RING BUFFER***.

Operations On A Circular Queue

- Enqueue- adding an element in the queue if there is space in the queue.
- Dequeue- Removing elements from a queue if there are any elements in the queue
- Front- get the first item from the queue.
- Rear- get the last item from the queue.
- isEmpty/isFull- checks if the queue is empty or full.






```
Q = circularQueue(6)
Q.Enqueue(5)
Q.Enqueue(10)
Q.Enqueue(15)
Q.Enqueue(20)
Q.Enqueue(25)
Q.Enqueue(30)
Q.Dequeue()
Q.Dequeue()
Q.Enqueue(35)
Q.Enqueue(40)
```

Circular Queue

Applications Of A Circular Queue

- Memory management: circular queue is used in memory management.
- Process Scheduling: A CPU uses a queue to schedule processes.
- Traffic Systems: Queues are also used in traffic systems.

Implementation of Circular Queue

- Using Array
- Using Linked List

Circular Queue using Array

- **Step 1:** Create a one dimensional array with above defined SIZE (**int queue[SIZE]**)

- **Step 2:** Define two integer variables **'front'** and **'rear'** and initialize both with **'-1'**. (**int front = -1, rear = -1)**

```
#define SIZE 5

int cqueue[SIZE],

Int front=-1,rear=-1;
```

Enqueue

enQueue(value) - Inserting value

Step 1: Check whether **queue** is **FULL**.

Step 2: If it is **FULL**, then display **"Queue is FULL!!!** and terminate

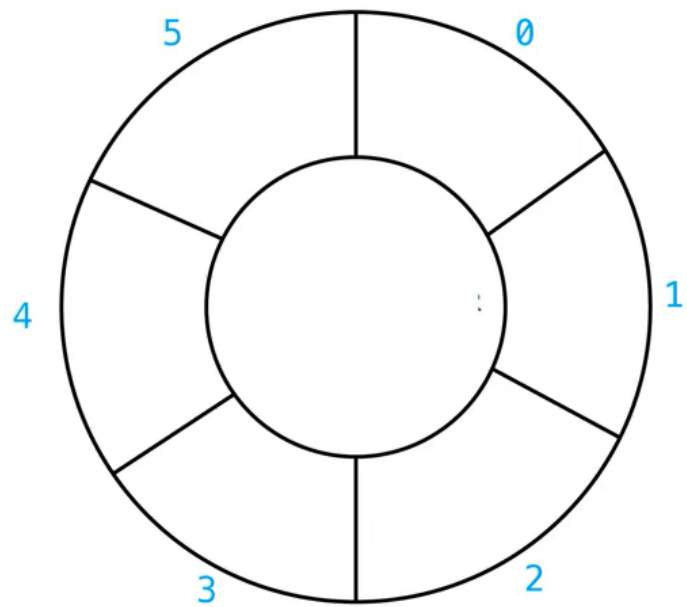
Step 3: If it is **EMPTY,** then place the first item in position **front = rear = 0**

Step 4 : if NOT FULL,NOT EMPTY , then increment **rear** value by one (**rear++**) or reset rear value to 0 and set **queue[rear]** = **value**.

Enqueue – Queue is Empty

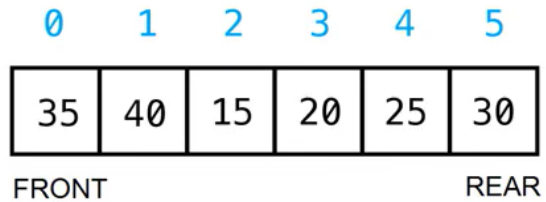
```
void insert(int item)
{
 if((front == 0 && rear == MAX-1) || (front == rear+1))
 {
  printf("Queue FULL n");
  return;
 }
 if(front == -1)
 {
  front = 0;
  rear = 0;
 }
 else
 {
  if(rear == MAX-1)
    rear = 0;
  else
    rear = rear+1;
 }
 cqueue[rear] = item ;
}
```

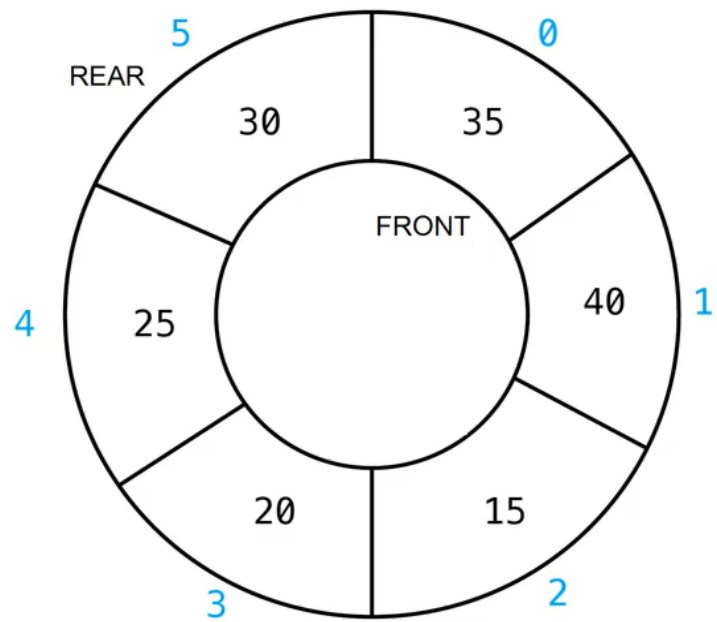
Front = Rear = -1 means "Circular Queue is empty"



Enqueue – Queue is FULL

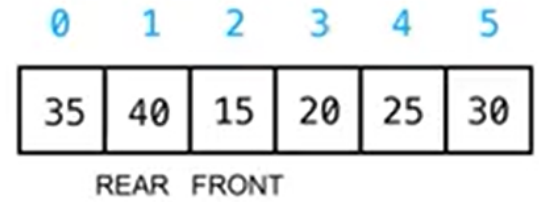
```
void insert(int item)
{
 if((front == 0 && rear == MAX-1) || (front == rear+1))
 {
  printf("Queue FULL n");
  return;
 }
 if(front == -1)
 {
  front = 0;
  rear = 0;
 }
 else
 {
  if(rear == MAX-1)
    rear = 0;
  else
    rear = rear+1;
 }
 cqueue[rear] = item ;
}
```

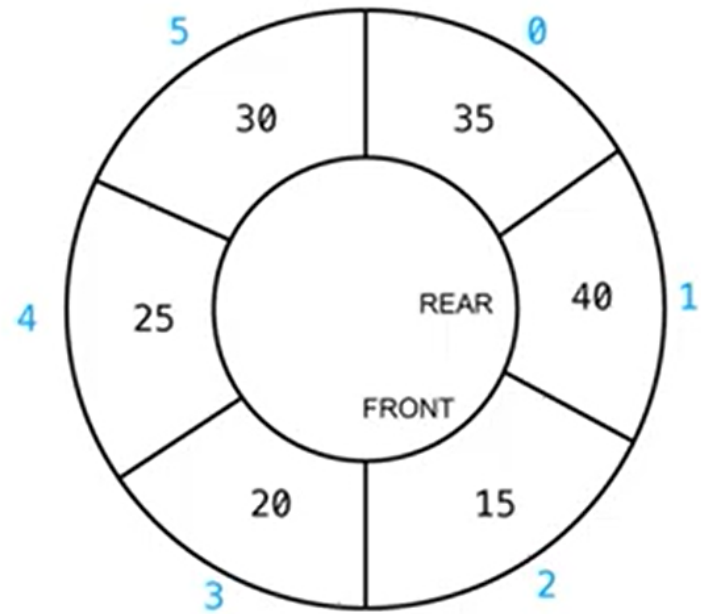




Enqueue – Queue is FULL

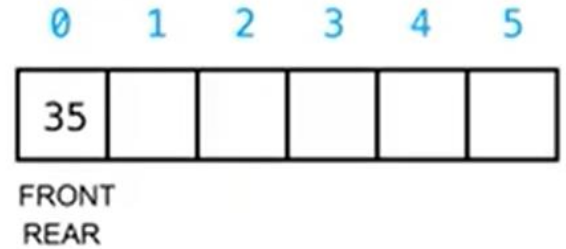
```
void insert(int item)
{
 if((front == 0 && rear == MAX-1) || (front == rear+1))
 {
  printf("Queue FULL n");
  return;
 }
 if(front == -1)
 {
  front = 0;
  rear = 0;
 }
 else
 {
  if(rear == MAX-1)
    rear = 0;
  else
    rear = rear+1;
 }
 cqueue[rear] = item ;
}
```

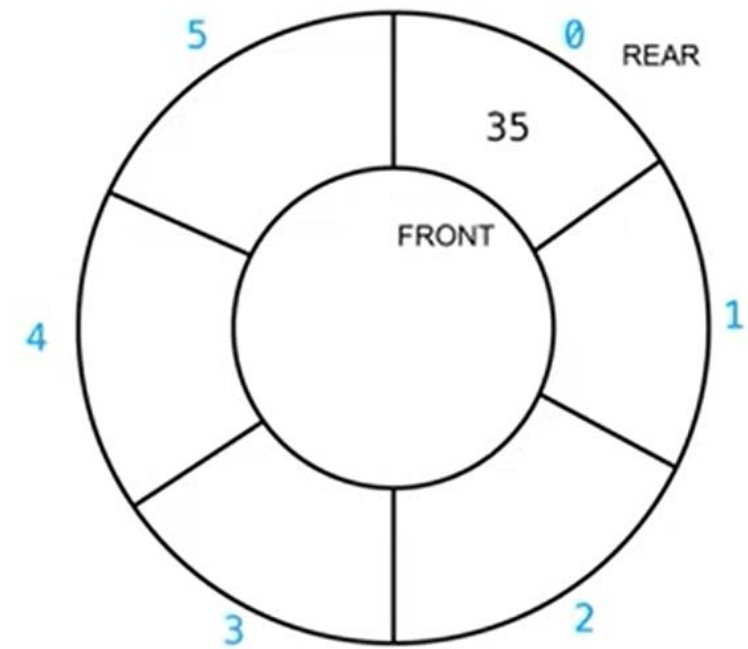




Enqueue

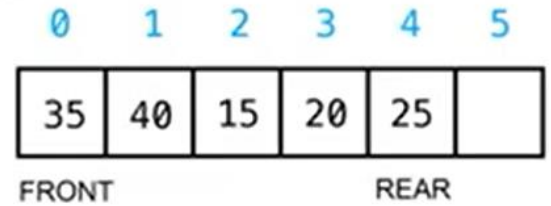
```
void insert(int item)
{
 if((front == 0 && rear == MAX-1) || (front == rear+1))
 {
  printf("Queue FULL n");
  return;
 }
 if(front == -1)
 {
  front = 0;
  rear = 0;
 }
 else
 {
  if(rear == MAX-1)
    rear = 0;
  else
    rear = rear+1;
 }
 cqueue[rear] = item ;
}
```

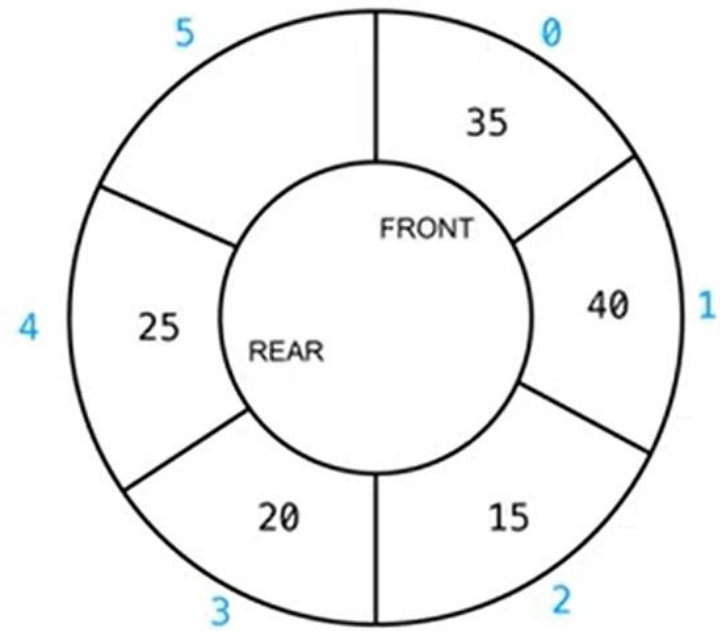




Enqueue

```
void insert(int item)
{
 if((front == 0 && rear == MAX-1) || (front == rear+1))
 {
  printf("Queue FULL n");
  return;
 }
 if(front == -1)
 {
  front = 0;
  rear = 0;
 }
 else
 {
  if(rear == MAX-1)
    rear = 0;
  else
    rear = rear+1;
 }
 cqueue[rear] = item ;
}
```

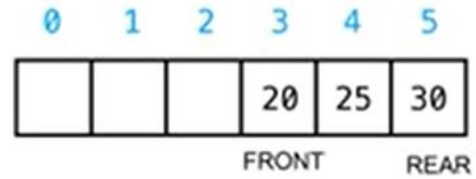


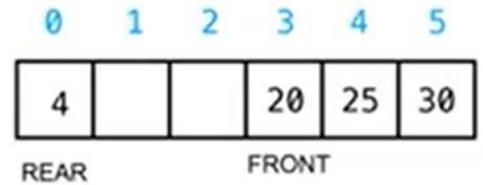


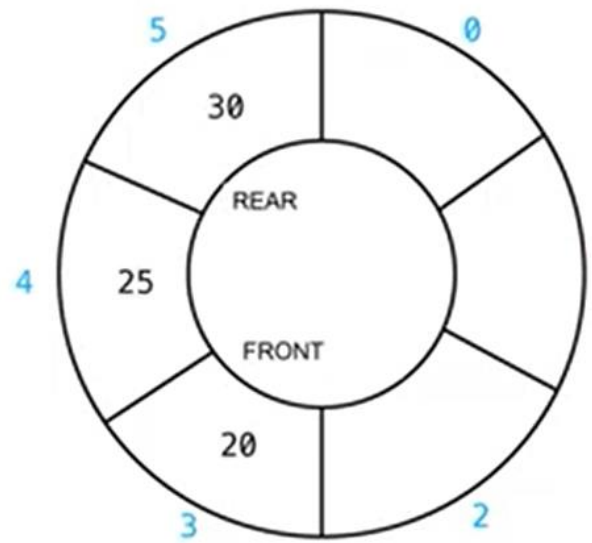
Enqueue

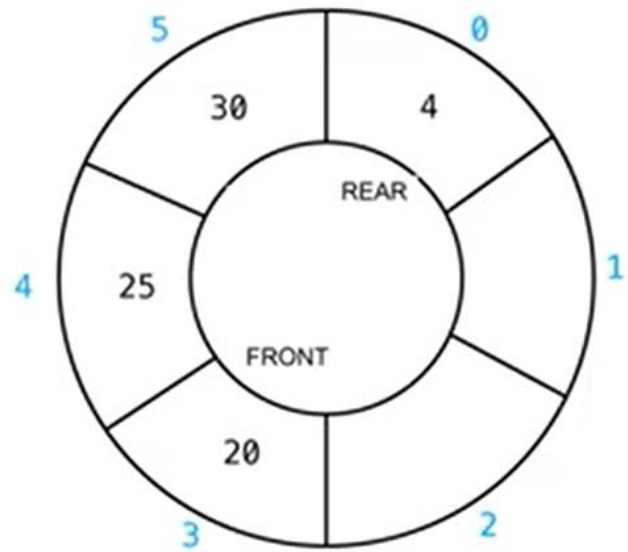
- **Suppose Three elements (35,45,15) are deleted from the Circular Queue. Then Front will be in 4 position i.e. is in element 20. If we try to insert, rear will move the 0th position since it is circular queue**

```
void insert(int item)
{
 if((front == 0 && rear == MAX-1) || (front == rear+1))
 {
  printf("Queue FULL n");
  return;
 }
 if(front == -1)
 {
  front = 0;
  rear = 0;
 }
 else
 {
  if(rear == MAX-1)
    rear = 0;
  else
    rear = rear+1;
 }
 cqueue[rear] = item ;
}
```



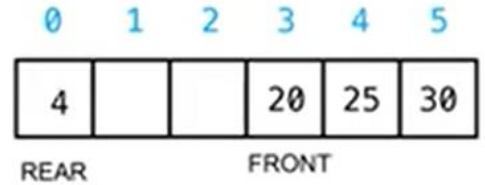


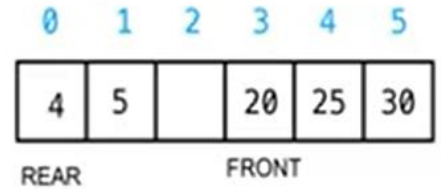


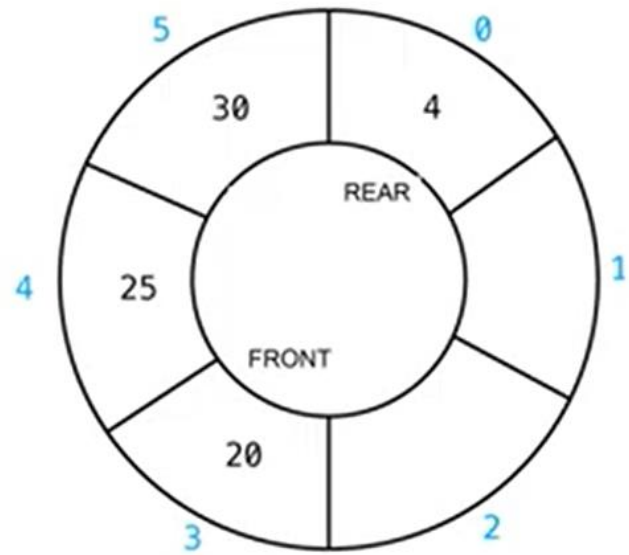


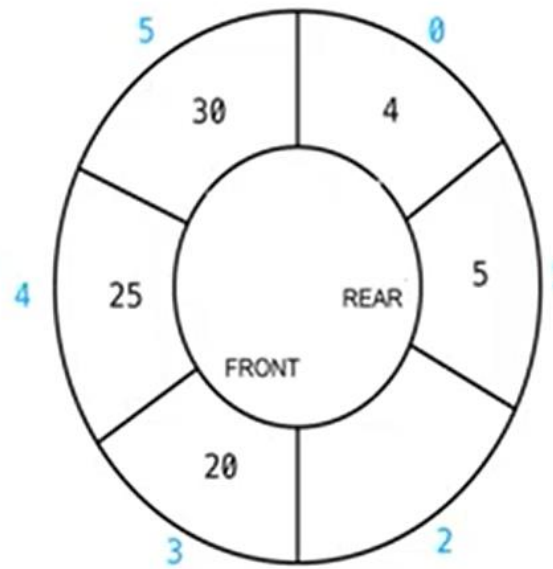
Enqueue

```
void insert(int item)
{
 if((front == 0 && rear == MAX-1) || (front == rear+1))
 {
  printf("Queue FULL n");
  return;
 }
 if(front == -1)
 {
  front = 0;
  rear = 0;
 }
 else
 {
  if(rear == MAX-1)
    rear = 0;
  else
    rear = rear+1;
 }
 cqueue[rear] = item ;
}
```









Dequeue

dequeue() – Removing value

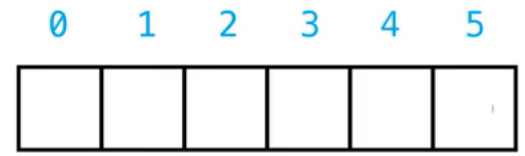
Step 1: Check whether **queue** is **EMPTY**. (**front == -1**)

Step 2: If it is **EMPTY**, then display **"Queue is EMPTY!!"** and terminate.

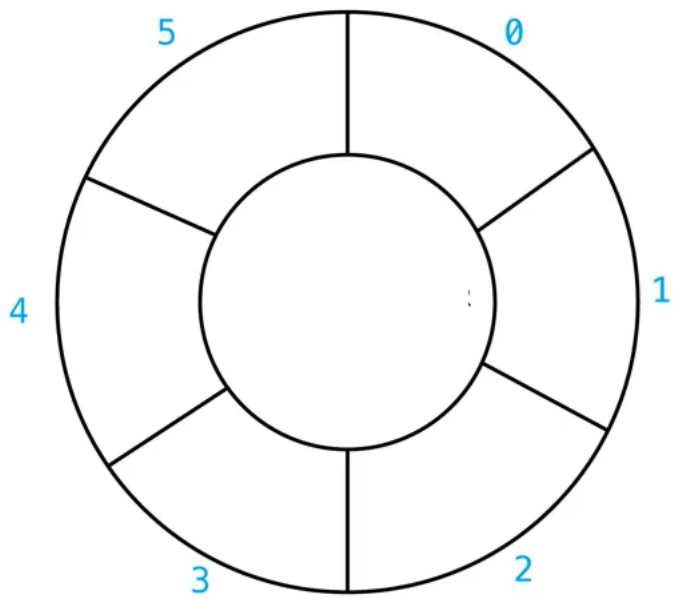
Step 3: If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check if both **front** and **rear** are equal (**front** == **rear**), then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**) . Else If **front == max-1** then **set front=0**

Dequeue

```
void deletion()
{
 if(front == -1)
 {
   printf("Queue EMPTY");
   return ;
 }
 printf("Element deleted from
      queue is : %dn",cqueue[front]);
 if(front == rear)
 {
   front = -1;
   rear=-1;
 }
 else
 {
   if(front == MAX-1)
     front = 0;
   else
     front = front+1;
 }
}
```

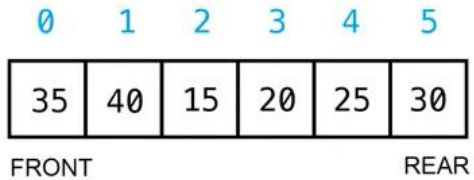


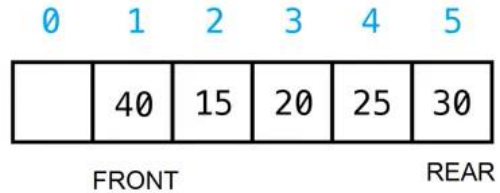
Front = Rear = -1 means "Circular Queue is empty"

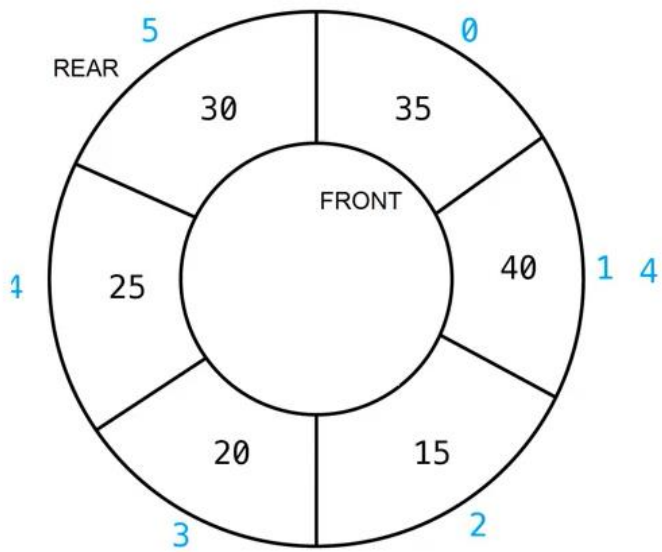


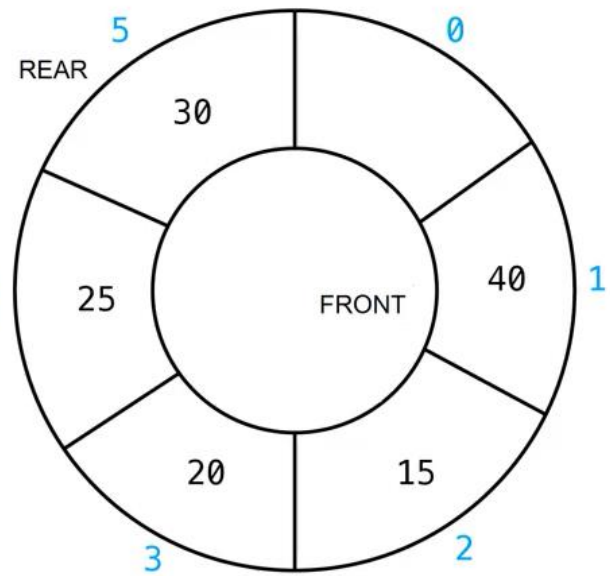
Dequeue

```
void deletion()
{
 if(front == -1)
 {
   printf("Queue EMPTY");
   return ;
 }
 printf("Elt deleted is :
 %dn",cqueue[front]);
 if(front == rear)
 {
   front = -1;
   rear=-1;
 }
 else
 {
   if(front == MAX-1)
     front = 0;
   else
     front = front+1;
 }
}
```



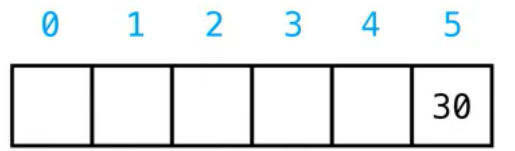


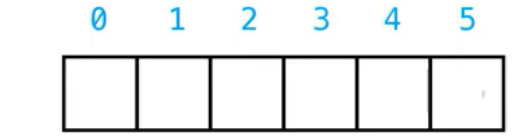




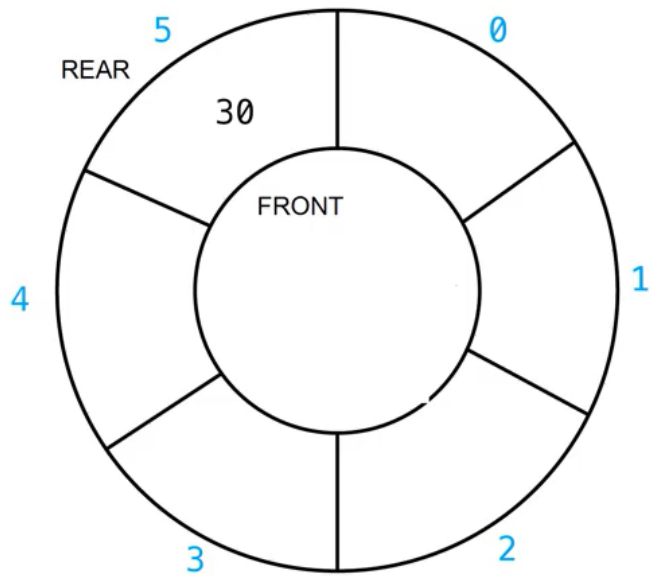
Dequeue

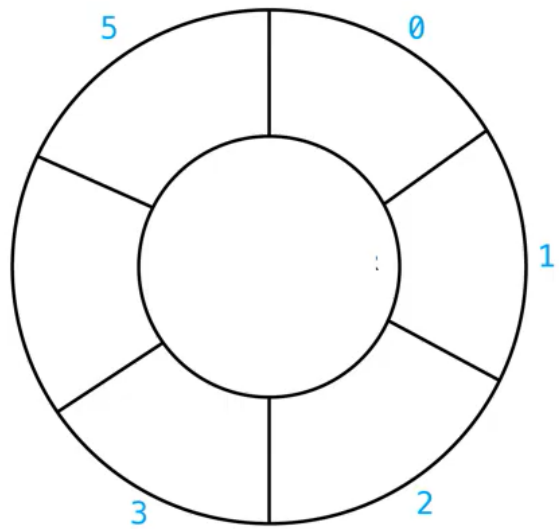
```
void deletion()
{
 if(front == -1)
 {
   printf("Queue EMPTY");
   return ;
 }
 printf("Elt deleted is :
 %dn",cqueue[front]);
 if(front == rear)
 {
   front = -1;
   rear=-1;
 }
 else
 {
   if(front == MAX-1)
     front = 0;
   else
     front = front+1;
 }
}
```





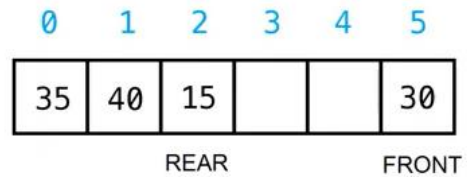
Front = Rear = -1 means "Circular Queue is empty"

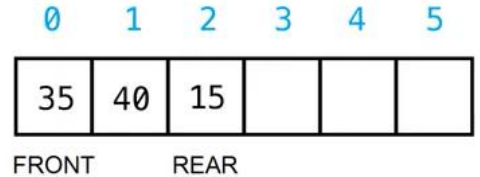


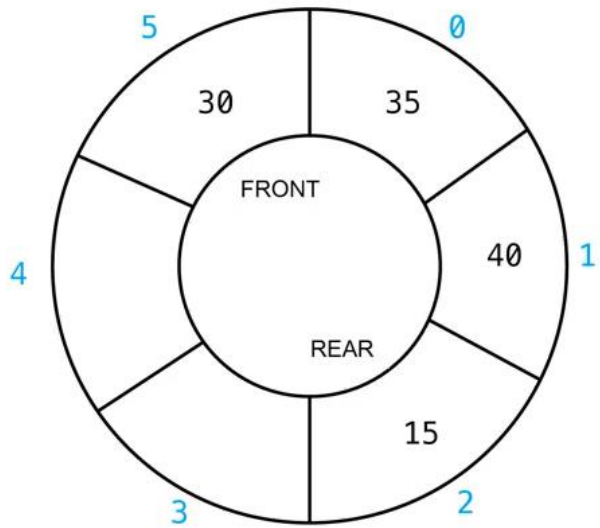


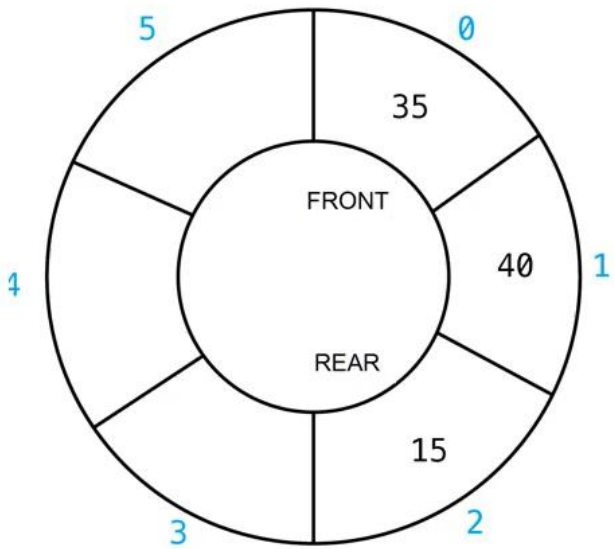
Dequeue

```
void deletion()
{
 if(front == -1)
 {
   printf("Queue EMPTY");
   return ;
 }
 printf("Elt deleted is :
 %dn",cqueue[front]);
 if(front == rear)
 {
   front = -1;
   rear=-1;
 }
 else
 {
   if(front == MAX-1)
     front = 0;
   else
     front = front+1;
 }
}
```









Display

display() - Displays the elements of a Queue

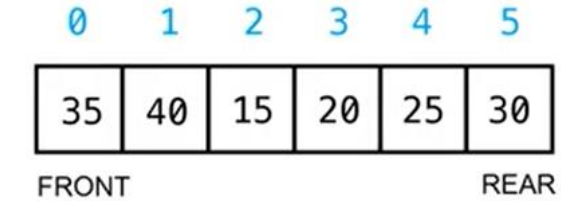
1: Check whether **queue** is **EMPTY**. (**front == -1**)

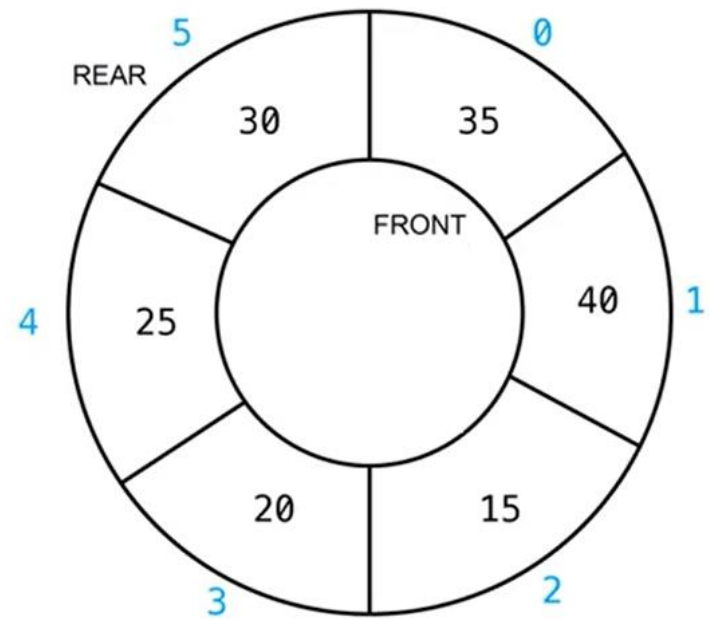
2a: If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate.

2b: If it is **NOT EMPTY**, print the element from front to rear if front < rear, else Print from front to max-1 and 0 to rear.

Display

```
void display()
{
 int f;
 if(front == -1)
 {
    printf("Queue is emptyn");
    return;
 }
 else
 {
  printf("Queue elements :n");
  if( front<= rear )
   for(int f=front; f <= rear; f++)
        printf("%d ",cqueue[f]);
 }
 else
 {
    for(f=front; f <= MAX-1; f++)
        printf("%d ",cqueue[f])
    for(f=0;f <= rear; f++)
        printf("%d ",cqueue[f]);
 }
}
```

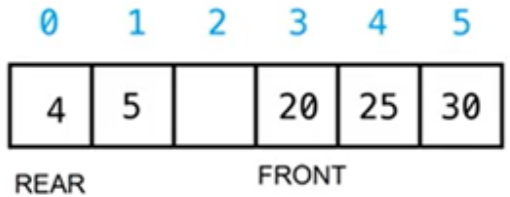


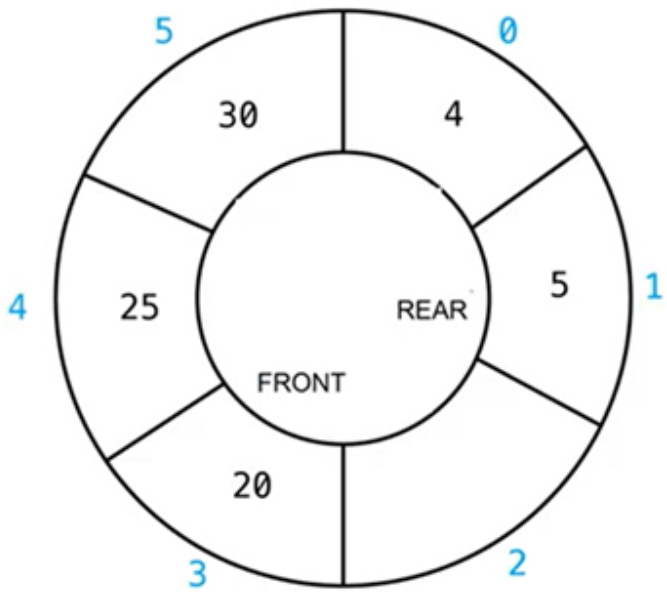


Element are 35, 40, 15, 20, 25, 30

Display

```
void display()
{
 int f;
 if(front == -1)
 {
    printf("Queue is emptyn");
    return;
 }
 else
 {
  printf("Queue elements :n");
  if( front<= rear )
   for(int f=front; f <= rear; f++)
        printf("%d ",cqueue[front_pos]);
 }
 else
 {
    for(f=front; f <= MAX-1; f++)
        printf("%d ",cqueue[f])
    for(f=0;f <= rear; f++)
        printf("%d ",cqueue[f]);
 }
}
```





Element are 20, 25, 30, 4, 5