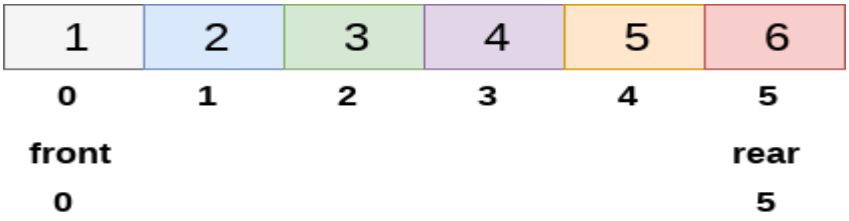
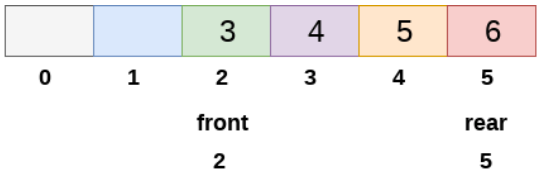
INTRODUCTION TO CIRCULAR QUEUE

In a linear queue insertion and deletion operations can only be performed at rear and front end of the queue.



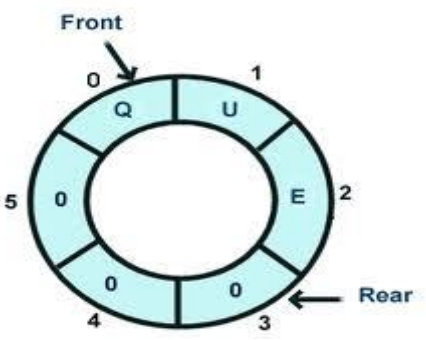
Queue

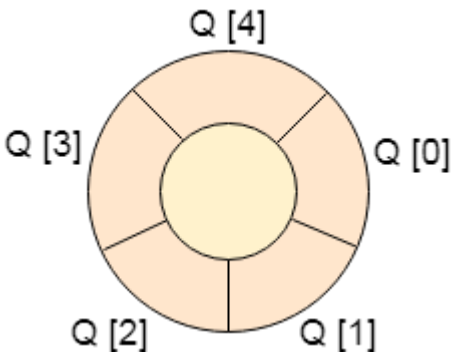
- ➔ The Queue shown in above figure is completely filled and there can't be inserted any more element due to the condition **rear == max - 1 becomes true**.
- ➔ However, if we delete 2 elements at the front end of the queue, we still cannot insert any element since the condition **rear = max -1 still holds**.
- ➔ This is the main problem with the linear queue, although we have space available in the array, but we cannot insert any more element in the queue. This is simply the memory wastage and we need to overcome this problem.
- ➔ We can shift the elements towards left to create empty space at rear end, but when queue size very large , shifting operation takes more execution time which is expensive.



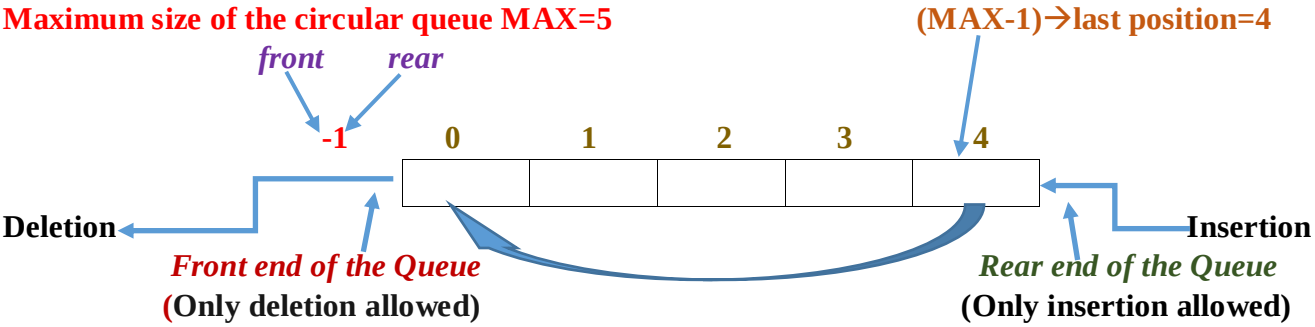
Queue after deletion of first 2 elements

One of the solution of this problem is circular queue. In the circular queue, the first index comes right after the last index. You can think of a circular queue as shown in the following figure.





/*Fig: 1 An empty Circular Queue */



Underflow or empty condition in circular Q:

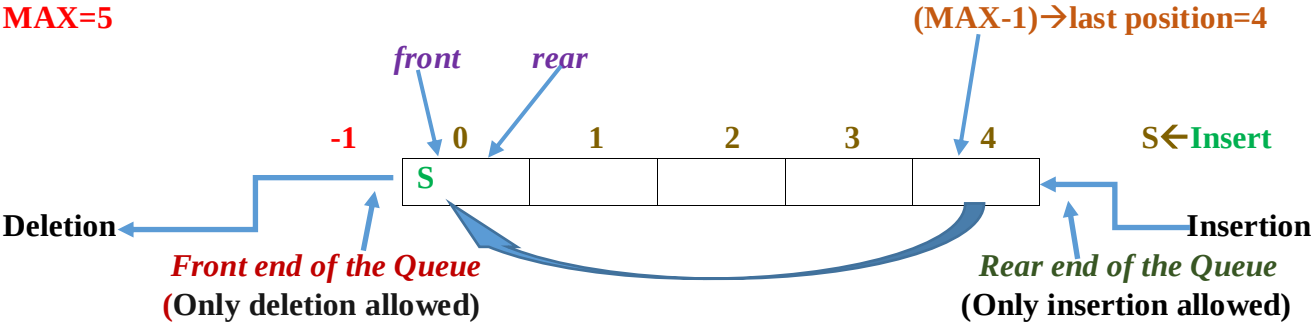
→From the above figure you can easily say that a circular queue is empty when either rear < 0 or front < 0.

→Therefore you can use any one condition to check underflow or empty condition in circular Queue.

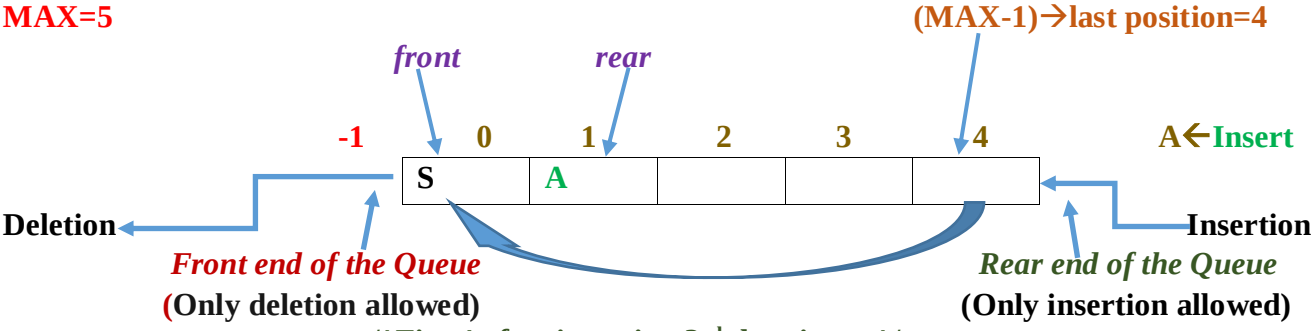
```
If (front < 0)
   System.out.println("the circular Q is empty");
  Or
If (rear < 0)
   System.out.println("the circular Q is empty");
```

Let's perform some insertion operations on the above empty circular queue:

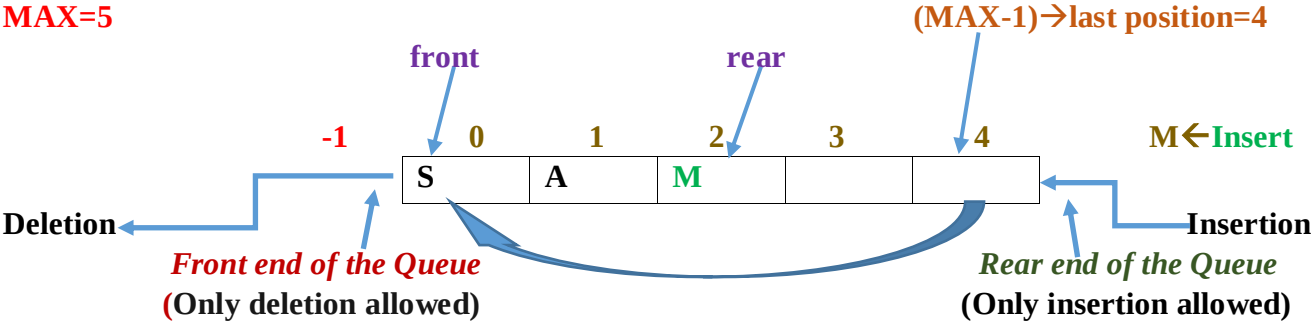
/*Fig: 2 After insertion of the 1st item onto the above empty Circular Queue */



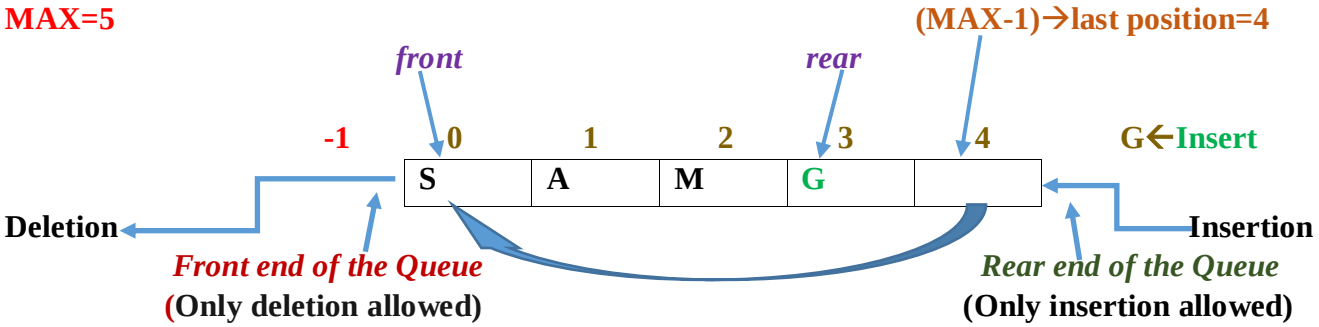
/*Fig: 3 After insertion of the 2nd item onto the above non-empty Circular Queue */



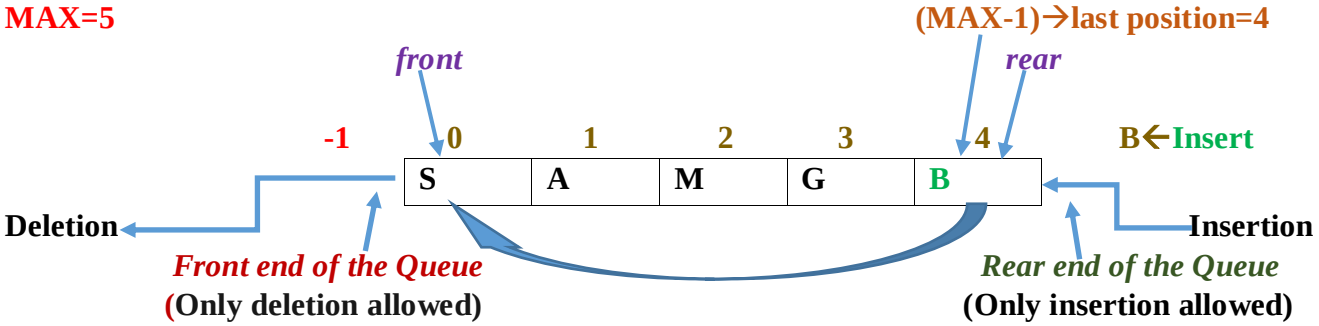
/*Fig: 4 after inserting 3rd data item */



/*Fig: 5 After insertion of the 4th item into the above non-empty Circular Queue */



/*Fig: 6 After insertion of the 5th item onto the above non-empty Circular Queue */

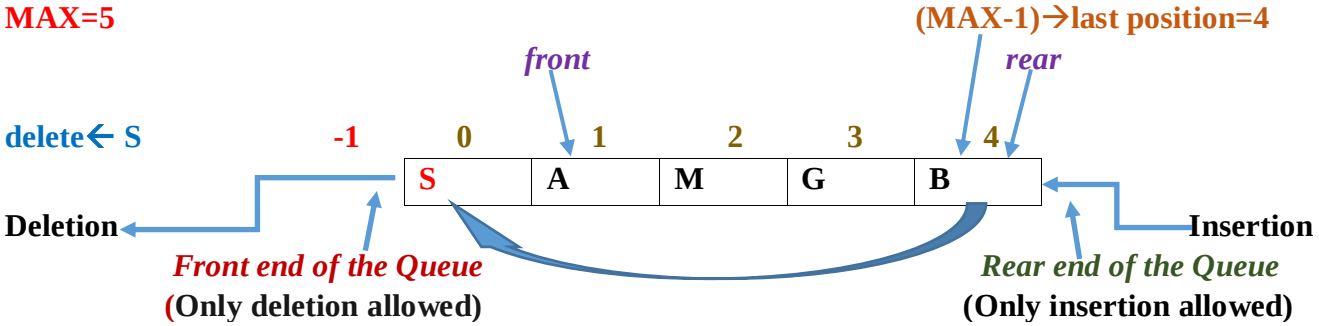


Now you can see all the positions in the above Q are occupied, after this we can't insert more items into above Q, therefore this one condition when a circular Q becomes overflow or full : so in java you can write

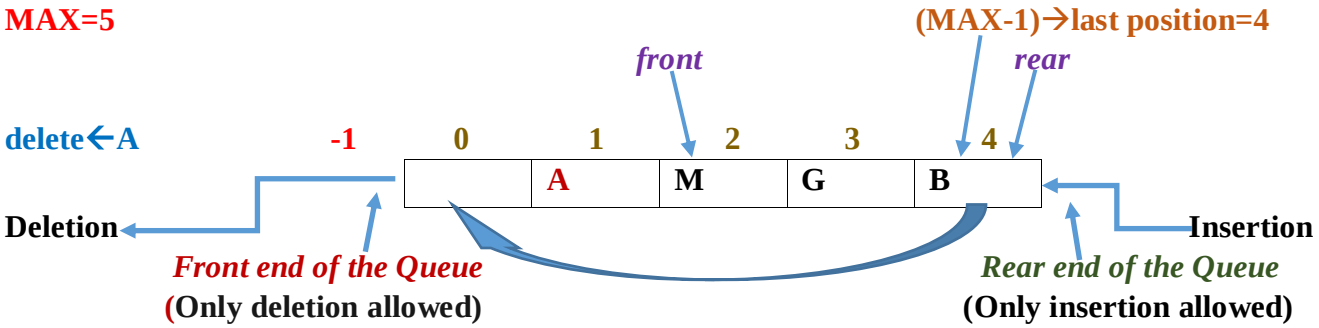
```
If( front=0 && rear = MAX-1)
    System.out.println("Circular Q is full");
```

Now let's perform some deletion operation:

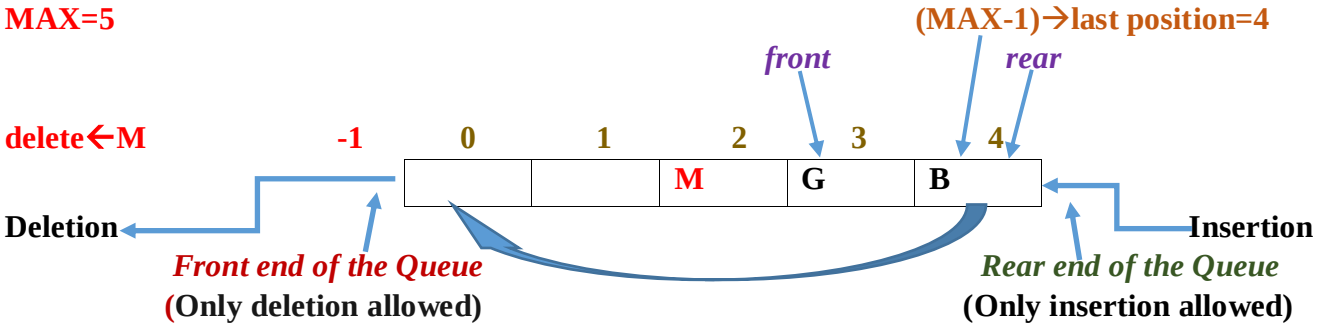
/*Fig: 7 After deletion of 1st item from the above non-empty Circular Queue */



/*Fig: 8 After deletion of 2nd item from the above non-empty Circular Queue */



/*Fig: 9 After deletion of 3rd item from the above non-empty Circular Queue */



Now let's again perform some insertion operations into the above Q:

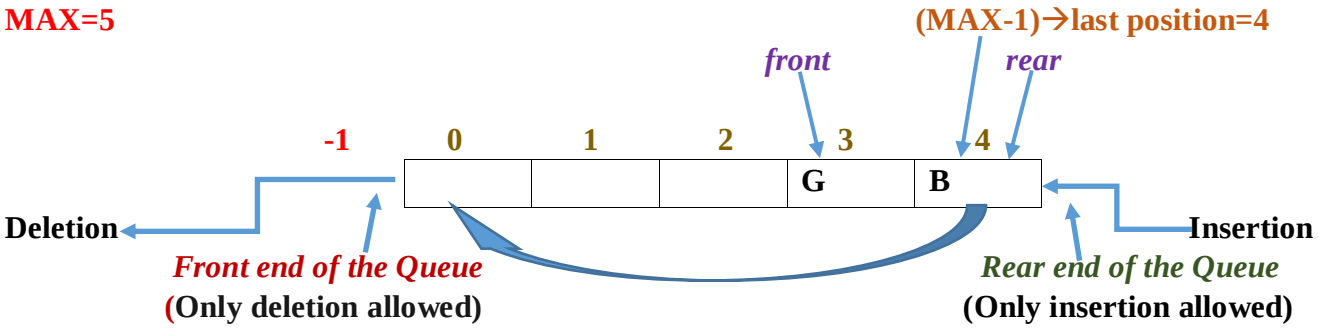
→You can see in fig: 9 the rear pointer refers to the last index (MAX-1) of the circular Q.

→If it would be a linear Q then you can't insert more items after insertion of data item 'B'.

→But as it is a circular Q so you can insert more items because in circular Q, we can move again to the 0^{th} position after the last (MAX-1) position of the Q.

→0^{th} position is also empty, therefore you can insert another item at 0^{th} position after last (MAX-1) position is occupied i.e. after 'rear' pointer refers to last position.

/*Fig: 10 A non-empty Circular Queue */



→So, if you increment 'rear' pointer by writing rear=rear+1, then it would be wrong, because after 4^{th} position there are no more empty positions to the right.

→Therefore, to go to the 0^{th} position again which is empty you should use the following expression to update 'rear' pointer in circular Q before inserting a new item:

$$rear = (rear+1)\%MAX \ ;$$

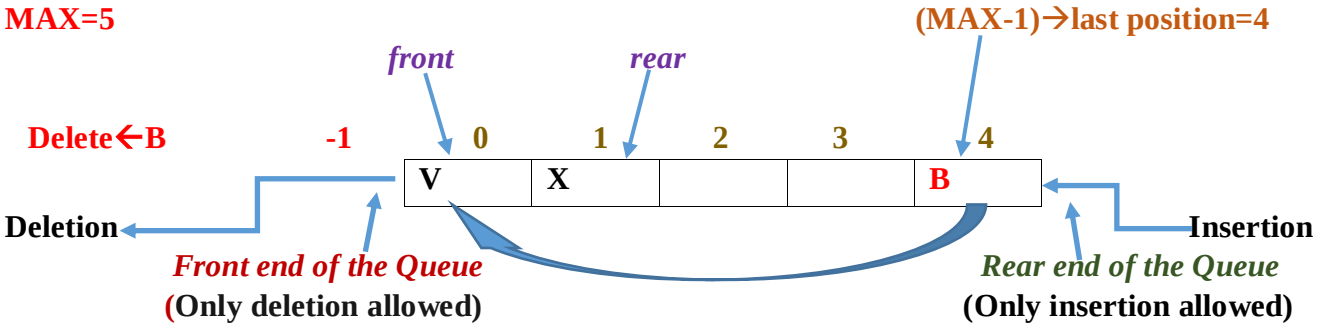
if current value of 'rear' = 4, then value of the above right hand side expression will be 0 which is assigned 'rear' again. So new value of 'rear' will be 0 where you can insert the next data item.

Similarly for deletion operations: update front pointer by using the following expression as shown in fig:

$$front = (front+1)\%MAX \ ;$$

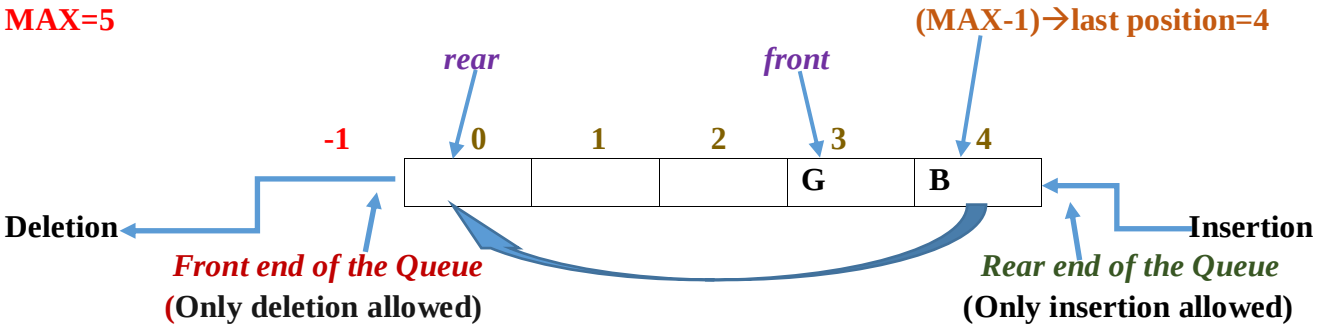
In the following figure: after deletion of the front most element i.e. B, 'front' pointer will move to 0^{th} position again, so instead of using front = front + 1, we should use the above expression for updating 'front' pointer after each deletion operation in circular Q.

/*'front' refers to last position i.e. the current front element of the Q is at last index */

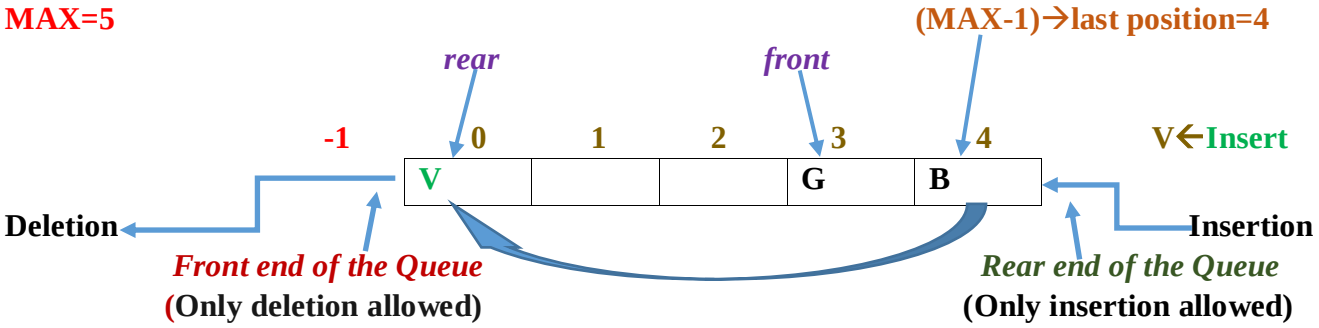


Now again perform some insertion operations by using the expression: rear = (rear+1)%MAX to update the 'rear' pointer:

/*Fig: 11 after updating the 'rear' pointer to the next empty position */

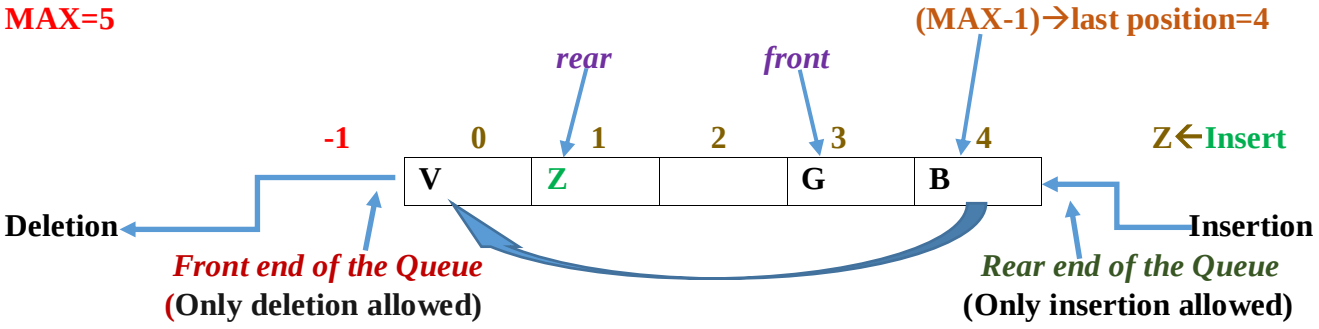


/*Fig: 12 after inserting the new item at updated position of the 'rear' pointer */

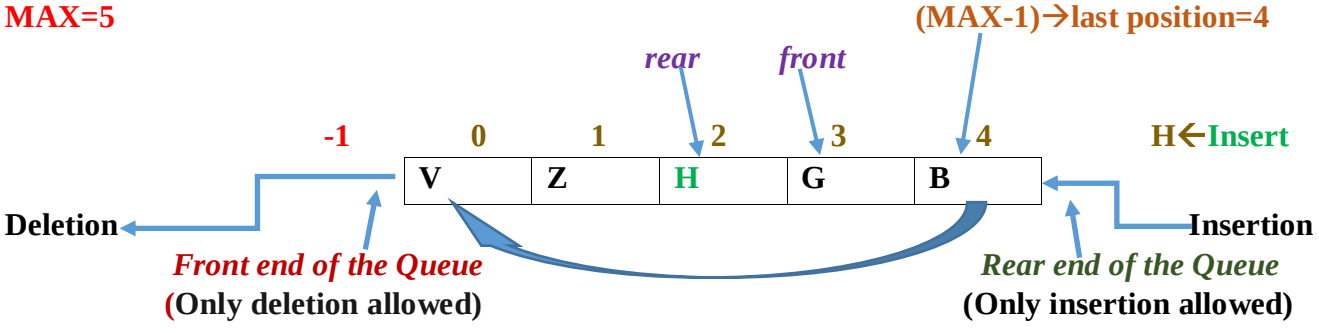


Again insert two more items:

/*Fig: 13 after inserting another new item at updated position of the 'rear' pointer */



/*Fig: 14 after inserting another new item at updated position of the 'rear' pointer */



Now you can see, all the position of the Q are occupied, hence you can't insert more items. Here rear<=front, in this condition the expression that is to be used to check overflow condition of the circular Q is:

```
If( (rear+1) <= front )
    System.out.println("Q is full");
```

So, we got two conditions when a circular is full or overflow i.e.

1. **If (front=0 && rear=MAX-1) is true , when front <=rear**
2. **If((rear+1) <=front) is true , when rear <= front**

But instead of using the above two conditions by writing separate if…statements for each, we can use only one equivalent condition to check overflow in any one of the above conditions of a circular Q, which is as follows:

front == (rear+1)%MAX

Where, MAX is total size of the circular Q

Overflow condition in circular Q:

1. When **front == 0 && rear == max-1**

   Or

2. **front == rear + 1**

Condition 1 & 2 equivalent to

front == (rear + 1) % MAX

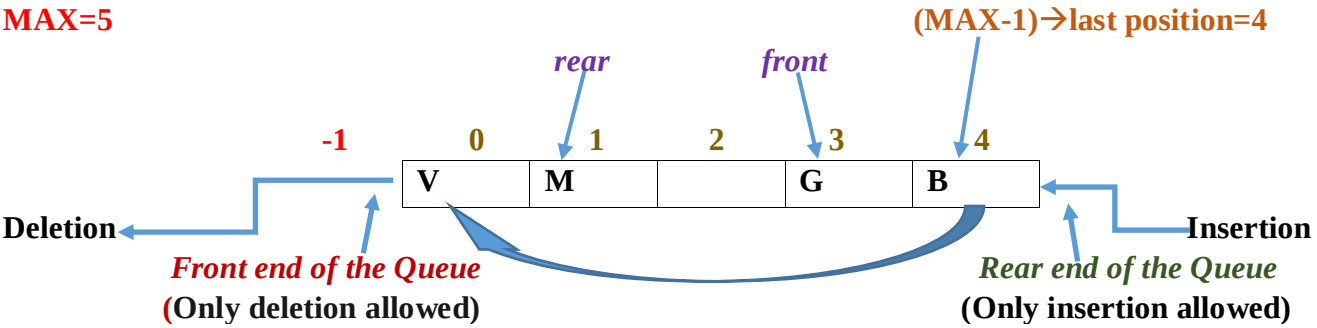
Where **MAX** is maximum size of the queue

→Implementation of circular queue is similar to that of a linear queue.
→Only the logic part that is implemented in the case of insertion and deletion is different from that in a linear queue.
→Like linear queue, in circular queue also, the following operations are performed:

1. **En-Queue:** Insertion of a new data item at rear-end.
2. **De-Queue:** Deletion of the front most data item at front-end.
3. **Traverse:** Visiting each and every element from front element to Rear-most element.

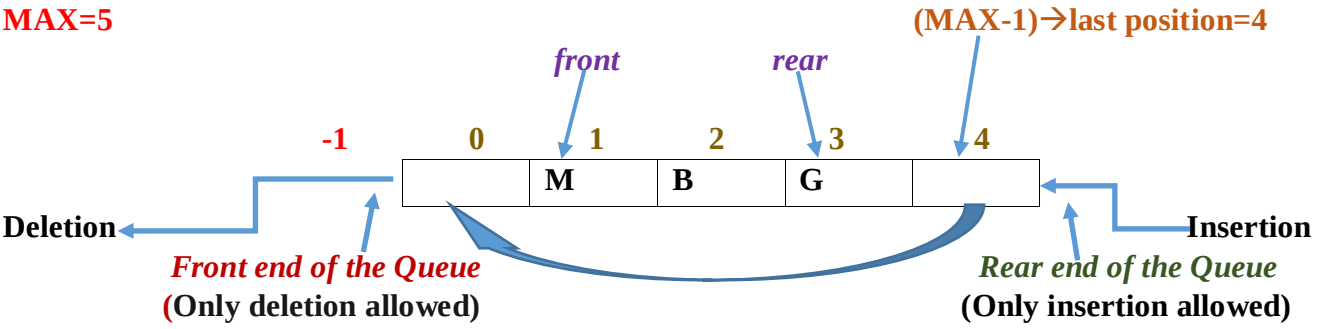
En-Queue operation on circular Q:

/*Fig: 15 A non-empty circular Queue */



In the above fig: 15 if you want to insert a new item then the next empty position after rear is 2, when the rear pointer is behind front pointer and in the fig: 16 the next empty position after rear is 4 :

/*Fig: 16 A non-empty circular Queue */



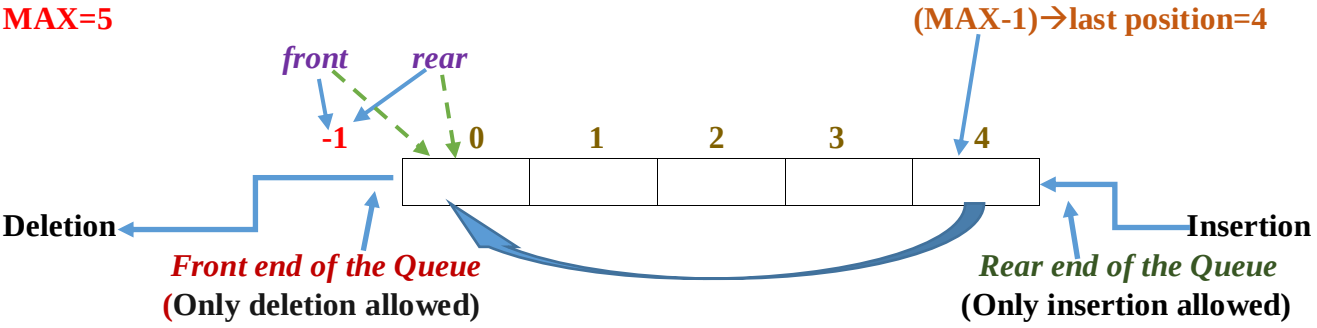
So, for both the above cases you can use only one expression to update the rear pointer to the next empty position i.e. rear = (rear+1)%MAX.

Steps for insertion operation in a circular Q

Step-1→check if the Q is full? If yes then stop else go to step 2

Step-2→if the Q is not full but empty as shown in fig: 17 then, set both 'rear' & 'front' pointer to 0^{th} position of the Q where the new item will be inserted: i.e. rear=front=0.

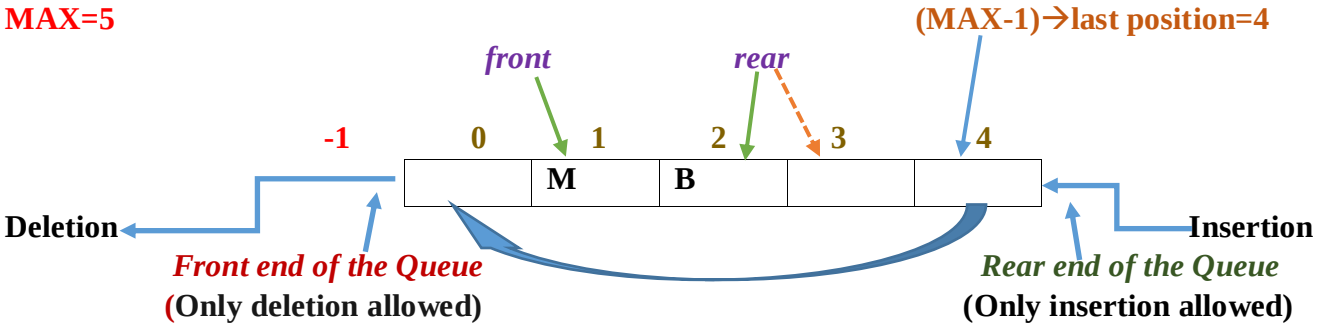
/*Fig: 17 When the circular Q is empty, before insertion of a new item */



/*in fig: 17 the new position of 'rear' & 'front' is shown by dotted arrow i.e 0^{th} position where the new item will be inserted */

Step-3→if the Q contains some elements then only update the rear pointer by using the expression rear = (rear+1)%MAX as shown in fig: 18:

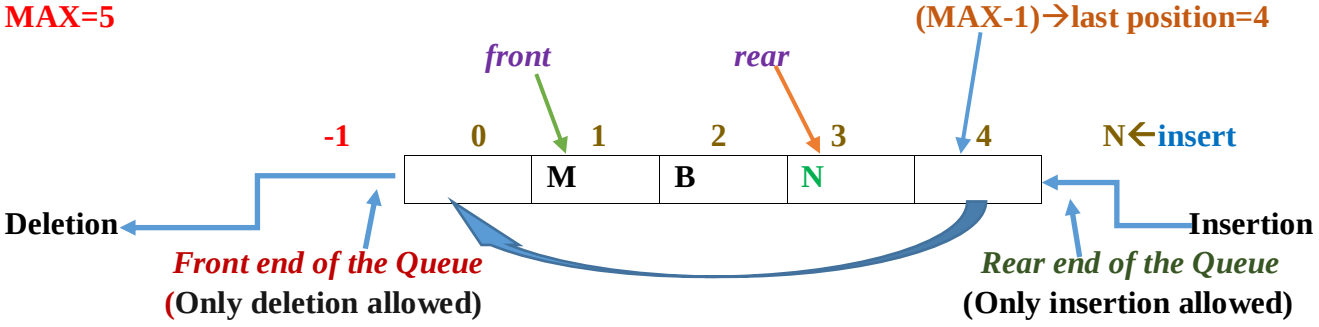
/*Fig: 18 after updating rear pointer only to the next empty position when Q is not empty */



/*The new position of rear pointer i.e. 3rd position is shown by dotted arrow*/

Step-4→finally after updating rear pointer store the new item at new position of rear pointer: Q[rear]=new_item.

/*Fig: 19 after updating rear pointer only to the next empty position when Q is not empty */



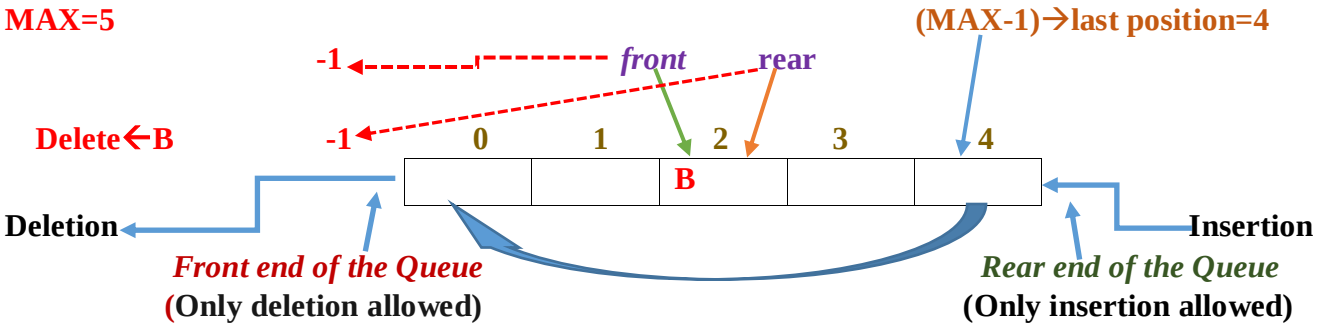
/*after insertion of new item N at new position of 'rear' pointer as shown by dotted arrow*/

De-Queue operation on circular Q:

Step-1→check if the Q is empty? If yes then stop else store the front most item to which front pointer is referring in a temporary variable : temp=Q[front] and go to step 2.

Step-2→check if the Q has only one element as shown in fig: 20? If yes then set both the 'rear' & 'front' pointer to -1 i.e. front=rear= -1 , because after deletion of only item the Q will be empty.

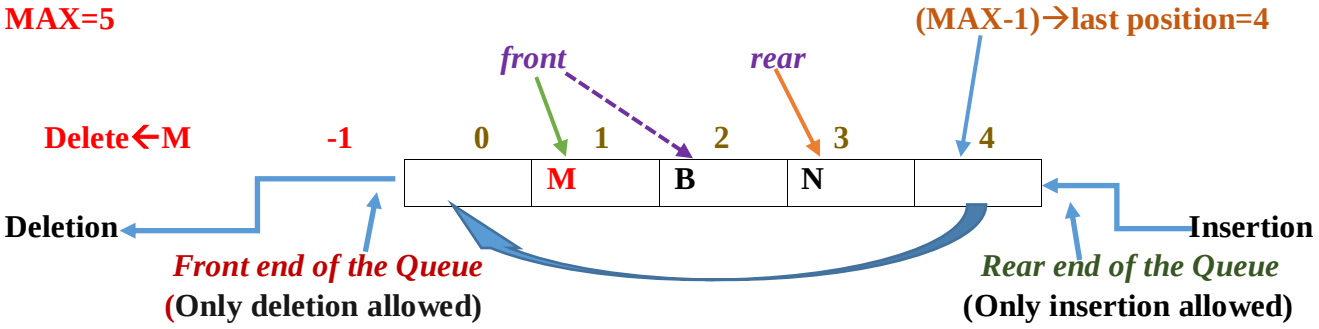
/*Fig: 20 When the Q has only one element that is to be deleted */



/*after setting the 'rear' & 'front' to -1 as show by dotted arrow*/

Step-3→if the Q has more than one element then only update the front pointer by using the expression: front = (front+1)%MAX as shown in fig: 21:

/*Fig: 21 after updating rear pointer only to the next empty position when Q is not empty */



/*after deleting the front most item M, the new position of front pointer as shown by dotted arrow*/

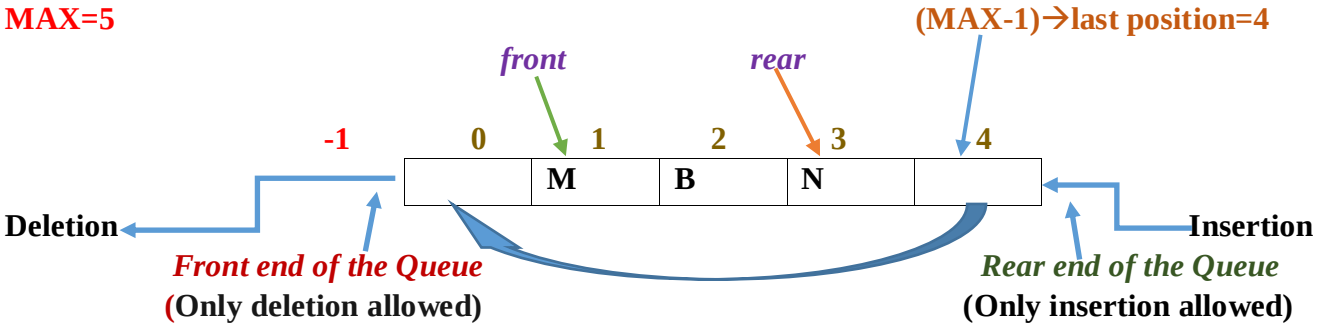
Step-4→finally print the deleted item is temp.

Traverse_Q operation in circular Q:

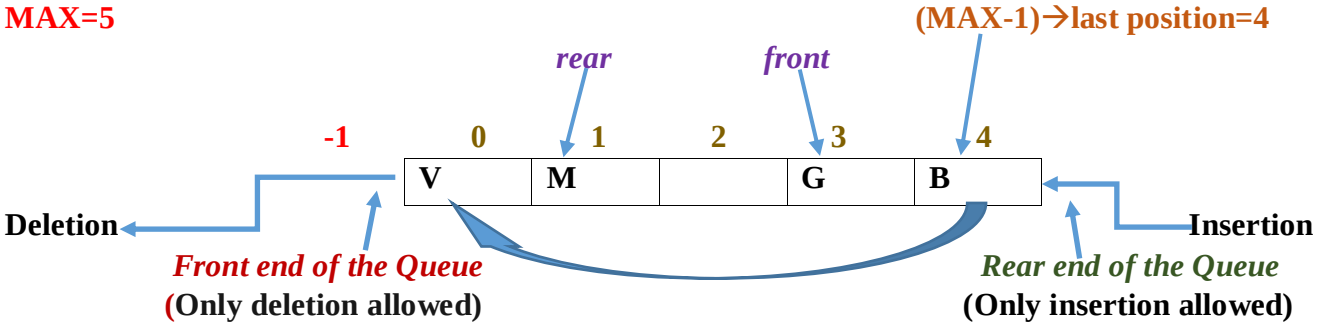
Step-1→check if the Q is empty? If yes then stop else go to step-2

Step-2→check if the front is less than equal to rear pointer i.e. front most element is behind the rearmost element? if yes then visit each element from front pointers position to rear pointers position as shown in fig: 22

/*Fig: 22 A circular Q when front element is behind the rear most element*/



/*Fig: 23 A non-empty circular Queue when rear element is behind front element */



Step-3→if rear most element is behind the front most element i.e. rear<=front as shown in fig: 21, then at first visit each element from 'front' pointers position up to the element at last (MAX-1) position and again visit from 0^{th} position to 'rear' pointers position.