INTRODUCTION TO QUEUE

Queue is a linear data structure which follows FIFO principle to perform any operation like insertion, deletion, traversal etc. on it.

→FIFO means First-In-First-Out order in which operations on queues are performed.

→This is also called FCFS which mean first-come-first-serve principle.

→A good example of a queue is: a queue of persons in a ticket counter to buy tickets.

→The person who comes first is served first i.e. first person gets the ticket first.

→The difference between stacks and queues is in removing or deleting the data items: In a stack we remove most recently inserted item but in a queue, we remove the item that was inserted first.

→There are two end points of a queue: rear and front end

→At rear end only insertion operations are allowed.

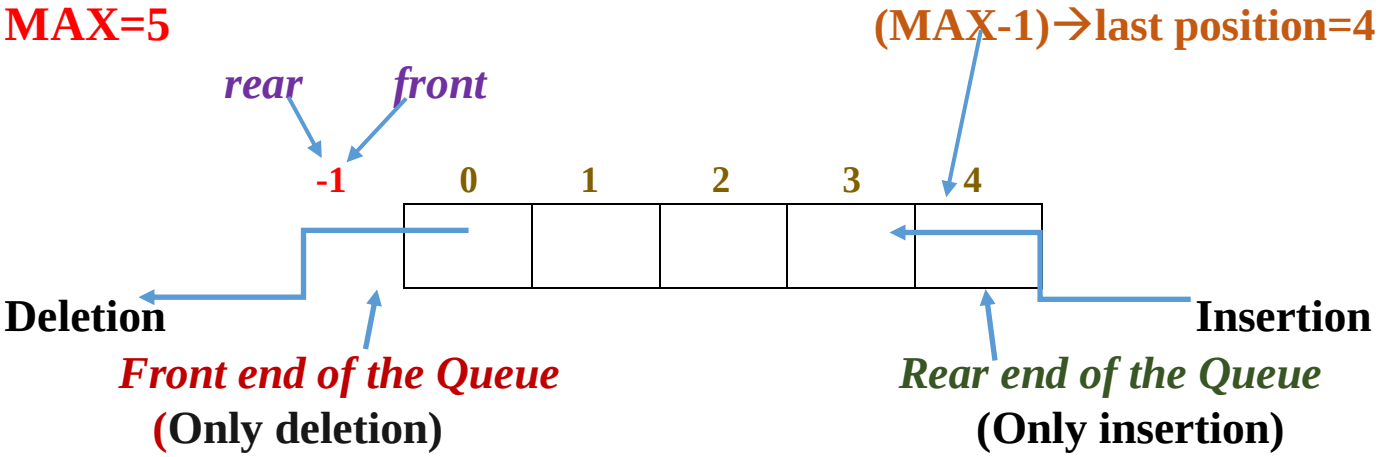
→At the front end only deletions or retrieval operation is allowed.

→Therefore in queue two special reference or pointer variables are used: rear pointer and front pointer.

→Rear pointer always refers index or position of the rearmost element i.e. the last element inserted recently onto the queue.

→Front pointer always refers to index or position of the front-most element i.e. the first element inserted onto the queue.

/*Fig: 1 An empty Queue */



Operations on Queues:

Mainly the following three basic operations are performed on queue:

En-queue (insert_Q)

→Inserts a new data item at rear or back end of the queue.

→If the queue is full, then it is said to be in overflow condition.

De-queue (delete_Q)

→Removes or deletes a data item at front most position from a queue i.e. the first data item inserted onto the queue.

→If the queue is empty, then it is said to be an underflow condition.

Traverse-queue:

→Visit or print each data item of a queue from the front to last position.

Applications of Queue's:

→Queues are used in those application where we need to process data the order in which data items arrive or inserted i.e FCFS order.

→Queue are used when things don't have to be processed immediately, but have to be processed in First-In-First-Out order like Breath first search.

This property of Queue makes it also useful in following kind of scenarios.

→When a resource is shared among multiple consumers.
Examples: include CPU scheduling, Disk Scheduling.

→When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Types of Queue's:

1. **Linear Queue**
2. **Circular Queue**
3. **D-Queue (Double-Ended Queue)**
4. **Priority Queue**

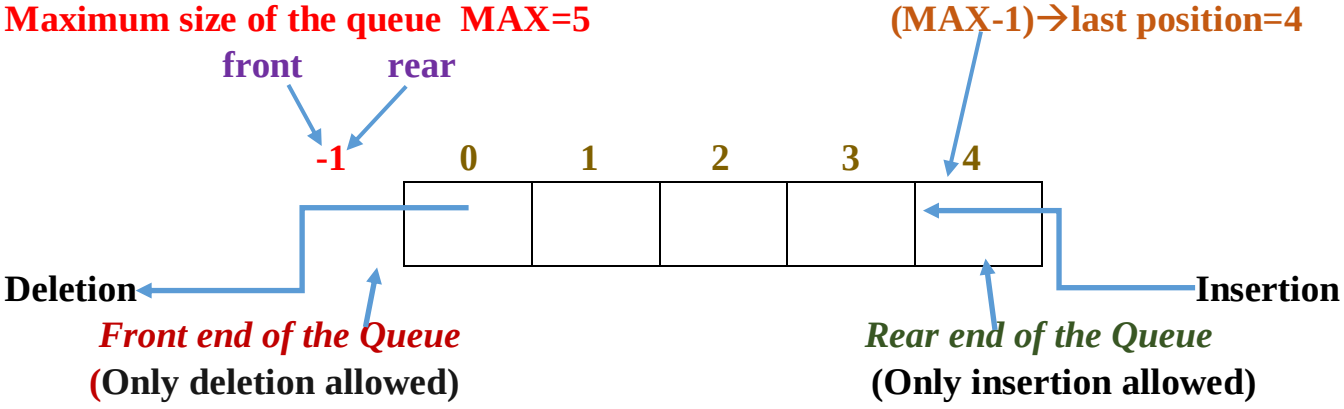
Implementation of Linear Queue:

A queue where insertions are made only at rear and deletions are made only at front end is called a linear Queue.

→In linear queue we can't insert more elements after the rear pointer refers to maximum or the last position of the queue.

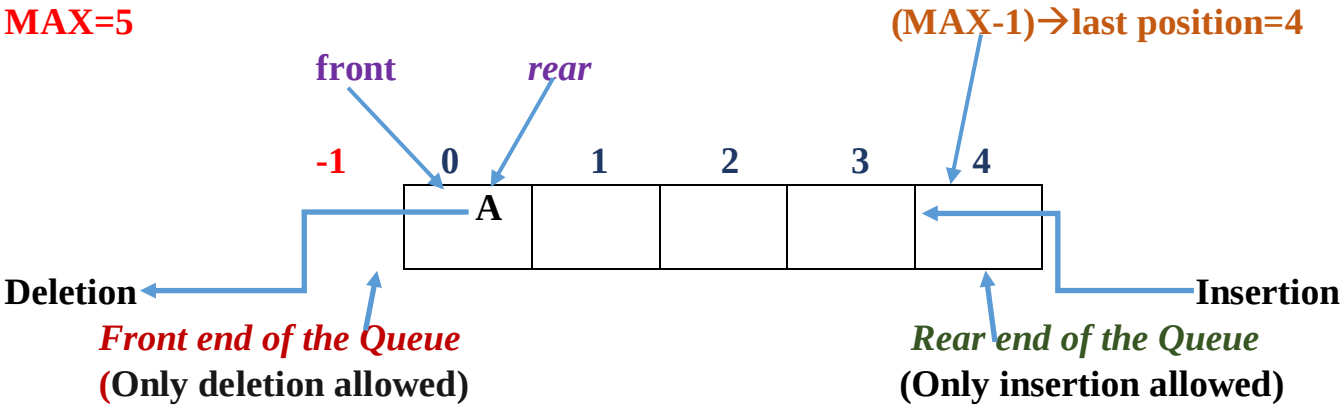
→The value of 'front' pointer is always less than the 'rear' pointer value i.e. always the front most element will be behind the rear element.

/*Fig: 2 An empty Linear Queue */

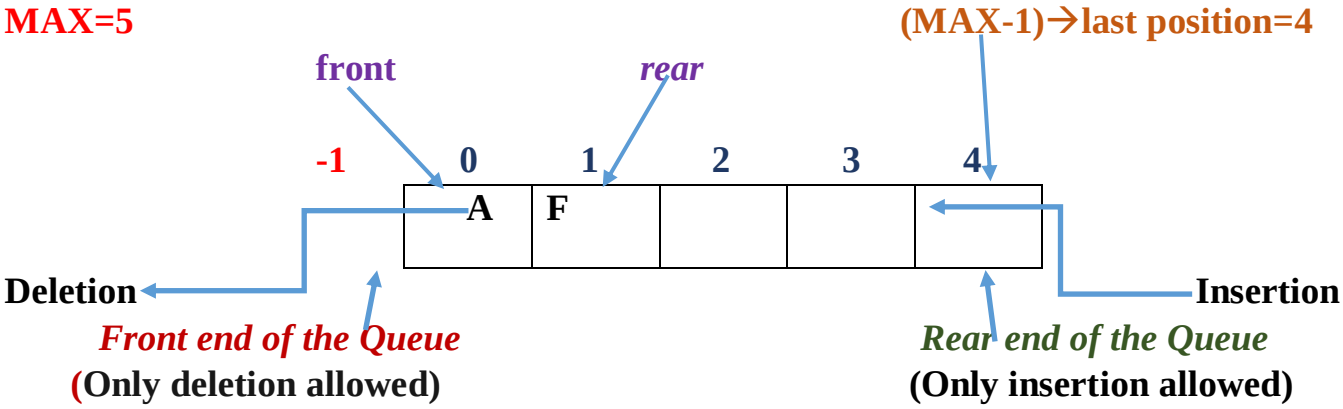


Let's perform a series of insertion operations:

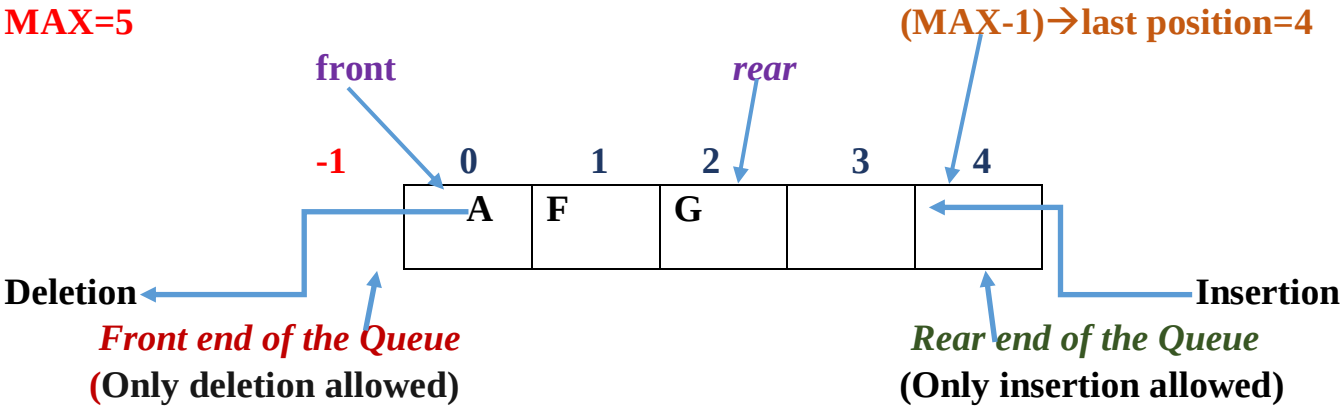
/*Fig: 2 after insertion of 1st item onto the Queue */



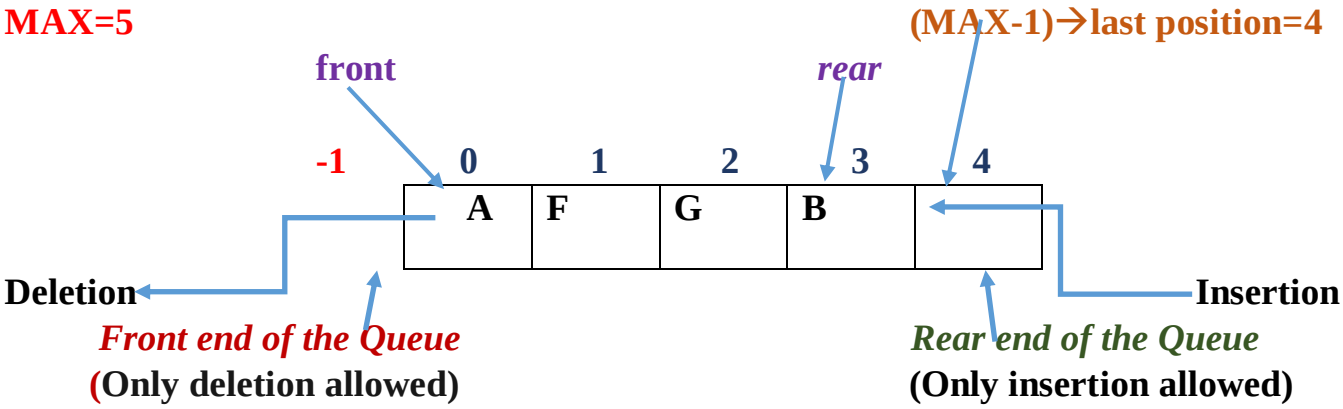
/*Fig: 3 after insertion of 2nd item onto the Queue */



/*Fig: 4 after insertion of 3rd item onto the Queue */

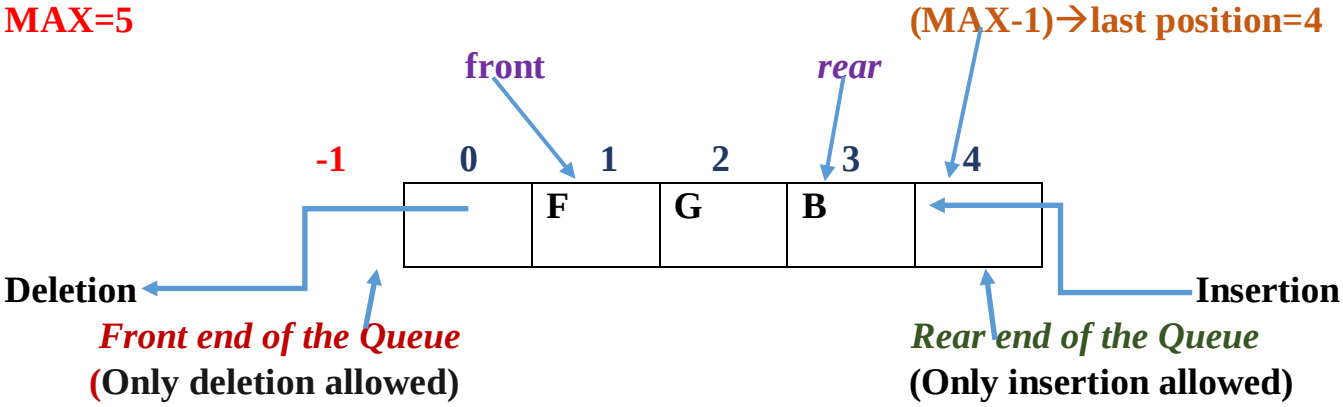


/*Fig: 5 after insertion of 4th item onto the Queue */

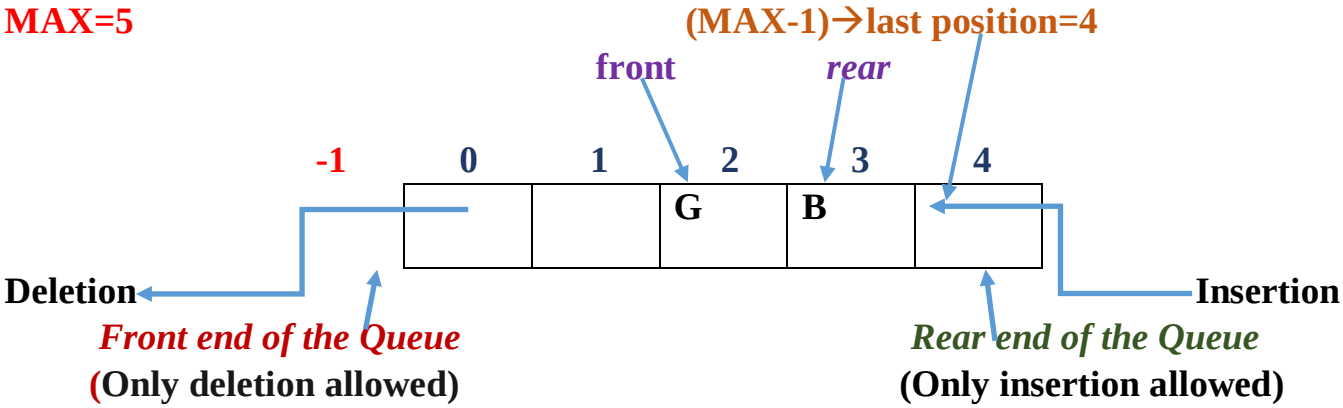


Now let's perform a series of deletion operation:

/*Fig: 6 after deletion of the 1st item from the Queue */

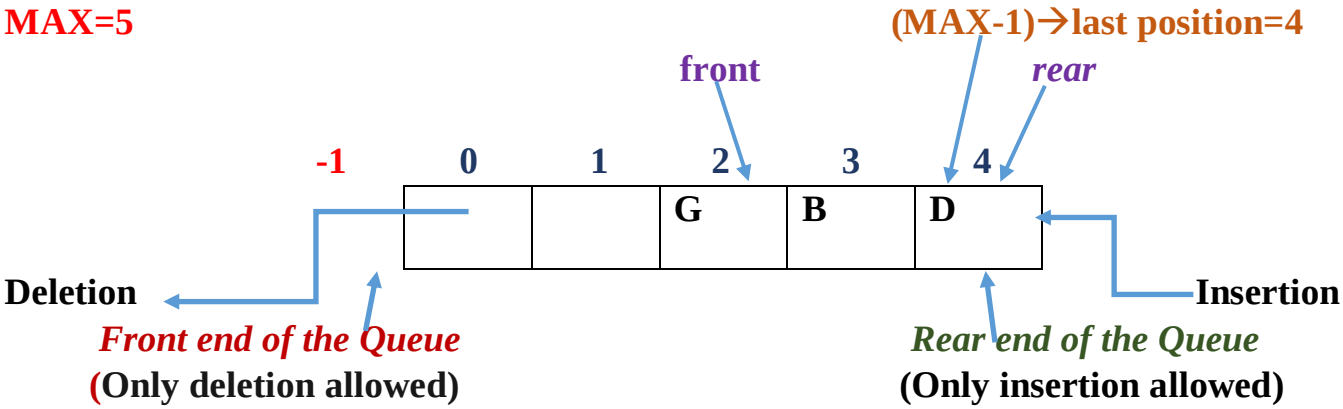


/*Fig: 7 after deletion of the 2nd item from the Queue */



Now let's again perform some insertion operations:

/*Fig: 8 after insertion of another new item onto the Queue */



Now can we insert more items (Yes/No)? Answer is : No

→Because *'rear'* pointer is now referring to the last index of the queue i.e. the condition rear >= (MAX-1) is true.

→But you can see there are empty positions to left side of front most element, still you can't insert more items because in linear queue insertions are only made at rear end and when rear >=MAX-1 the queue becomes overflow.

→This is the major limitation of linear queue.

Array implementation of Linear Queue:

→The first step to create a using arrays is: declare an array of some specified size according to the type of data items you want to store in the queue.

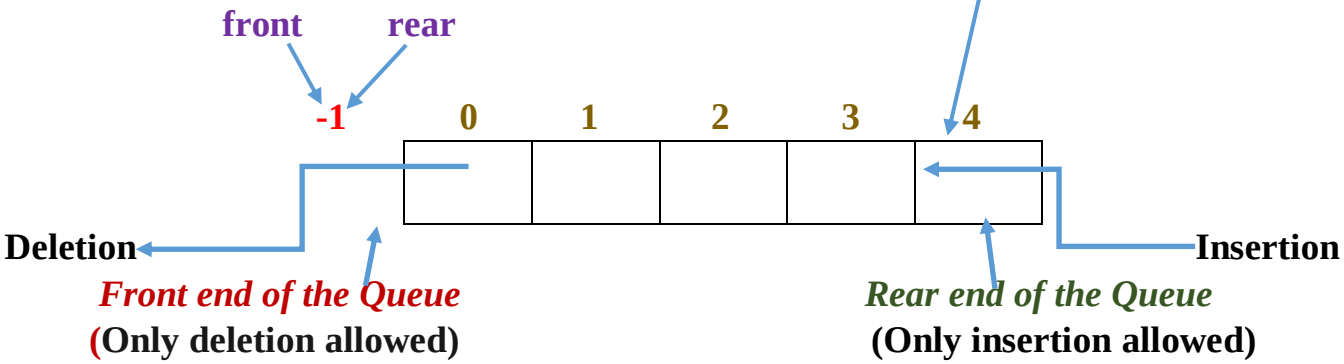
→Declare the two integer type variables called as 'front' and 'rear' pointers that will always refer to front-most and rearmost elements index respectively in the queue.

→To create an empty queue initially, initialize both 'front' and 'rear' pointers by -1 as shown in the following java program:

```
public class Q_ARRAY_DEMO
{
    static int MAX=5;   /*maximum size of the queue*/
    static int front = -1 ; /* creates an empty Q*/
    static int rear = -1 ;  /* creates an empty Q*/
    static char Q[]=new char[MAX]; /*The array Q will hold queue elements*/

/*The above queue Q can hold character type data items */
        public static void main(String[] args)
        {
            Scanner sc=new Scanner(System.in);
            char ch;
            int opt;
            int item;
            do
            {
                System.out.println("1.INSERTQ 2.DELETEQ");
                System.out.println("3.TRAVERSE_Q");

                System.out.println("Enter your option");
                opt=sc.nextInt();

                switch(opt)
                {
                    case 1:
                        System.out.println("Enter the item to push:");
                        item=sc.nextInt();
                        insert_Q(item);
                     break;

                    case 2:
                        delete_Q();
```

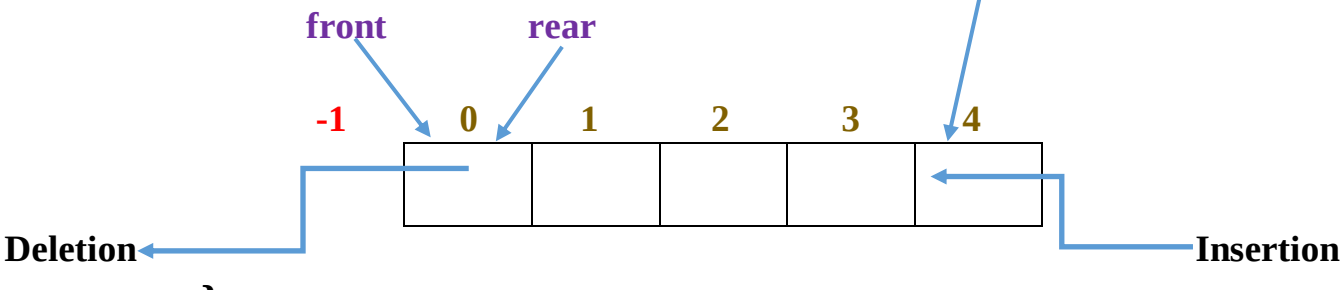
```
                    break;
                case 3:
                    traverse_Q();
                    break;

                default:
                        System.out.println("invalid option");
                } /*End of switch*/
                System.out.println("\nDo you want to perform"
                                    +"another operation(y/n)");
                ch=sc.next().charAt
            }while(ch=='y'|| ch== 'Y'); /*End of do…while loop*/
    }  /* End of main method*/

/* Insert_Q or en-Queue method */
    public static void insert_Q(char new_item)
    {       if ( rear >= MAX-1 )
            {       System.out.println("The Queue is full");
                    System.out.println("You can't insert more items");
                   return;
            }
            else if ( rear < 0 )
            {       /*Fig: 9 When Q is empty: before updating 'rear' & 'front'*/
```

Maximum size of the queue MAX=5 (MAX-1)→last position=4




```
            rear = 0 ;
            front = 0 ;
        /*Fig: 10 after updating 'rear' & 'front' when the Queue is initially empty */
```

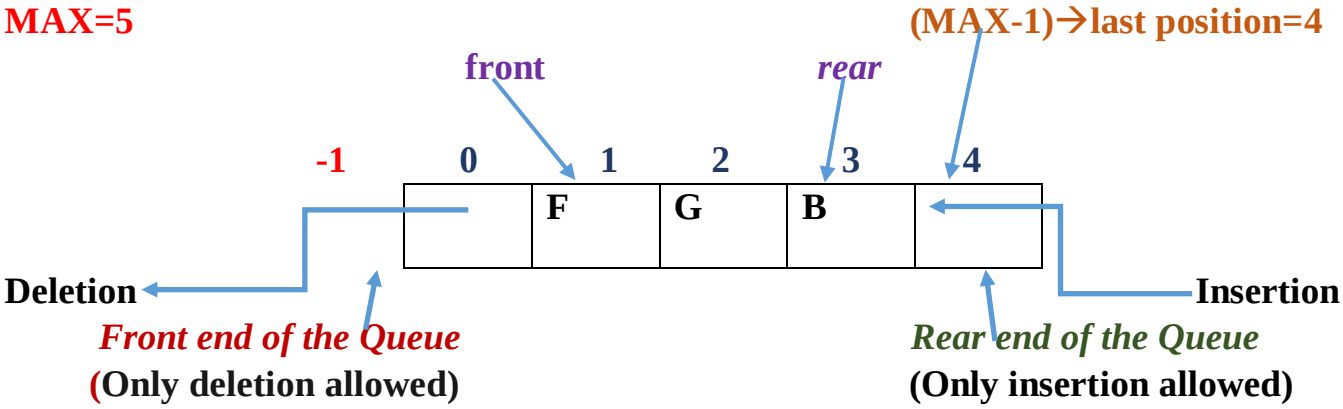
MAX=5 (MAX-1)→last position=4




```
            }
```

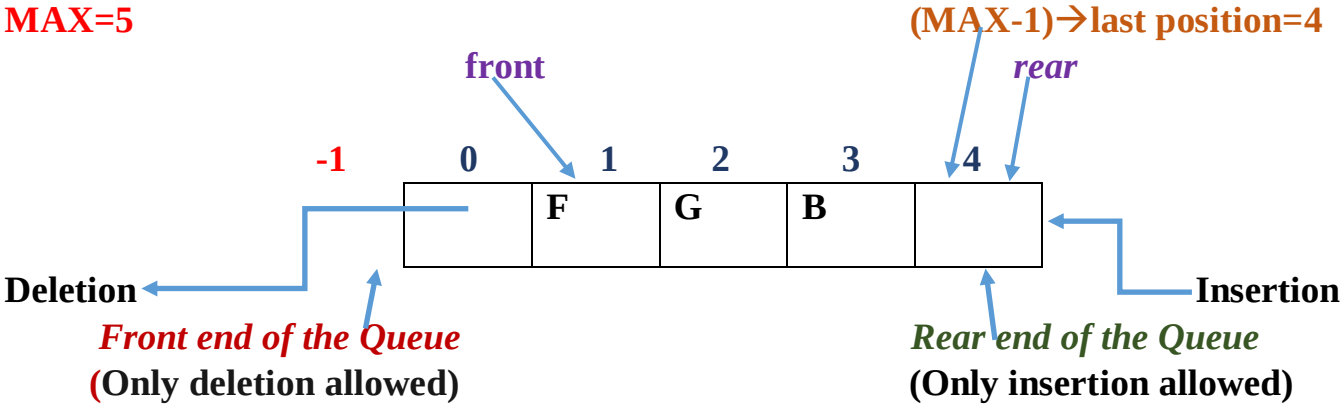
```
else
{
```

/*Fig: 11 before updating only 'rear' when the Q is not empty */



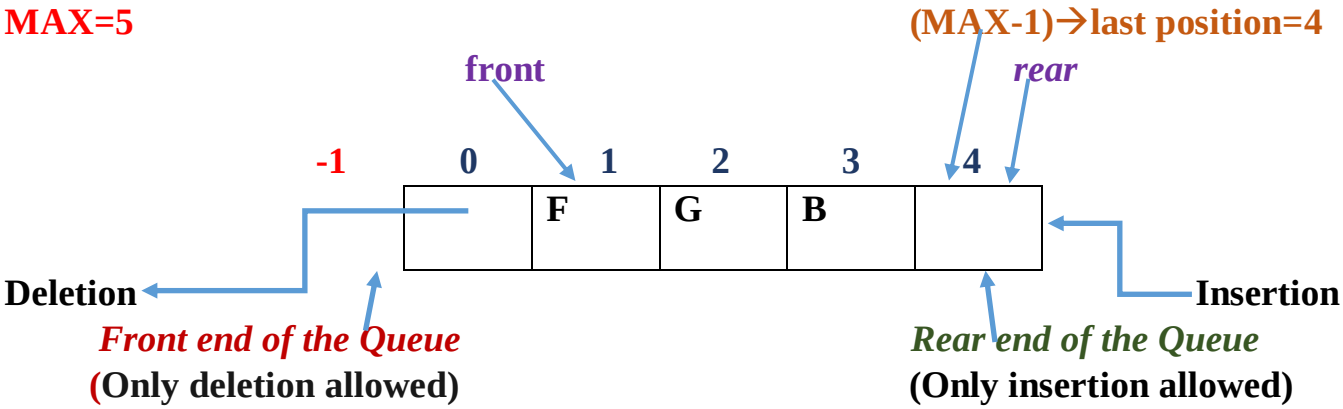

```
rear = rear + 1; /*updates 'rear' pointer to next  empty position*/
```

/*Fig: 12 after updating only 'rear' to the next empty position when the Q is not empty */




```
} /* End of else clause*/
Q[rear]= new_item; /* store the new_item at rear postion*/
```

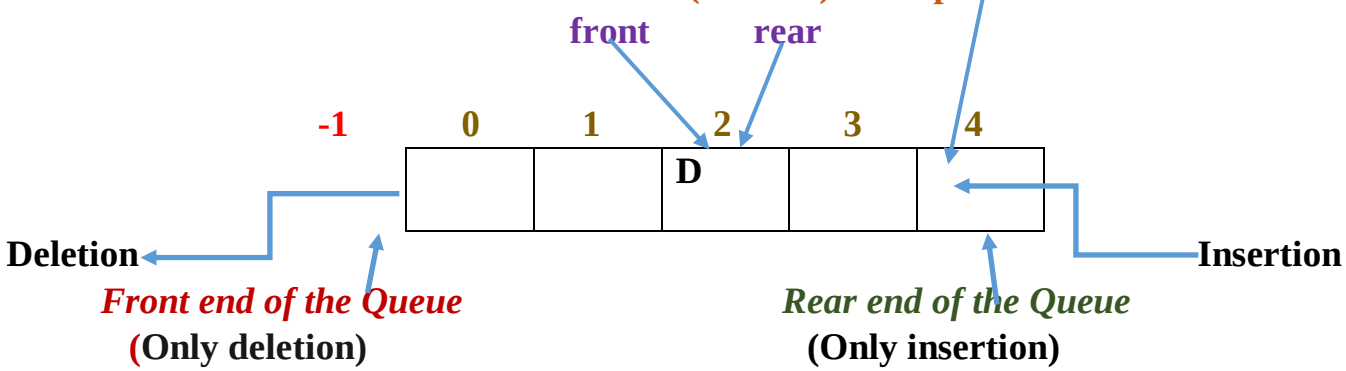
/*Fig: 12 after insertion of the new item at the updated position of 'rear' in the Q */




```
} /* End of insert_Q method*
```

```
/* Delete_Q method*/
public static void delete_Q()
{           if ( front < 0 )
            {
                  System.out.println("The Queue is empty");
                  System.out.println("You can't delete more items");
               return;
            }
           char temp=Q[front];
           if ( rear == front  )
            {       /*Fig: 13 When a Queue contains only one item */
```

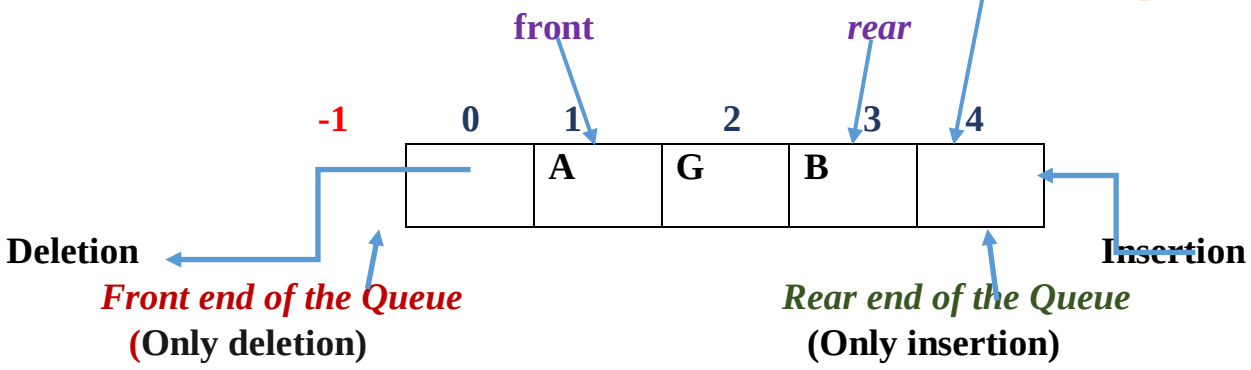
MAX=5 (MAX-1)→last position=4




```
                  rear = -1;
                  front = -1; /* set both 'rear' & 'front' pointer to -1*/
            }  /*End of 'if' clause*/
            else
            {
```

/*Fig: 14 before updating only 'front' pointer when Q contains more than one item*/

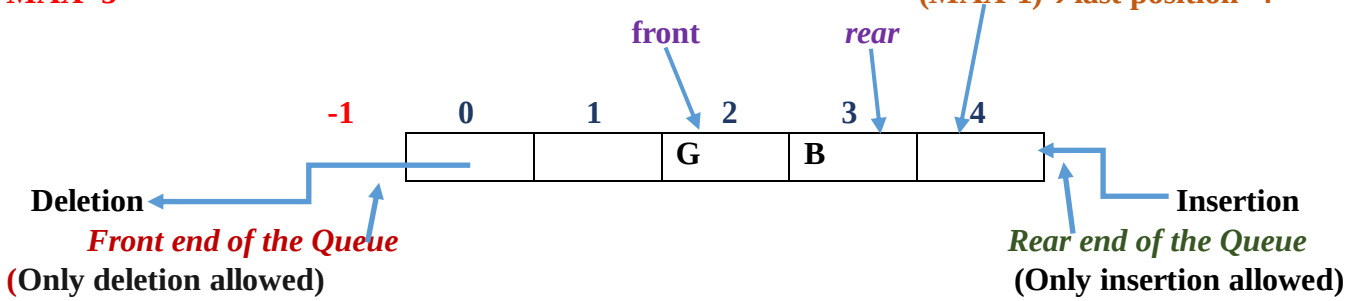
MAX=5 (MAX-1)→last position=4




```
                  front = front + 1; /* front moves to next front item in Q*/
```

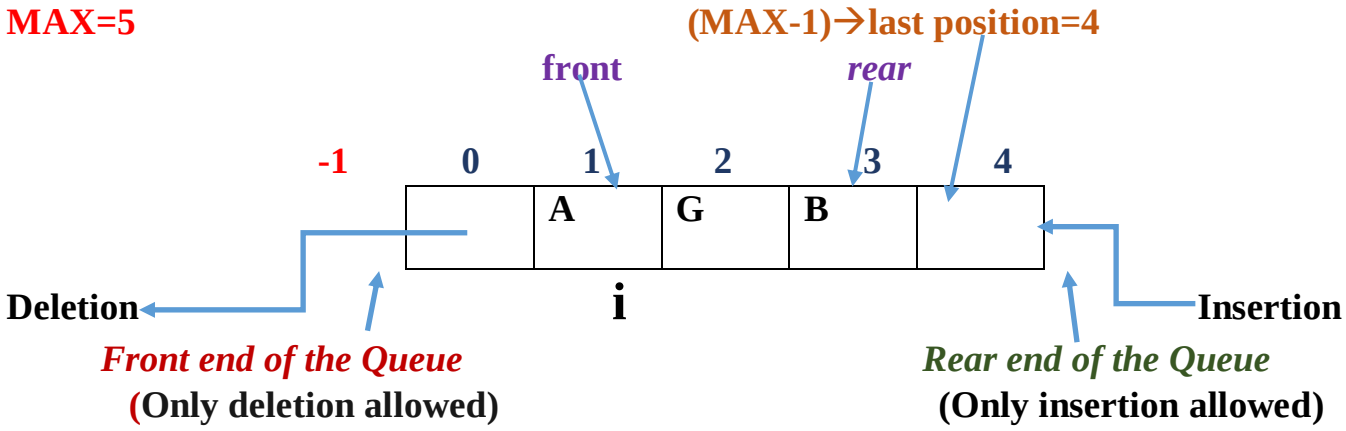
/*Fig: 14 before updating only 'front' pointer when Q contains more than one item*/

MAX=5 (MAX-1)→last position=4



```
            } /* End of else clause*/
            System.out.println("the deleted item is: "+  temp);
    } /* End of de_queue method*/

/* Traversing the Q elements*/
public static void traverse_Q()
        {
            if ( front < 0 )
            {
                 System.out.println("The Queue is empty");
               return;
            }
            else
            {
                System.out.println("The Queue elements are");
                int i=front;
                while ( i <= rear )
                {
                        System.out.print( Q[i] + " → " );
                         i = i + 1;
                }
             } /* end of else clause*/
```

/*Fig: 15 the current status of Queue */




```
        } /* End of traverse_Q method*/

} /* END OF Q_ARRAY_DEMO CLASS*/
```