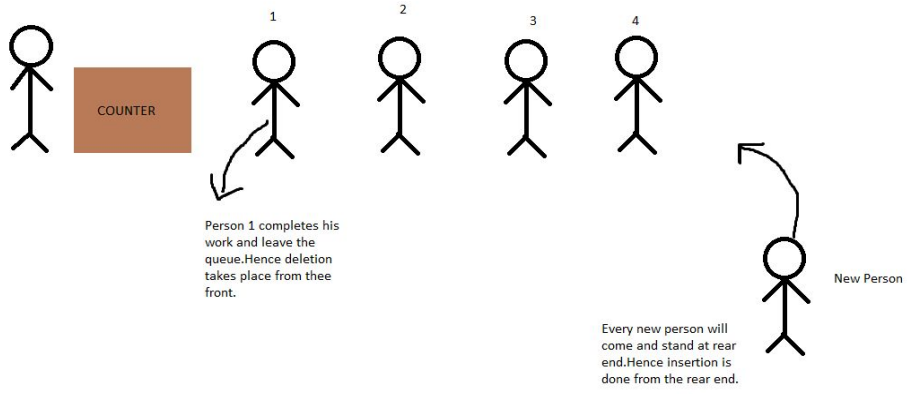
Queue

The next Data Structure that we are going to discuss is Queue Data Structure. Just like Stack, Queue also comes under the category of logical Data Structure. The main difference between stack and queue is the principal on which these Data Structure operates. Stack works on the Principle of LIFO (Last In First Out), whereas **Queue works on the principle of FIFO (First in First out)**.

Now, let us understand the principle of FIFO (First in First out) . As the name suggests , the element which is inserted first is also deleted first . Unlike stack, where all the insertion and deletion operations were performed on the top of the stack, in Queue, the insertion and deletion operations are performed at different ends. Every new element is inserted at the rear end of the queue whereas the deletion always takes place from the front end.

Let us understand the Concept of FIFO with the help of a real life example. Suppose you are standing in a straight line in a bank to withdraw money and waiting for your turn. There are still many people ahead of you in the line and many people behind you waiting for their respective turns. Now, every new person who enters the bank to withdraw money has to stand at the rearest end of the line. Hence every new individual is at the rear end of the line. Similarly, every individual in the line can perform the transaction only and only if the person ahead of him has completed his/her transaction and has left the line. For ex:- The second person in the line can not perform transactions till the first person has not completed his work and same is the case with other people in the line. Hence deletion in the line takes place from front. This line is working on the Principle of FIFO and hence it is a Queue.



ADT(Abstract Data Type of Queue) Of Queue

Now, we know that the ADT of any Data structure consists of two parts i.e Data Representation of that Data Structure and Operation we can perform on that Data Structure. So , ADT Of Queue means what things we need to represent Queue in memory and operations we can perform on Queue . Now let's talk about both of these things on Queue in Detail:-

1.)Data Representation:-

To represent Queue in memory, we require 3 things:-

1)Space for Storing Elements :-

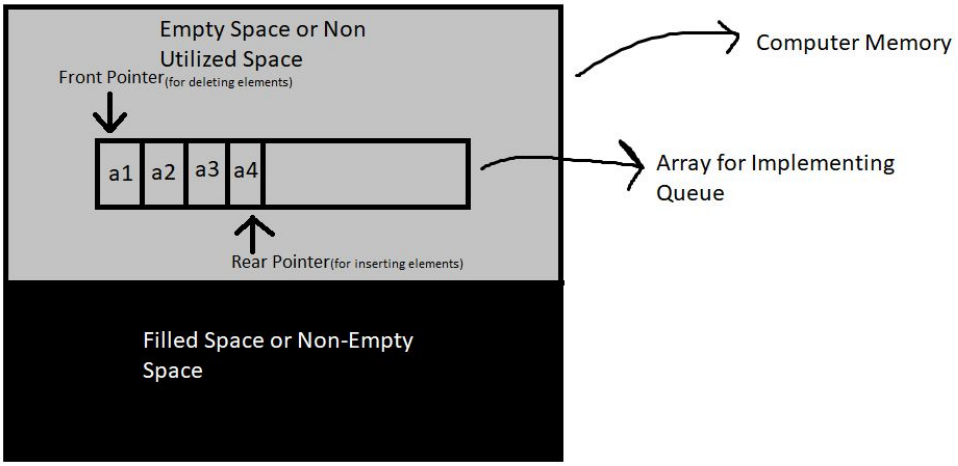
We know that Queue is a Logical Data Structure which can be implemented using any one of the Physical Data Structure i.e Array or Linked list.Now to create an array or linked list and store elements inside them,we require some empty space in memory.

2)Front Pointer:-

As already discussed , elements are deleted from the front in a queue . Hence we require a front pointer which will keep track of the current element in the front and will delete it whenever required.

3)Rear Pointer:-

We also know that new elements are inserted inside a queue from the rear End . Hence we need a rear pointer which will always point to the rear end of the queue and will insert a new element whenever needed.



Data Representation of Queue

2.)Operations on Queue:-

So Basically 6 types of Operations can be performed on Queue.

1) **enqueue(x)** → Insert an element 'x' inside the queue from the rear end.

2) **dequeue()** → Delete the element from the front side of the queue.

3) **isEmpty()** → Finding out whether the Queue is Empty or not.

4) **isFull()** → Finding out whether the Queue is Full or not.

5) **first()** → Finding the current first element of the queue i.e element pointed by the front pointer.

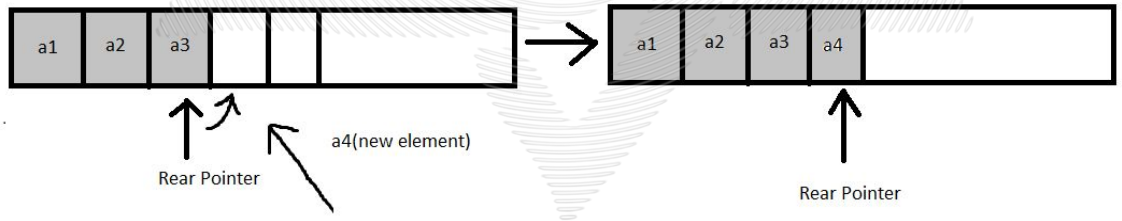
6) **last()** → Finding the current last element of the queue i.e element pointed by the rear pointer.

Implementing Queue using Arrays

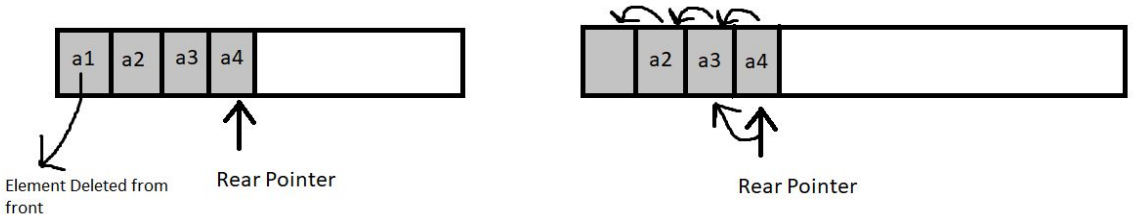
Array is a common physical data structure which can be used to implement Queue. Now, we can implement Queue using one pointer as well as by using 2 pointers. So let us Explore both of them.

A) By using Single Pointer:-

Now let us implement Queue using a single pointer only. Since elements are inserted at the rear end of the Queue , we will name our pointer **rear pointer** . Now,if we want to insert a new element inside the queue , we will just increment the value of the rear pointer and insert the element and hence , Insertion takes only **O(1)** time complexity . But the problem comes while deleting the element . Since we are not using any second pointer to keep track of the current first element , we will always delete the element at index 0 i.e a[0] . Now after deleting a[0] , that space will become empty and hence we have to shift all elements in the array by one position in the left direction to fill the empty space and again delete a[0] if required. Now if we have 'N' elements in array, then we have to shift N-1 elements in array every time whenever we delete an element . Hence deletion in array using Single pointer takes **O(N)** time complexity in average and this is the Major Drawback of Implementing Queue by just using a single pointer and hence we will move to double pointer approach.



Insertion of new Element in queue using a Single Pointer just takes O(1) time complexity



Deletion of element takes O(n) time complexity if Implementing Queue using Single Pointer since shifting of Elements has to be done

B)Implementing Queue Using Two Pointers:-

Now , since deletion using Single Pointer takes O(N) time and this is not what we want , we will change our approach a little bit to implement deletion in constant time complexity just like Insertion . We will make a front pointer which will always point to the current first Element in the Queue.

Now whenever the front most element of Queue is deleted , we will shift our front pointer to the next index of the array and hence our front pointer will be pointing to the current first element of the queue and that can be also deleted using the front pointer.

Hence , in this case we are just shifting our Single front pointer to the next index which takes **O(1) Time Complexity** rather than shifting N-1 elements which takes O(N) time complexity .

This approach for removing elements from the queue is faster , more efficient and easy to implement . Hence, we will implement a queue with an array using two separate pointers **front** and **rear** rather than using one single pointer.

Now let us define the structure for Implementing Queue using Array.

```
1.  struct queue
2.  {
3.  int *Q;//pointer pointing to array
4.  int size;//size variable to store size of array
5.  int front;//front pointer pointing to current front element in array
6.  int rear;//rear pointer pointing to the rear most element of array
7.  }
```

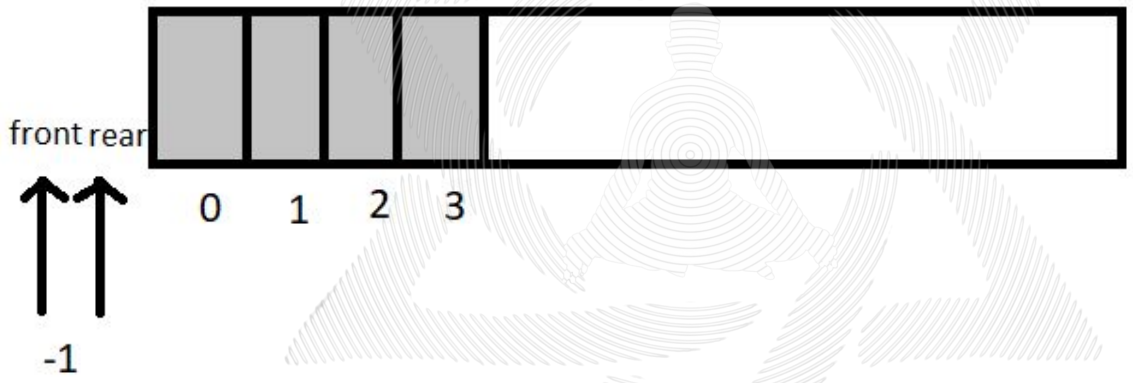
Queue Empty and Full Condition

Let us talk about Queue empty and Queue full conditions . We should know when we will say that a given queue is empty and when a given queue is full in order to prevent the violation of queue structure . We cannot delete anything from an empty queue and cannot add elements to a completely filled queue and hence knowing the Queue empty and Full Condition is important.

A)Queue Empty condition:-

A queue is said to be empty when front and rear pointer are Pointing to the same Index i.e when

Front==Rear



In the above Diagram , both front and rear are pointing to the same index and hence the queue is Empty. If we want to add a new element inside the queue , we must increment the rear pointer by 1 and add our new element. Now if we want to delete this inserted element , we will increment our front pointer by 1 and delete this element and the queue will become empty and again , front and rear will be pointing to the same index. Hence when both front and rear points to the same index , our Queue will become empty.

B)Queue full condition:-

Queue full condition is more easy to visualize and understand as compared to Queue empty condition . As we have seen , the stack is said to be full if the top pointer value becomes *equal to size of the (array - 1)*. Similarly , in a queue , since the data is inserted from the rear end of the array , a queue will be said to become full if the value of the rear pointer becomes equal to *size of (array-1).*

Hence , queue full condition is :

Rear == size-1

Now , let's Implement Queue and its operation using array and Do some Coding.

```
1.  #include <bits/stdc++.h>
2.  #include <iostream>
3.  using namespace std;
4.
5.  //defining a structure Queue
6.  struct Queue
7.  {
8.     int size;
9.     int front;
10.    int rear;
11.    int *Q;
12. };
13.
14. //create function to create a queue of a given size
15. void create(struct Queue *q , int size)
16. {
17.    q->size=size;
18.    q->front = q->rear = 0;
19.    q->Q = (int *)malloc(q->size*sizeof(int));
20.}
21.
```