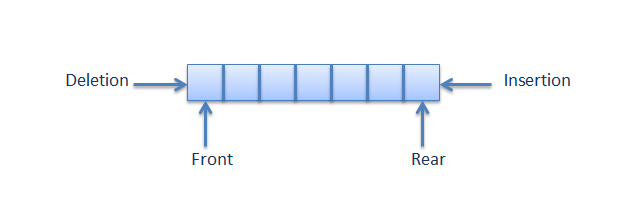
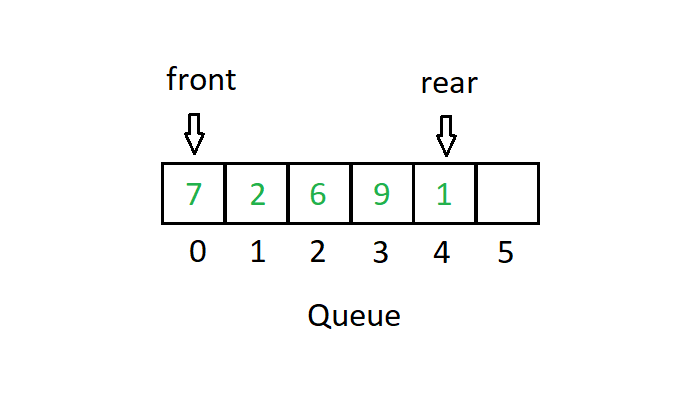
**-: Queue :-**

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing elements are performed at two different positions or ends.

In a queue data structure, the insertion operation is performed at a position which is known as 'rear' and the deletion operation is performed at a position which is known as 'front'. The insertion and deletion operations are performed based on FIFO (First In First Out) principle. 



**Operation performed on Queue: -**

1. Insert element
2. Delete element
3. Traverse and display

Here we are implementing queue using array so we have two ends as shown in the figure. We have two variable one is **front** which is used to delete the element from one end and another is **rear** which is used to add element from another end. Front should always points to the first index position i.e. we will perform the deletion operation from the first position always.

rare should always points to the last element in the list. By the help of rare we are inserting the elements in the list. When the list is empty i.e. not having elements, both the front and rare contains the index position 0.

Creation of Queue:

#define SIZE 6

int queue[SIZE];

int front, rear;

front=rear=0; // Queue is Empty

//These all are taken as global variable so that it can be used any where in the method.

1. **Insertion of element:**

Insertion takes place by the help of variable rear. It points to the position where a new element is inserted. If the queue is empty then it holds 0 i.e. index of first position.

Ex: If array is having 3 elements then rear should hold the 4th index i.e. 3.

When the queue is full at that time rare is not pointing to the to any element and it holds the value = **Size of Queue + 1.**

**Algorithm:**

* First we have to check the queue is full or not if full we cant add element into the queue.
* Then we read the element and insert the element into the queue.
* After insertion we should increase the value of rear by one so that it hold the index of next location.

**Code:**

void enQueue(){ // enQueue is enter + Queue

int data;

if (rear == size) {

printf(“Queue is Full. Element can’t be entered!”);

} else {

printf(“Enter the element: ”);

scanf(“%d”,&data);

queue[rear++] = data;

}

}

1. **Delete the element:**

We do deletion operation by front variable. It always holds 0, index of first element. When the first element in the queue is deleted then the first location is empty and front can’t go to the second location. So we have to shift all the elements to its previous location. So that the 2nd element will come to 1st , 3rd come to 2nd and so on.

**Algorithm: -**

* First we have to check the queue is empty or having some element. If it is having some element then only we are going to perform delete operation.
* Here deletion means no memory will be deleted only the values are overwritten. So the values are shifted from right to left by for loop.
* After deletion we have one less element so rare should point to the last location by rear-- .

**Code: -**

void deQueue () {

int i=0;

if(front == rare){

ptintf(“Queue is Empty! ”);

} else {

for(i=0;i<rear-1 ; i++){

queue[i] = queue[i+1];

}

rear--;

Printf(“Element removed succssfully”);

}

}

1. **Traverse and Display: -**

We can display each element in the queue by traversing the element i.e. visiting each element in the queue.

**Algorithm:** We have to check the queue is empty or not. If it is not empty then only we are going to display the elements in the array. Then we have to use a for loop and isplay all the elements one by one

**Code: -**

void display () {

if (front==rear) {

printf(“The Queue is empty!”);

} else {

int i=0;

printf(“The Queue elements is:\n”);

for(i=0;i<rear;i++){

printf(“%d ”,queue[i]);

}

}

}

**Circular Queue: -**