Queue

A queue is a linear data structure in which Insertion and deletion happens at different end.

Insertion = enqueue
Deletion = dequeue

Queue can be implemented by array, stack or linked list.

front
(Deletion from front end)

fig:- Queue

rear
(Insertion from rear end).

Application

(i) It can be used in shareable resource.
(ii) In real life scenario, call center phone system uses queues to hold people calling them in an order, until a service representative is free.

Queue ADT (Abstract Data Type)

In order to create queue we need a front and rear pointer which stores the front data and rear data in queue.

Some of the queue ADT operations are:

(i) enqueue ( ) :→ To push an element into the queue.

(ii) dequeue ( ) :→ To remove the element from the queue.

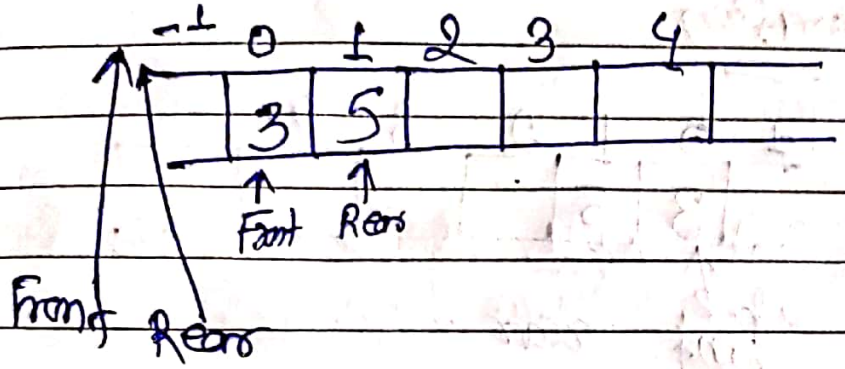
(iii) isEmpty ( ) :→ Determine queue is Empty or not.

(iv) isFull ( ) :→ Determine queue is Full or not.

(v) peek ( ) :→ Value of the front side.

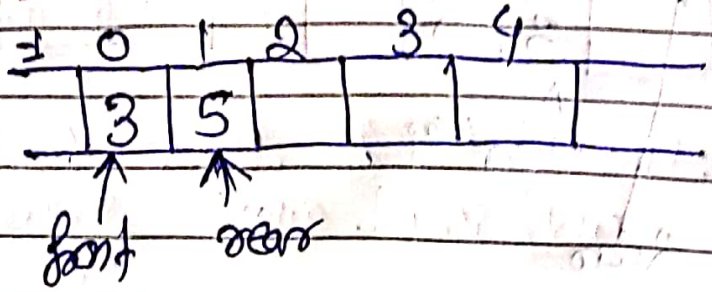
enqueue operation

```
#define MAX 5
int queue[MAX]
int front = -1;
int rear = -1;
```



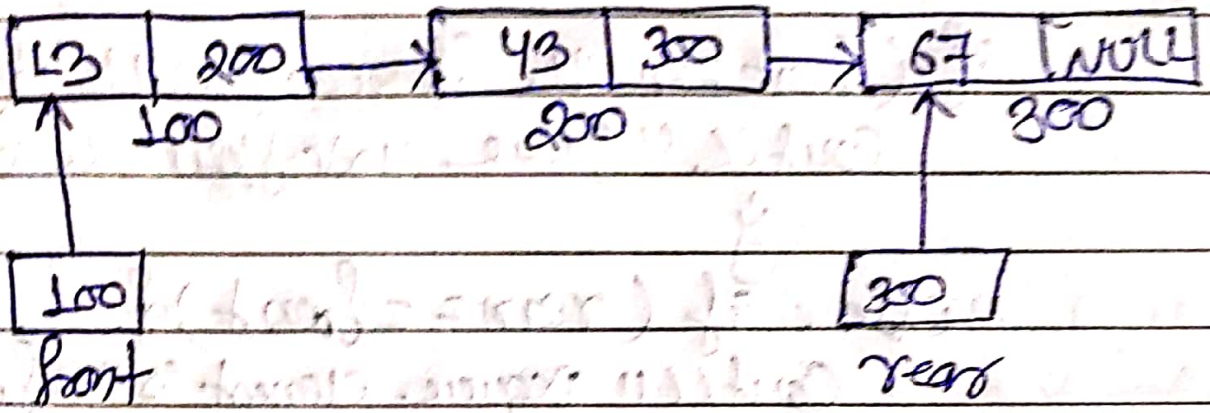

```
void enqueue (int data){
    if (rear = MAX - 1){
cout<<" queue overflow "<<endl;
    }
    else if (front == -1 && rear == -1){
        front = rear = 0;
        queue [rear] = data;
    }

    else {
        rear = rear + 1;
        queue [rear] = data;
    }
```

Dequeue operation




```
void dequeue () {
    if (front == -1 && rear == -1) {
        cout << " queue underflow" << endl;
    }
    else if ( front == rear )
    {
    cout << " deque element is " << queue[front];
        front = rear = -1 ;
    }
    else {
        cout << queue [front];
        front = front + 1 ;
    }
}
```

Using linked list




```
void Enqueue (int data ){
    node * temp = new node ();
    if (temp == NULL){
        cout << " queue overflow" << endl;
    }
    else {
        if (rear == NULL && front == NULL){
            temp -> data = data;
            temp -> next = NULL;
            rear = temp;
            front = temp;
        }
        else{
            temp -> data = data;
            rear -> next = temp;
            temp -> next = NULL;
            rear = temp;
        }
    }
}
```

```
void Dequeue(){
    if(front==NULL && rear==NULL){
        cout<<"Queue underflow"<<endl;
    }
    else if(rear==front){
        cout<<"Dequeue element is"<<front->data;
        rear=front=NULL;
    }
    else{
        node *p=front;
        cout<<"Deque element is"<<front->data;
        front=front->next;
        free(p);
    }
}
```