## **LEC-7 Queue**

//Queue

//Data Structures that stores data on first come first serve or we can say Queue in firt in out(FIFO)

#include <iostream>

using namespace std;

class queue{

private:

    int front;

    int rear;

    int maxsize;

    int currsize;

    int\* arr;

public:

    queue(int n=10){

        front=-1;

        rear=-1;

        maxsize=n;

        currsize=0;

        arr=new int [maxsize];

    }

    ~queue(){

        if(arr!=0){

            delete []arr;

            arr=0;

        }

    }

    bool isFull(){

        if(currsize==maxsize){

            return true;

        }

        return false;

    }

    bool isEmpty(){

        if(currsize==0){

            return true;

        }

        return false;

    }

    bool enqueue(int val){ //insert at end

        if(isFull()){

            return false;

        }

        else{

            rear=(rear+1)%maxsize;

            arr[rear]=val;

            currsize++;

            return true;

        }

    }

    bool dequeue(int val){ //delete from end

        if(isEmpty()){

            return false;

        }

        else{

            front=(front+1)%maxsize;

            currsize--;

            return true;

        }

    }

};

## **Queue Implementation using LinkedList**

#include <iostream>

using namespace std;

class Queue;

class Node{

    friend class Queue;

    int data;

    Node\* next;

};

class Queue{

private:

    Node\* front;

    Node\* rear;

public:

    Queue(){

        front=0;

        rear=0;

    }

    ~Queue();

    bool isEmpty();

    bool Enqueue(int val){ //insert at end

        Node\* temp=new Node;

        temp->data=val;

        temp->next=0;

        if(rear==0){

            front=temp;

            rear=temp;

            return true;

        }

        else{

            rear->next=temp;

            rear=temp;

        }

    }

    bool Dequeue(){ // delete from end

        if(isEmpty()){

            return false;

        }

        if(front==rear){

            delete rear;

            front=0;

            rear=0;

            return true;

        }

        else{

            Node\* temp =front;

            front =front->next;

            delete temp;

            temp=0;

            return true;

        }

    }

};