DSA_College\queue_arrayImplement.cpp

```cpp
// Queue - First In First Out
// Insertion at tail and Pop from head
/*
#include<iostream>
#include<queue>
using namespace std ;

int main()
{
    // create a queue
    // use Queue STL
    queue<int> q ;
    q.push(10);
    q.push(20);
    q.push(40) ;

    cout<<"size of queue is = "<<q.size()<<endl;

    q.pop() ;

    cout<<"size of queue is = "<<q.size()<<endl;
    cout<<"front of queue is = "<<q.front()<<endl;

    if(q.empty())
    {
        cout<<"queue is empty"<<endl;
    }
    else{
        cout<<"queue is not empty"<<endl;
    }

    return 0;
}
*/





/*

// Queue Implementation using Arrays

class Queue
{
    int* arr ;
    int qfront ; // qfront and rear are initially at 0th index of arr
    int rear ;
    int size ;

    public :
```

```cpp

    Queue() // constructor
    {
        size = 1000 ;
        arr = new int(size) ;
        qfront = 0;
        rear = 0;
    }

    void enqueue(int data)    //O(1)
    {
        if(rear == size) // queue is full
        {
            cout<<"queue is full"<<endl;
        }
        else{
            arr[rear] = data ;
            rear++ ;
        }
    }

    int dequeue()    // O(1)
    {
        if(qfront == rear) // empty queue
        {
            return -1 ;
        }
        else{
            int ans = arr[qfront] ;

            arr[qfront] = -1 ; // jo value delete ki hai uski jagah -1 krdo and front ko aage badhao
            qfront++ ;   // kyuki front se hi delete hua hai

            if(qfront == rear)   // empty queue - return the qfront and rear variable back to their original positions
            {                     // so that there is no wastage of memory
                qfront = 0;
                rear = 0;
            }
            return ans ;
        }
    }

    int front()  // O(1)
    {
        if(qfront == rear)
        return -1 ;

        else{
            return arr[qfront] ;
        }
    }

    bool isEmpty()   // O(1)
    {
        return rear == qfront ;
    }
};
*/










/*
// Circular Queue
class circularQueue
{
    int* arr ;
    int front ;
    int rear ;
    int size ;

    public :
    circularQueue(int n)
    {
        size = n ;
        arr = new int(size) ;
        front = rear = -1 ; // front and rear initially array ke bahar hai, dono ko ek aage badhane pr vo 0th index pr aajayenge
    }

    bool enqueue(int value)
    {
        if((front == 0 && rear == size -1) || (rear == (front-1)%(size-1)))
        {
            cout<<"queue is full"<<endl;
            return false ;
        }
        else if(front == -1)
        {
            // first element to push
            front = rear = 0;
            arr[rear] = value ;
        }
        else if(rear == size-1 && front != 0)
        {
            rear = 0; // 0th index(front) khali hai and rear last mai hai, then ab push krne pr
                     // rear 0th index pr aajayega
            arr[rear] = value ;
        }
        else{
            rear++ ;
            arr[rear] = value ;
        }
        return true ;
    }

    int dequeue()
    {
        if(front == -1) // to check if queue is empty
        {
            cout<<"queue is empty"<<Endl;
            return -1 ;
        }

        int ans = arr[front] ;
        arr[front] = -1 ;
        if(front == rear)
        {
            // single element is present
            front = rear = -1 ;
        }
        else if(front == size-1) // to maintain cyclic nature
        {
            front = 0 ;
        }
        else{ // normal flow
            front++ ;
        }
        return ans ;
    }
};
*/
```