Implement a Queue

Problem Statement Suggest Edit

Implement a Queue Data Structure specifically to store integer data using a Singly Linked List or an array.
You need to implement the following public functions :

```
1. Constructor: It initializes the data members as required.

2. enqueue(data): This function should take one argument of type
integer. It enqueues the element into the queue.

3. dequeue(): It dequeues/removes the element from the front of the
queue and in turn, returns the element being dequeued or removed. In
case the queue is empty, it returns -1.

4. front(): It returns the element being kept at the front of the queue.
In case the queue is empty, it returns -1.

5. isEmpty(): It returns a boolean value indicating whether the queue is
empty or not.
```

Operations Performed On The Queue :

```
Query-1(Denoted by an integer 1): Enqueues integer data to the queue.

Query-2(Denoted by an integer 2): Dequeues the data kept at the front of
the queue and returns it to the caller, return -1 if no element is
present in the queue.

Query-3(Denoted by an integer 3): Fetches and returns the data being
kept at the front of the queue but doesn't remove it, unlike the dequeue
function, return -1 if no element is present in the queue.

Query-4(Denoted by an integer 4): Returns a boolean value denoting
whether the queue is empty or not.
```

Detailed explanation (Input/output format, Notes, Images)

Constraints :

```
1 <= t <= 5
1 <= q <= 5000
1 <= x <= 4
1 <= data <= 2^31 - 1

Time Limit: 1 sec
```

Sample Input 1 :

```
1
7
1 17
1 23
1 11
2
2
2
2
```

Sample Output 1 :

```
17
23
11
-1
```

Explanation For Sample Output 1 :

```
The first three queries are of enQueue, so we will push 17, 23, and 11 into the queue.

The next four queries are of deQueue, so we will starting removing elements from the queue,
so the first element will be 17, then 23, and then 11. And after the third dequeue query, the
queue is now empty so for the fourth query, we will return -1.
```

Question Link:- **https://www.codingninjas.com/codestudio/problems/queue-using-array-or-singly-linked-list_2099908**

Code : -

```
#include <bits/stdc++.h>
class Queue {
    int * arr;
    int size;
    int f;
    int rear;
public:
    Queue() {
        size=100001;
        f=rear=0;
        arr=new int[size];
        // Implement the Constructor
    }

    /*---------------- Public Functions of Queue ----------------*/

    bool isEmpty() {
        // Implement the isEmpty() function
        if(f==rear){
            return true;
        }
        else{
            return false;
```

```
        }
    }

    void enqueue(int data)
    {
        // Implement the enqueue() function
        if(rear==size){
            cout<<"Queue is full"<<endl;
        }
        arr[rear]=data;
        rear++;
    }

    int dequeue() {
        // Implement the dequeue() function
        if(f==rear){
            return -1;
        }
        else{
        int ele=arr[f];
        arr[f]=-1;
        f++;
        if (f==rear) {
          f = 0;
          rear = 0;
        }
        return ele;
        }
    }

    int front() {
        // Implement the front() function
        if(isEmpty()){
            return -1;
        }
        int ele = arr[f];
        return ele;
    }
};
```

Circular Queue

You will be given 'Q' queries. You need to implement a circular queue according to those queries. Each query will belong to one of these two types:

```
1 'X': Enqueue element 'X' into the end of the nth queue.
Returns true if the element is enqueued, otherwise false.

2: Dequeue the element at the front of the nth queue. Returns -1
if the queue is empty, otherwise, returns the dequeued element.
```

Note:

```
Enqueue means adding an element to the end of the queue, while
Dequeue means removing the element from the front of the queue.
```

Constraints:

```
1 <= N <= 1000
1 <= Q <= 10^5
1 <= P <= 2
1 <= X <= 10^5

Time limit: 1 sec
```

Sample Input 1:

```
3 7
1 2
1 3
2
1 4
1 6
1 7
```

Sample Output 1:

```
True
True
2
True
True
False
3
```

Explanation Of Sample Output 1:

```
For this input, we have the size of the queue, 'N' = 3, and the
number of queries, 'Q' = 7.

Operations performed on the queue are as follows:

push(2): Push element ‘2’ into the queue. This returns true.
push(3): Push element ‘3’ into the queue. This returns true.
pop(): Pop the top element from the queue. This returns 2.
push(4): Push element ‘4’ into the queue. This returns true.
push(6): Push element ‘6’ into the queue. This returns true.
```

Question Link : - https://www.codingninjas.com/codestudio/problems/circular-queue_1170058?leftPanelTab=0

Approach 1: -(Some Test Cases will fail but it’s Class Rooms approach):

Code :-

```
#include <bits/stdc++.h>
class CircularQueue{
    public:
    // Initialize your data structure.
    int *arr;
    int size;
    int f;
    int r;
    CircularQueue(int n){
        // Write your code here.
        size=100001;
        arr=new int[size];
```

```
        f=r=-1;
    }

    // Enqueues 'X' into the queue. Returns true if it gets pushed into the st
ack, and false otherwise.
    bool enqueue(int value){
        // Write your code here.
        if(f==0 && r==size-1){
            return false;
        }
        else if(r==(f-1)%(size-1)){
            return false;
        }
        else{
            if(f==-1){
                f=0;
            }
            r=(r+1)%(size);
            arr[r]=value;
            return true;
        }
    }

    // Dequeues top element from queue. Returns -
1 if the stack is empty, otherwise returns the popped element.
    int dequeue(){
        // Write your code here.
        if(f==-1){
            return -1;
        }
        else
        {
        int ele=arr[f];
        if(f==r){
            f=-1;
            r=-1;
        } else {
          f = (f + 1) % (size-1);
        }
        return ele;
        }
    }

};
```

Code 2:- Proper Approach

```
class CircularQueue{
    // Initialise front and rear of the queue.
    int rear, front;

    // Size of the queue.
    int d;

    // Array To be used for the implementation.
    vector<int> arr;
//use int *arr; (here)

public:
    // Initialize the circular queue.
    CircularQueue(int n){
        d = n;

        // For a circular queue the front and rear both start as empty.
        front = rear = -1;

        // Create the arrays.
        arr = vector<int>(d);
    }

    // Enqueues 'X' into the Queue. Returns true if it gets enqueued into the
queue, and false otherwise..
    bool enqueue(int value){
        if ((front == 0 && rear == d - 1) || (rear == (front - 1) % (d - 1)))
{
            /*
               If the queue is full, no more elements can be added so return
false.
               Queue will be full if front is at 1st element and rear is at
last element.
               OR if rear is equal to front - 1.
```

```
            */
            return false;
        }
        else if (front == -1) {
            // Queue is empty, hence insert the first element.
            front = rear = 0;
            arr[rear] = value;
        }
        else if (rear == d - 1 && front != 0) {
            // If rear reaches end of queue but the first element is empty.
            rear = 0;
            arr[rear] = value;
        }
        else{
            // Otherwise simply enqueue the element.
            rear ++;
            arr[rear] = value;
        }
        return true;
    }

    // Dequeues top element from the Queue. Returns -1 if the queue is empty,
otherwise returns the dequeued element.
    int dequeue(){
        if (front == -1) {
            // If queue is empty.
            return -1;
        }

        // Initialise element to store dequeud element.
        int data = arr[front];
        arr[front] = -1;
```

```
        if (front == rear) {
            // If the queue has only one element.
            front = -1;
            rear = -1;
        }
        else if (front == d - 1){
            // If front is the last element of the queue.
            front = 0;
        }
        else{
            // In all other cases simply dequeue.
            front++;
        }
        // Return the dequeued element.
        return data;
    }
};
```

Implement Deque

Problem Statement

Design a data structure to implement deque of size 'N'. It should support the following operations:

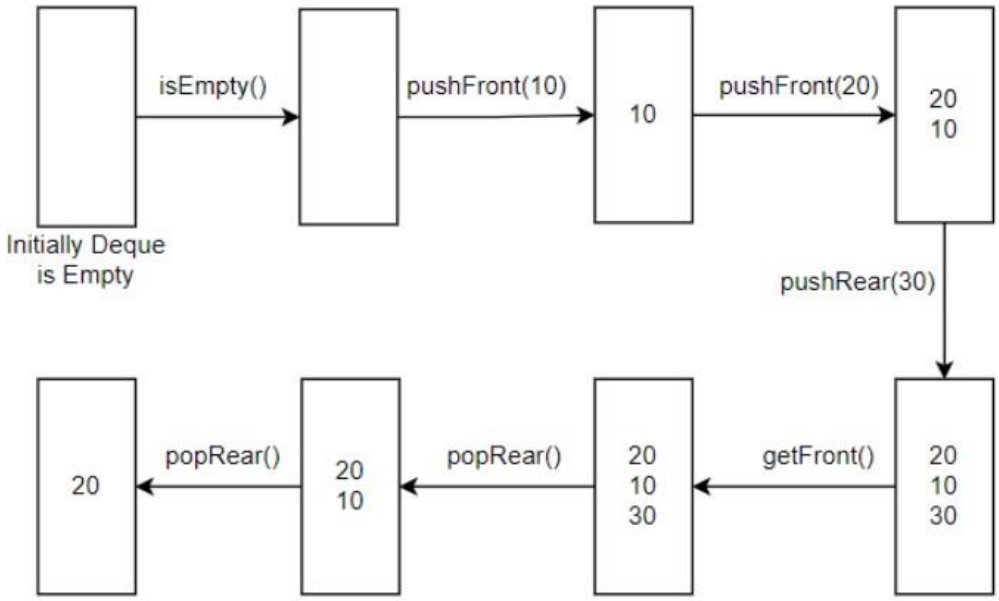
```
pushFront(X): Inserts an element X in the front of the deque.
Returns true if the element is inserted, otherwise false.

pushRear(X): Inserts an element X in the back of the deque.
Returns true if the elemen inserted, otherwise false.

popFront(): Pops an element from the front of the deque. Returns
-1 if the deque is empty, otherwise returns the popped element.

popRear(): Pops an element from the back of the deque. Returns
-1 if the deque is empty, otherwise returns the popped element.

getFront(): Returns the first element of the deque. If the deque
is empty, it returns -1.

getRear(): Returns the last element of the deque. If the deque
is empty, it returns -1.

isEmpty(): Returns true if the deque is empty, otherwise false.

isFull(): Returns true if the deque is full, otherwise false.
```

Sample Input 1:

```
5 7
7
1 10
1 20
2 30
5
4
4
```

Sample Output 1:

```
True
True
True
True
20
30
10
```



Question Link:- https://www.codingninjas.com/codestudio/problems/deque_1170059

Code:-

```
class Deque
{
    int *arr;
    int f;
    int r;
    int size;
public:
    // Initialize your data structure.
    Deque(int n)
    {
        size = n;
        arr = new int[n];
        f = -1;
        r = -1;
    }

    // Pushes 'X' in the front of the deque. Returns true if it gets pushed into the deque, and false otherwise.
    bool pushFront(int x)
```

```
    {
        //check full or not
        if( isFull() ) {
            return false;
        }
        else if(isEmpty()) {
            f=r= 0;
        }
        else if(f== 0&&r!= size-1) {
            f= size-1;
        }
        else
        {
            f--;
        }
        arr[f] = x;
        return true;
    }


    // Pushes 'X' in the back of the deque. Returns true if it gets pushed
into the deque, and false otherwise.
    bool pushRear(int x)
    {
        if( isFull() ) {
            return false;
        }
        else if(isEmpty()) {
            f=r= 0;
        }
        else if(r==size-1 && f!= 0) {
            r= 0;
        }
        else
```

```
        {
             r++;
        }
        arr[r] = x;
        return true;
    }

    // Pops an element from the front of the deque. Returns -1 if the deque is
empty, otherwise returns the popped element.
    int popFront()
    {
        if(isEmpty()){//to check queue is empty
            //cout << "Queue is Empty " << endl;
            return -1;
        }

        int ans = arr[f];
        arr[f]=-1;

        if(f==r) { //single element is present
            f=r=-1;
        }
        else if(f== size - 1) {
            f= 0; //to maintain cyclic nature
        }
        else
        {//normal flow
            f++;
        }
        return ans;
    }
```

```
    // Pops an element from the back of the deque. Returns -1 if the deque is
empty, otherwise returns the popped element.
    int popRear()
    {
         if(isEmpty()){//to check queue is empty
            //cout << "Queue is Empty " << endl;
            return -1;
        }

        int ans = arr[r];
        arr[r] = -1;

        if(f== r) { //single element is present
            f=r= -1;
        }
        else if(r== 0) {
            r= size-1; //to maintain cyclic nature
        }
        else
        {//normal flow
            r--;
        }
        return ans;
    }

    // Returns the first element of the deque. If the deque is empty, it
returns -1.
    int getFront()
    {
        if(isEmpty()){
            return -1;
        }
        return arr[f];
```

```
    }


    // Returns the last element of the deque. If the deque is empty, it
returns -1.
    int getRear()
    {
       if(isEmpty()){
            return -1;
        }
        return arr[r];
    }
    bool isEmpty()
    {
        if(f==-1)
            return true;
        else
            return false;
    }
    bool isFull()
    {
                                                    if( (f== 0 && r== size-1) ||
(f!= 0 && r== (f-1)%(size-1) ) ) {
            return true;
        }
        else
        {
            return false;
        }
    }
};
```