641. Design Circular Deque

Design your implementation of the circular double-ended queue (deque).
Implement the `MyCircularDeque` class:

- `MyCircularDeque(int k)` Initializes the deque with a maximum size of `k`.
- `boolean insertFront()` Adds an item at the front of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean insertLast()` Adds an item at the rear of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean deleteFront()` Deletes an item from the front of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean deleteLast()` Deletes an item from the rear of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `int getFront()` Returns the front item from the Deque. Returns `-1` if the deque is empty.
- `int getRear()` Returns the last item from Deque. Returns `-1` if the deque is empty.
- `boolean isEmpty()` Returns `true` if the deque is empty, or `false` otherwise.
- `boolean isFull()` Returns `true` if the deque is full, or `false` otherwise.

Example 1:

```
Input
["MyCircularDeque", "insertLast", "insertLast", "insertFront", "insertFront",
"getRear", "isFull", "deleteLast", "insertFront", "getFront"]
[[3], [1], [2], [3], [4], [], [], [], [4], []]
Output
[null, true, true, true, false, 2, true, true, true, 4]

Explanation
MyCircularDeque myCircularDeque = new MyCircularDeque(3);
myCircularDeque.insertLast(1);  // return True
myCircularDeque.insertLast(2);  // return True
myCircularDeque.insertFront(3); // return True
myCircularDeque.insertFront(4); // return False, the queue is full.
myCircularDeque.getRear();      // return 2
myCircularDeque.isFull();       // return True
myCircularDeque.deleteLast();   // return True
myCircularDeque.insertFront(4); // return True
myCircularDeque.getFront();     // return 4
```

Constraints:

- `1 <= k <= 1000`
- `0 <= value <= 1000`
- At most `2000` calls will be made to `insertFront`, `insertLast`, `deleteFront`, `deleteLast`, `getFront`, `getRear`, `isEmpty`, `isFull`.

641. Design Circular Deque

```
/*
   Double linked list
   Time & space complexity: all functions are O(1)
*/
class MyCircularDeque{
   public:
      // Create double linked list node
      class Node{
         public:
            int val;
            Node* next;
            Node* prev;
         public:
            Node(int val,Node* next,Node* prev):val(val),next(next),prev(prev){}
      };

      // head and tail
      Node* head=nullptr;
      Node* tail=nullptr;

      // Actual size
      int _size;

      // max size
      int _capacity;
```

public:

```
    MyCircularDeque(int k) {
        _size=0;
        _capacity=k;
    }

    bool insertFront(int value) {
        if(isFull()) return false;
        if(!head) head=tail=new Node(value,nullptr,nullptr);
        else{
            Node* new_node=new Node(value,head,nullptr);
            head->prev=new_node;
            head=new_node;
        }
        _size++;
        return true;
    }

    bool insertLast(int value) {
        if(isFull()) return false;
        if(!tail) head=tail=new Node(value,nullptr,nullptr);
        else{
            Node* new_node=new Node(value,nullptr,tail);
            tail->next=new_node;
            tail=new_node;
        }
        _size++;
        return true;
    }

    bool deleteFront() {
        if(isEmpty()) return false;
        Node* tmp=head;
        if(_size>1){
            head=head->next;
            head->prev=nullptr;
            tmp->next=nullptr;
        }
        else head=tail=nullptr;
        delete tmp;
        _size--;
        return true;
    }
```

```
    bool deleteLast() {
        if(isEmpty()) return false;
        Node* tmp=tail;
        if(_size>1){
            tail=tail->prev;
            tail->next=nullptr;
            tmp->prev=nullptr;
        }
        else head=tail=nullptr;
        delete tmp;
        _size--;
        return true;
    }

    int getFront() {
        if(isEmpty()) return -1;
        return head->val;
    }

    int getRear() {
        if(isEmpty()) return -1;
        return tail->val;
    }

    bool isEmpty() {
        return _size==0;
    }

    bool isFull() {
        return _size==_capacity;
    }
};
```