# 题目

设计实现双端循环队列。

你的实现需要支持以下操作：

MyCircularDeque(k)：构造函数,双端队列的大小为k。

insertFront()：将一个元素添加到双端队列头部。如果操作成功返回true。

insertLast()：将一个元素添加到双端队列尾部。如果操作成功返回true。

deleteFront()：从双端队列头部删除一个元素。如果操作成功返回true。

deleteLast()：从双端队列尾部删除一个元素。如果操作成功返回true。

getFront()：从双端队列头部获得一个元素。如果双端队列为空，返回-1。

getRear()：获得双端队列的最后一个元素。如果双端队列为空，返回-1。

isEmpty()：检查双端队列是否为空。

isFull()：检查双端队列是否满了。

示例：

MyCircularDeque circularDeque = new MycircularDeque(3); // 设置容量大小为3

circularDeque.insertLast(1); // 返回 true

circularDeque.insertLast(2); // 返回 true

circularDeque.insertFront(3); // 返回 true

circularDeque.insertFront(4); // 已经满了，返回 false

circularDeque.getRear(); // 返回 2

circularDeque.isFull(); // 返回 true

circularDeque.deleteLast(); // 返回 true

circularDeque.insertFront(4); // 返回 true

circularDeque.getFront(); // 返回 4

提示：

所有值的范围为[1, 1000]

操作次数的范围为[1, 1000]

请不要使用内置的双端队列库。

# 分析

**思路：**

head是头部待出队位置（获取头元素，即直接访问arr[head]）

从头插入，head是逆时针旋转（需要先减1）

从头删除，head是顺时针旋转

tail是尾部待入队位置，（获取尾元素，需要先将tail逆时针旋转1）

从尾插入，tail是顺时针旋转

从尾删除，tail是逆时针旋转

**代码：**

class MyCircularDeque {

private:

int \*m\_data;

int m\_head;

int m\_tail;

int m\_size;

int m\_capacity;

public:

/\*\* Initialize your data structure here. Set the size of the deque to be k. \*/

MyCircularDeque(int k) {

m\_data = new int[k];

m\_head = m\_tail = 0;

m\_size = 0;

m\_capacity = k;

}

/\*\* Adds an item at the front of Deque. Return true if the operation is successful. \*/

bool insertFront(int value) {

if(isFull())

return false;

m\_head--;//在前面插入，索引应该减小

if(m\_head < 0)

{

m\_head = m\_capacity -1;

}

m\_data[m\_head] = value;

m\_size++;

return true;

}

/\*\* Adds an item at the rear of Deque. Return true if the operation is successful. \*/

bool insertLast(int value) {

if(isFull())

return false;

m\_data[m\_tail] = value; //先赋值，然后m\_tail++

m\_tail++;

if(m\_tail == m\_capacity)

{

m\_tail = 0;

}

m\_size++;

return true;

}

/\*\* Deletes an item from the front of Deque. Return true if the operation is successful. \*/

bool deleteFront() {

if(isEmpty())

return false;

m\_head++;

if(m\_head == m\_capacity)

{

m\_head = 0;

}

m\_size--;

return true;

}

/\*\* Deletes an item from the rear of Deque. Return true if the operation is successful. \*/

bool deleteLast() {

if(isEmpty())

return false;

m\_tail--;

if(m\_tail < 0)

{

m\_tail = m\_capacity-1;

}

m\_size--;

return true;

}

/\*\* Get the front item from the deque. \*/

int getFront() {

if(isEmpty())

return -1;

return m\_data[m\_head];

}

/\*\* Get the last item from the deque. \*/

int getRear() {

if(isEmpty())

return -1;

int index = m\_tail - 1;

if(index < 0)

{

index = m\_capacity - 1;

}

return m\_data[index];

}

/\*\* Checks whether the circular deque is empty or not. \*/

bool isEmpty() {

return (0 == m\_size);

}

/\*\* Checks whether the circular deque is full or not. \*/

bool isFull() {

return (m\_size == m\_capacity);

}

};

/\*\*

\* Your MyCircularDeque object will be instantiated and called as such:

\* MyCircularDeque\* obj = new MyCircularDeque(k);

\* bool param\_1 = obj->insertFront(value);

\* bool param\_2 = obj->insertLast(value);

\* bool param\_3 = obj->deleteFront();

\* bool param\_4 = obj->deleteLast();

\* int param\_5 = obj->getFront();

\* int param\_6 = obj->getRear();

\* bool param\_7 = obj->isEmpty();

\* bool param\_8 = obj->isFull();

\*/