设计循环队列

设计你的循环队列实现。 循环队列是一种线性数据结构，其操作表现基于 FIFO（先进先出）原则并且队尾被连接在队首之后以形成一个循环。它也被称为“环形缓冲器”。

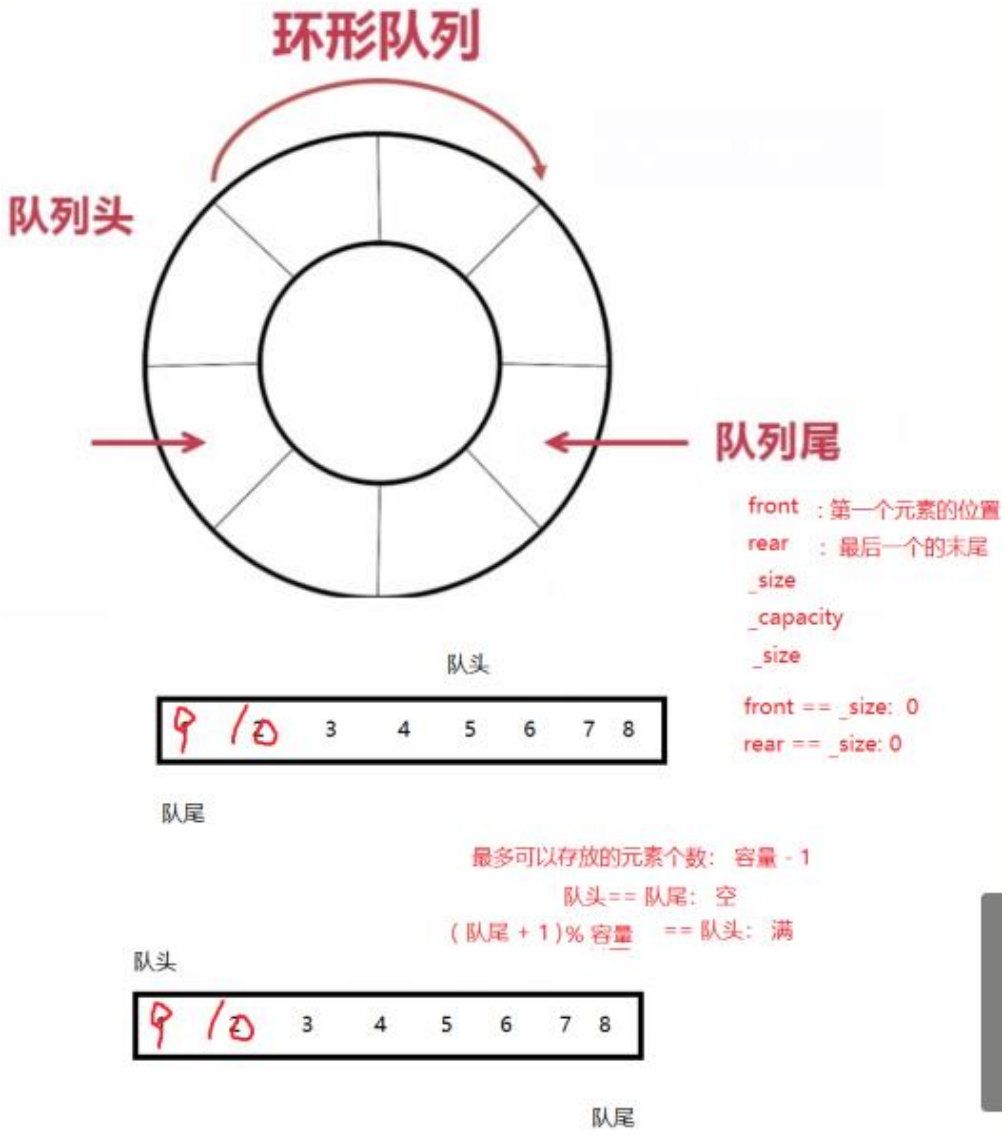
循环队列的一个好处是我们可以利用这个队列之前用过的空间。在一个普通队列里，一旦一个队列满了，我们就不能插入下一个元素，即使在队列前面仍有空间。但是使用循环队列，我们能使用这些空间去存储新的值。

你的实现应该支持如下操作：

- `MyCircularQueue(k)`: 构造器，设置队列长度为 k 。
- `Front`: 从队首获取元素。如果队列为空，返回 -1 。
- `Rear`: 获取队尾元素。如果队列为空，返回 -1 。
- `enQueue(value)`: 向循环队列插入一个元素。如果成功插入则返回真。
- `deQueue()`: 从循环队列中删除一个元素。如果成功删除则返回真。
- `isEmpty()`: 检查循环队列是否为空。
- `isFull()`: 检查循环队列是否已满。

示例：

```
MyCircularQueue circularQueue = new MyCircularQueue(3); // 设置长度为 3

circularQueue.enQueue(1);  // 返回 true

circularQueue.enQueue(2);  // 返回 true

circularQueue.enQueue(3);  // 返回 true

circularQueue.enQueue(4);  // 返回 false，队列已满

circularQueue.Rear();  // 返回 3

circularQueue.isFull();  // 返回 true

circularQueue.deQueue();  // 返回 true

circularQueue.enQueue(4);  // 返回 true

circularQueue.Rear();  // 返回 4
```




```
typedef struct {
    int *data;
    int front;//指向队头元素
    int rear;//指向队尾元素的下一个
    int size;//当前有效元素个数
    int capcity;//容量
} MyCircularQueue;


MyCircularQueue* myCircularQueueCreate(int k) {
    if(k<=0) return NULL;
    MyCircularQueue* obj=(MyCircularQueue*)malloc(sizeof(MyCircularQueue));
    obj->data=(int*)(malloc(sizeof(int)*k));
    obj->front=0;
    obj->rear=0;
    obj->size=0;
```

```
    obj->capcity=k;
    return obj;
}

bool myCircularQueueIsEmpty(MyCircularQueue* obj) {
    if(obj->size==0)
        return true;
    return false;
}

bool myCircularQueueIsFull(MyCircularQueue* obj) {
    if(obj->size==obj->capcity)
        return true;
    return false;
}

bool myCircularQueueEnQueue(MyCircularQueue* obj, int value) {
    if(myCircularQueueIsFull(obj))
        return false;
    obj->data[obj->rear++]=value;//添加：尾指针后移
    if(obj->rear>=obj->capcity)//如果尾指针后退越界，重置为0
    {
        obj->rear=0;
    }
    obj->size++;
    return true;
}

bool myCircularQueueDeQueue(MyCircularQueue* obj) {
    if(myCircularQueueIsEmpty(obj))
        return false;
    obj->front++;//删除：头指针后移
    if(obj->front>=obj->capcity)//如果头指针后退越界，重置为0
    {
        obj->front=0;
    }
    obj->size--;
    return true;
}

int myCircularQueueFront(MyCircularQueue* obj) {
    if(myCircularQueueIsEmpty(obj))
        return -1;
    return obj->data[obj->front];
```

```
}

int myCircularQueueRear(MyCircularQueue* obj) {
    if(myCircularQueueIsEmpty(obj))
        return -1;
    if(obj->rear!=0)//如果尾指针不在头部，回退一位
        return obj->data[obj->rear-1];
    else//如果尾指针在头部，返回数组末尾元素，数组大小为 capcity，尾元素序号为
capcity-1
        return obj->data[obj->capcity-1];
}

void myCircularQueueFree(MyCircularQueue* obj) {
    free(obj->data);
    obj->data=NULL;
    free(obj);
    obj=NULL;
}
```