用队列实现栈

使用队列实现栈的下列操作：

- **push(x) -- 元素 x 入栈**
- **pop() -- 移除栈顶元素**
- **top() -- 获取栈顶元素**
- **empty() -- 返回栈是否为空**

注意：

- 你只能使用队列的基本操作-- 也就是 `push to back`, `peek/pop from front`, `size`, 和 `is empty` 这些操作是合法的。
- 你所使用的语言也许不支持队列。 你可以使用 list 或者 deque（双端队列）来模拟一个队列, 只要是标准的队列操作即可。
- 你可以假设所有操作都是有效的（例如, 对一个空的栈不会调用 pop 或者 top 操作）。

```
typedef int QDataType;
typedef struct QNode
{
    QDataType _data;
    struct QNode* _next;
}QNode;

typedef struct Queue {
    //包含头尾指针
    struct QNode* _head;
    struct QNode* _tail;
    int _size;
}Queue;
QNode* createNode(QDataType val)
{
    QNode* node = (QNode*)malloc(sizeof(QNode));
    if (node != NULL)
    {
        node->_data = val;
        node->_next = NULL;
    }
    return node;
}

//初始化队列
void initQueue(Queue* q)
```

```
{
    if (q == NULL)
        return;
    q->_head = NULL;
    q->_tail = NULL;
    q->_size = 0;
}

//队尾插入
void queuePush(Queue* q, QDataType val)
{
    if (q == NULL)
        return;
    struct QNode* newnode = createNode(val);
    if (q->_head == NULL)//若头为空
        q->_head = q->_tail = newnode;
    //头不为空，尾指针存在
    else
    {
        q->_tail->_next= newnode;
        q->_tail = newnode;//更新尾指针
    }
    q->_size++;
}

//队头删除
void queuePop(Queue* q)
{
    if (q == NULL || q->_head == NULL)
        return;
    struct QNode* next = q->_head->_next;
    free(q->_head);
    q->_head = next;
    if (q->_head == NULL)//如果空队列 head=tail  头删之后 head 为空  此时为了不
让 tail 变野指针，将其置 NULL
        q->_tail = NULL;
    q->_size--;
}

//返回队头元素
QDataType queueFront(Queue* q)
{
    return q->_head->_data;
}
```

```c
//返回队尾元素
QDataType queueBack(Queue* q)
{
    return q->_tail->_data;
}

int EmptyQueue(Queue* q)
{
    return q->_head == NULL;
}
//获取队列大小
int queueSize(Queue* q)
{
    if (q == NULL)
        return 0;
    return q->_size;
}
//销毁队列
void queueDestory(Queue* q)
{
    QNode* cur = q->_head;
    while (cur)
    {
        QNode* next = cur->_next;
        free(cur);
        cur = next;
    }
    q->_head = q->_tail = NULL;
}
typedef struct {
    struct Queue q;
} MyStack;

/** Initialize your data structure here. */

MyStack* myStackCreate() {
    //动态创建
    MyStack* st=(MyStack*)malloc(sizeof(MyStack));
    initQueue(&st->q);
    return st;
}

/** Push element x onto stack. */
```

```
void myStackPush(MyStack* obj, int x) {
    queuePush(&obj->q,x);
}

/** Removes the element on top of the stack and returns that element. */
int myStackPop(MyStack* obj) {
    int n=queueSize(&obj->q);
    while(n>1)//前 n-1 个元素先出队再入队
    {
        int front=queueFront(&obj->q);
        queuePop(&obj->q);
        queuePush(&obj->q,front);
        n--;
    }
    //此时 n=1；
    int top=queueFront(&obj->q);
    queuePop(&obj->q);
    return top;
}

/** Get the top element. */
int myStackTop(MyStack* obj) {
    return queueBack(&obj->q);
}

/** Returns whether the stack is empty. */
bool myStackEmpty(MyStack* obj) {
    return EmptyQueue(&obj->q);
}

void myStackFree(MyStack* obj) {
    queueDestory(&obj->q);
    free(obj);
}

/**
 * Your MyStack struct will be instantiated and called as such:
 * MyStack* obj = myStackCreate();
 * myStackPush(obj, x);

 * int param_2 = myStackPop(obj);

 * int param_3 = myStackTop(obj);
```

```
 * bool param_4 = myStackEmpty(obj);

 * myStackFree(obj);
*/
```