用栈实现队列

请你仅使用两个栈实现先入先出队列。队列应当支持一般队列的支持的所有操作（push、pop、peek、empty）：

实现 MyQueue 类：

- void push(int x) 将元素 x 推到队列的末尾
- int pop() 从队列的开头移除并返回元素
- int peek() 返回队列开头的元素
- boolean empty() 如果队列为空，返回 true ；否则，返回 false

说明：

- 你只能使用标准的栈操作 —— 也就是只有 push to top, peek/pop from top, size, 和 is empty 操作是合法的。
- 你所使用的语言也许不支持栈。你可以使用 list 或者 deque（双端队列）来模拟一个栈，只要是标准的栈操作即可。

进阶：

- 你能否实现每个操作均摊时间复杂度为 O(1) 的队列？换句话说，执行 n 个操作的总时间复杂度为 O(n)，即使其中一个操作可能花费较长时间。

示例：

输入：

```
["MyQueue", "push", "push", "peek", "pop", "empty"]
```

```
[[], [1], [2], [], [], []]
```

输出：

```
[null, null, null, 1, 1, false]
```

解释：

```
MyQueue myQueue = new MyQueue();
```

```
myQueue.push(1); // queue is: [1]
```

```
myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)
```

```
myQueue.peek(); // return 1

myQueue.pop(); // return 1, queue is [2]

myQueue.empty(); // return false
```

```c
typedef int STDataType;
typedef struct Stack
{
    STDataType* _data;
    //int _top; // 栈顶
    int _size;
    int _capacity; // 容量
}Stack;
//检查容量
void checkCapcity(Stack* ps)
{
    if (ps->_size == ps->_capacity)
    {
        int newCap = ps->_capacity == 0 ? 1 : 2 * ps->_capacity;
        ps->_data = (STDataType*)realloc(ps->_data, sizeof(STDataType)*newC
ap);
        ps->_capacity = newCap;
    }
}
// 初始化栈
void StackInit(Stack* ps)
{
    if (ps == NULL)
        return;
    ps->_data = NULL;
    ps->_size = ps->_capacity = 0;
}
// 入栈
void StackPush(Stack* ps, STDataType data)
{
    if (ps == NULL)
        return;
    checkCapcity(ps);
    ps->_data[ps->_size++] = data;
}
// 出栈
void StackPop(Stack* ps)
{
    if (ps == NULL || ps->_size == 0)
```

```
        return;
    ps->_size--;

}
// 获取栈顶元素
STDataType StackTop(Stack* ps)
{
    return ps->_data[ps->_size - 1];
}

// 检测栈是否为空，如果为空返回非零结果，如果不为空返回 0
int StackEmpty(Stack* ps)
{
    if (ps == NULL||ps->_size==0)
        return 1;
    else
        return 0;
}
//销毁栈
void StackDestory(Stack* ps)
{
    free(ps->_data);
    ps->_size=0;
    ps->_capacity=0;
}

typedef struct {
    struct Stack in_st;//入队栈
    struct Stack out_st;//出队栈
} MyQueue;

/** Initialize your data structure here. */

MyQueue* myQueueCreate() {
     //动态创建
    MyQueue* q=(MyQueue*)malloc(sizeof(MyQueue));
    StackInit(&q->in_st);
    StackInit(&q->out_st);
    return q;
}

/** Push element x to the back of queue. */
void myQueuePush(MyQueue* obj, int x) {
    StackPush(&obj->in_st,x);
```

```
}

/** Removes the element from in front of queue and returns that element. */
int myQueuePop(MyQueue* obj) {
    int top;
    if(!StackEmpty(&obj->out_st))//出队栈非空
    {
        top=StackTop(&obj->out_st);
        StackPop(&obj->out_st);
    }
    else//出队栈为空  将入队栈元素放入出队栈
    {
       while(!StackEmpty(&obj->in_st))
       {
           StackPush(&obj->out_st,StackTop(&obj->in_st));
           StackPop(&obj->in_st);
       }
       top=StackTop(&obj->out_st);
       StackPop(&obj->out_st);
    }
    return top;
}

/** Get the front element. */
int myQueuePeek(MyQueue* obj) {
    int top;
    if(StackEmpty(&obj->out_st))//出队栈空
    {
       while(!StackEmpty(&obj->in_st))
       {
           StackPush(&obj->out_st,StackTop(&obj->in_st));
           StackPop(&obj->in_st);
       }
       top=StackTop(&obj->out_st);
    }
    else
    {
       top=StackTop(&obj->out_st);
    }
    return top;
}

/** Returns whether the queue is empty. */
bool myQueueEmpty(MyQueue* obj) {
```

```
    return StackEmpty(&obj->in_st)&&StackEmpty(&obj->out_st);
}

void myQueueFree(MyQueue* obj) {
    StackDestory(&obj->in_st);
    StackDestory(&obj->out_st);
    free(obj);
}

/**
 * Your MyQueue struct will be instantiated and called as such:
 * MyQueue* obj = myQueueCreate();
 * myQueuePush(obj, x);

 * int param_2 = myQueuePop(obj);

 * int param_3 = myQueuePeek(obj);

 * bool param_4 = myQueueEmpty(obj);

 * myQueueFree(obj);
*/
```