

思路：用后进先出的栈来实现先进先出的队列，和用队列实现栈一个思路，来回导。但和用队列实现栈还是有一些不一样，两个队列导的时候，数据的顺序是不会发生改变的，但如果用两个栈来实现队列的话，例如一开始进栈的数据为1，2，3，4，那么导入到另一个栈里时数据就变为了4，3，2，1。所以此时连续出数据的时候不用每次都要导了，直接出的数据就为1，2，3，4。入数据的时候也是一直在同一个栈中入数据。所以两个栈各有分工，一个可以叫做pushST，一个叫做popST。

代码：

typedef int STDataType;

typedef struct Stack

{

    STDataType\* a;

    int top;  //栈顶

    int capacity;  //容量,方便增容

}Stack;

void StackInit(Stack\* pst);

void StackDestory(Stack\* pst);

//性质决定在栈顶出入数据

void StackPush(Stack\* pst,STDataType x);

void StackPop(Stack\* pst);

STDataType StackTop(Stack\* pst);

bool StackEmpty(Stack\* pst);

int StackSize(Stack\* pst);

void StackInit(Stack\* pst)

{

    assert(pst);

    //pst->a = NULL;

    //pst->top = 0;

    //pst->capacity = 0;

    //或者一上来给一些空间

    pst->a = (STDataType\*)malloc(sizeof(STDataType) \* 4);

    pst->top = 0;

    pst->capacity = 4;

}

void StackDestory(Stack\* pst)

{

    assert(pst);

    free(pst->a);

    pst->a = NULL;

    pst->capacity = pst->top = 0;

}

void StackPush(Stack\* pst, STDataType x)

{

    assert(pst);

    if (pst->top == pst->capacity)

    {

        STDataType\* tmp = (STDataType\*)realloc(pst->a,sizeof(STDataType) \* pst->capacity \* 2);

        if (tmp == NULL)

        {

            printf("realloc fail\n");

            exit(-1);  //结束整个程序

        }

        pst->a = tmp;

        pst->capacity \*= 2;

    }

    //入数据

    pst->a[pst->top] = x;

    pst->top++;

}

void StackPop(Stack\* pst)

{

    assert(pst);

    assert(!StackEmpty(pst));

    pst->top--;

}

STDataType StackTop(Stack\* pst)

{

    assert(pst);

    assert(!StackEmpty(pst));

    return pst->a[pst->top - 1];

}

bool StackEmpty(Stack\* pst)

{

    assert(pst);

    return pst->top == 0;

}

int StackSize(Stack\* pst)

{

    assert(pst);

    return pst->top;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

typedef struct {

    Stack pushST;

    Stack popST;

} MyQueue;

/\*\* Initialize your data structure here. \*/

MyQueue\* myQueueCreate() {

    MyQueue\* q = (MyQueue\*)malloc(sizeof(MyQueue));

    StackInit(&q->pushST);

    StackInit(&q->popST);

    return q;

}

/\*\* Push element x to the back of queue. \*/

void myQueuePush(MyQueue\* obj, int x) {

    StackPush(&obj->pushST,x);

}

/\*\* Removes the element from in front of queue and returns that element. \*/

int myQueuePop(MyQueue\* obj) {

    /\*if(StackEmpty(&obj->popST))

    {

        while(!StackEmpty(&obj->pushST))

        {

            StackPush(&obj->popST,StackTop(&obj->pushST));

            StackPop(&obj->pushST);

        }

    }\*/

    //此处可以接口复用

    int top = myQueuePeek(obj);

    //int top = StackTop(&obj->popST);

    StackPop(&obj->popST);

    return top;

}

/\*\* Get the front element. \*/

int myQueuePeek(MyQueue\* obj) {

    if(StackEmpty(&obj->popST))

    {

        while(!StackEmpty(&obj->pushST))

        {

            StackPush(&obj->popST,StackTop(&obj->pushST));

            StackPop(&obj->pushST);

        }

    }

    return StackTop(&obj->popST);

}

/\*\* Returns whether the queue is empty. \*/

bool myQueueEmpty(MyQueue\* obj) {

    return StackEmpty(&obj->pushST) && StackEmpty(&obj->popST);

}

void myQueueFree(MyQueue\* obj) {

    StackDestory(&obj->pushST);

    StackDestory(&obj->popST);

    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);

\*/