

思路：

先把数据入在一个队列中，出数据的时候把出的数据放在另一个队列中，只剩下最后一个数据是时候再Pop。同理另外一个队列也是一样，来回导（本题未要求时间和空间）。

同时入数据的时候往不为空的队列入，时刻保持另一个队列为空。

代码：

typedef int QDataType;

typedef struct QueueNode

{

    struct QueueNode\* next;

    QDataType data;

}QueueNode;

typedef struct Queue

{

    QueueNode\* head;

    QueueNode\* tail;

}Queue;

void QueueInit(Queue\* pq);

void QueueDestory(Queue\* pq);

void QueuePush(Queue\* pq, QDataType x);

void QueuePop(Queue\* pq);

QDataType QueueFront(Queue\* pq);

QDataType QueueBack(Queue\* pq);

bool QueueEmpty(Queue\* pq);

int QueueSize(Queue\* pq);

void QueueInit(Queue\* pq)

{

    assert(pq);

    pq->head = NULL;

    pq->tail = NULL;

}

void QueueDestory(Queue\* pq)

{

    assert(pq);

    QueueNode\* cur = pq->head;

    while (cur)

    {

        QueueNode\* next = cur->next;

        free(cur);

        cur = next;

    }

    pq->head = pq->tail = NULL;

}

void QueuePush(Queue\* pq, QDataType x)

{

    assert(pq);

    QueueNode\* newnode = (QueueNode\*)malloc(sizeof(QueueNode));

    if (newnode == NULL)

    {

        printf("malloc fail\n");

        exit(-1);

    }

    newnode->data = x;

    newnode->next = NULL;

    if (pq->tail == NULL)

    {

        pq->head = pq->tail = newnode;

    }

    else

    {

        pq->tail->next = newnode;

        pq->tail = newnode;

    }

}

void QueuePop(Queue\* pq)

{

    assert(pq);

    assert(!QueueEmpty(pq));  //防止队列为空

    if (pq->head->next == NULL)

    {

        free(pq->head);

        pq->head = pq->tail = NULL;

    }

    else

    {

        QueueNode\* next = pq->head->next;

        free(pq->head);

        pq->head = next;

    }

}

QDataType QueueFront(Queue\* pq)  //取队头数据

{

    assert(pq);

    assert(!QueueEmpty(pq));

    return pq->head->data;

}

QDataType QueueBack(Queue\* pq)  //取队尾数据

{

    assert(pq);

    assert(!QueueEmpty(pq));

    return pq->tail->data;

}

bool QueueEmpty(Queue\* pq) //bool类型为真返回0,为假返回1

{

    assert(pq);

    return pq->head == NULL;

}

int QueueSize(Queue\* pq)

{

    assert(pq);

    int size = 0;

    QueueNode\* cur = pq->head;

    while (cur)

    {

        ++size;

        cur = cur->next;

    }

    return size;

}

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

typedef struct {

    Queue q1;

    Queue q2;

} MyStack;

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

MyStack\* myStackCreate() {

    MyStack\* pst = (MyStack\*)malloc(sizeof(MyStack));

    QueueInit(&pst->q1);

    QueueInit(&pst->q2);

    return pst;

}

/\*\* Push element x onto stack. \*/

void myStackPush(MyStack\* obj, int x) {

    if(!QueueEmpty(&obj->q1))

    {

        QueuePush(&obj->q1,x);

    }

    else

    {

        QueuePush(&obj->q2,x);

    }

}

/\*\* Removes the element on top of the stack and returns that element. \*/

int myStackPop(MyStack\* obj) {

    Queue\* pEmpty = &obj->q1;

    Queue\* pNonEmpty = &obj->q2;

    if(!QueueEmpty(&obj->q1))

    {

        Queue\* pEmpty = &obj->q2;

        Queue\* pNonEmpty = &obj->q1;

    }

    while(QueueSize(pNonEmpty)>1)

    {

        QueuePush(pEmpty,QueueFront(pNonEmpty));

        QueuePop(pNonEmpty);

    }

    int front = QueueFront(pNonEmpty);

    QueuePop(pNonEmpty);

    return front;

}

/\*\* Get the top element. \*/

int myStackTop(MyStack\* obj) {

    if(!QueueEmpty(&obj->q1))

    {

        return  QueueBack(&obj->q1);

    }

    else

    {

        return  QueueBack(&obj->q2);

    }

}

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

bool myStackEmpty(MyStack\* obj) {

    return QueueEmpty(&obj->q1) && QueueEmpty(&obj->q2);

}

void myStackFree(MyStack\* obj) {

    QueueDestory(&obj->q1);

    QueueDestory(&obj->q2);

    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);

\*/

测试用例分析：

输入：

["MyStack", "push", "push", "top", "pop", "empty"] //这里表示的就是调用的是哪个接口函数

[[], [1], [2], [], [], []] //表示向接口函数传的参数值

输出：

[null, null, null, 2, 2, false] //表示接口函数的返回值或者输出结果

解释：

MyStack myStack = new MyStack();

myStack.push(1);

myStack.push(2);

myStack.top(); // 返回 2

myStack.pop(); // 返回 2

myStack.empty(); // 返回 False