顺序栈基本操作

文件声明

```
#include<stdio.h>
#include<stdlib.h>

#define MaxSize 50

//顺序栈定义
typedef struct{
    int data[MaxSize];  //存放栈中元素
    int top;            //栈顶指针
} SqStack;

//链栈定义
typedef struct Linknode{
    int data;
    struct Linknode *next;
}*LiStack;
```

栈的初始化

```
//栈的初始化
void InitStack(SqStack &S){
    S.top =  -1;        //初始化栈顶指针
}
```

判断栈是否为空

```
//判断栈是否为空
bool StackEmpty(SqStack S){
    if(S.top == -1){
        return true;    //栈为空返回true
    }
    else return false;  //栈非空，返回false
}
```

进栈

```
//进栈
bool Push(SqStack &S, int x){
    if(S.top = MaxSize - 1){
        return true;
    }
    else{
        S.data[++S.top] = x;    //指针+1，然后进栈
    }
}
```

出栈

```
bool Pop(SqStack &S, int &x){
    if(S.top == -1){
        return false;
    }
    else{
        x = S.data[S.top--];    //出栈后，指针减1
        return true;
    }
}
```

读栈顶元素

```
//读栈顶元素
bool GetTop(SqStack S, int &x){
    if(S.top == -1){
        return false;
    }
    else{
        x = S.data[S.top];
    }
}
```

循环队列基本操作

前面文件声明

```
#include<stdio.h>
#include<stdlib.h>

#define MaxSize 50
//队列的顺序存储
typedef struct
{
    int data[MaxSize];      //存放队列元素
    //队头指针允许删除，队尾指针允许插入
    int front, rear;        //队头指针和队尾指针
}SeQueue;
```

判断队空，队满

```
//顺序队列容易出现假溢出的情况，所以采用循环队列
/*循环队列区分队列满还是空

    1、牺牲一个存储单元
    2、结构体中增加一个数据元素size，存储队列中元素个数
    3、类型中，增加tag，tag=0，为空。若因删除导致队头和队尾相同，则为队空，若因插入导致队头与队尾相同，则为队蛮
*/
```

初始化

```
//初始化
void InitQueue(SeQueue &Q){
    Q.rear = Q.front = 0;
}
```

判断队是否为空

```
//判队空
bool isEmpty(SeQueue Q){
    if(Q.rear == Q.front) return true;
    else return false;
}
```

入队

```
//入队
bool EnQueue(SeQueue &Q, int x){
    //队满报错
    if((Q.rear+1)%MaxSize == Q.front){
        return false;
    }
    Q.data[Q.rear] = x;
    //队尾指针加1取模
    Q.rear = (Q.rear+1)%MaxSize;
    return true;
}
```

出队

```
//出队
bool Deqeue(SeQueue &Q,int &x){
    if(Q.rear == Q.front)  return false;
    x = Q.data[Q.front];
    Q.front = (Q.front+1)%MaxSize;
    return true;
}
```

链栈基本操作

前面文件声明

```
#include<stdio.h>
#include<stdlib.h>

#define MaxSize 50

//队列的链式存储，采用带头结点
typedef struct LinkNode{
    int data;
    struct LinkNode *next;
}LinkNode;
typedef struct{
    LinkNode *front,*rear;      //存放队列的队头和队尾
}LinkQueue;
```

链队的初始化

```
//链的初始化
void InitQueue(LinkQueue &Q){
    Q.front=Q.rear=(LinkNode *)malloc(sizeof(LinkNode));        //建立头结点
    Q.front->next = NULL;                                       //初始为空
}
```

判断队是否为空

```
//判队空
bool IsEmpty(LinkQueue Q){
    if(Q.front == Q.rear) return true;
    else return false;
}
```

入队

```
//入队
void EnQueue(LinkQueue &Q, int x){
    LinkNode *s = (LinkNode *)malloc(sizeof(LinkNode));
    //把节点插到队尾，然后移动队尾到s
    s->data = x;
    s -> next = Q.rear->next;
    Q.rear -> next = s;
    Q.rear = s;
}
```

出队

```
//出队
bool DeQueue(LinkQueue &Q, int &x){
    if(Q.front == Q.rear)  return false;          //如果队为空，报错
    //这里采用带头结点的链表，所以需要找到下一个节点，才是带值节点
    LinkNode *p = Q.front->next;
    x = p->data;
    Q.front ->next = p->next;
    if(Q.rear == p){
        Q.rear = Q.front;       //如果原始队列中，只有一个元素，删除后变为空
    }
    free(p);
    return true;

}
```