

Lecture 7: 栈基本概念

7.1 栈的定义

概念：栈

- 定义：
 - 只允许一段进入插入或删除操作的线性表；
- 解释：
 - 栈顶 (Top)
 - 栈底 (Button)
 - 空栈
- 栈的基本操作
 - `InitStack(&S)`: 初始化栈。构造一个空栈 S, 分配内存空间
 - `DestroyStack(&L)`: 销毁栈。销毁并释放栈 S 所占用的内存空间
 - `Push(&S, x)`: 进栈, 若栈 S 未满, 则将 x 加入使之成为新栈顶
 - `Pop(&S, &x)`: 出栈, 若栈 S 非空, 则弹出栈顶元素, 并用 x 返回
 - `GetTop(S, &x)`: 读栈顶元素。若栈 S 非空, 则用 x 返回栈顶元素
- 总结
 - 创, 消: `InitStack DestroyStack`
 - 增, 删: `Push Pop`
 - 查: `GetTop`
 - `StackEmpty (S)`: 判断一个栈 S 是否为空。若 S 为空, 则返回 true, 否则返回 false;
- 补充: 卡特兰数
 - n 个不同元素进栈, 出栈元素不同排列的个数为: $\frac{1}{n+1}C_{2n}^n$
 - 上述公式称为卡特兰 (Catalan) 数, 可采用数学归纳法证明;

7.2 栈的顺序存储实现

7.2.1 顺序栈定义

定义

- 和顺序表的定义类似

代码定义

```
//顺序栈的定义
#define MaxSize 10          //定义栈中元素的最大个数
typedef struct{
    ElemType data[MaxSize]; //静态数组存放栈中元素
    int top;                //栈顶指针
}SqStack; //Sq:sequence -- 顺序
```

初始化

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

//下面是初始化声明
void testStack(){
    SqStack S;           //声明一个顺序栈(分配空间)
    InitStack(S);
    //...后续操作
    //函数运行结束后，系统自动回收空间(不需理会)
}
```

7.2.2 顺序栈基本操作

判断栈空

- 最开始栈中无元素时，`s.top=-1`，表示当前无元素，所以栈顶在数组中的下表为-1（无）；
- 如果 `s.top` 的初始值设置为 `s.top=0`，则表示初始时栈顶指针指向 `a[0]` 的位置，当压入一个数据时，栈顶指针指向 `a[1]` 的位置、栈满的条件为 `s.top==MaxSize`；

```
//判断栈空
bool StackEmpty(SqStack S){
    if(S.top == -1) //栈空
        return true;
    else           //不空
        return false;
}
```

出入栈

```
//新元素进栈操作
bool Push(SqStack &S, ElemType x){
    if(S.top == MaxSize-1) //栈满，报错
        return false;
    S.top = S.top + 1;      //指针先加1
    S.data[S.top] = x;      //新元素入栈
    return true;
}
```

- 注意：等价写法 `S.data[++S.top]=x`，其中的 `++号` 不能写反；

```
//出栈操作
bool Pop(SqStack &S, ElemType &x){
    if(S.top == -1) //栈空，报错
        return false;
    x=S.data[S.top]; //栈顶元素先出栈
```

```

    S.top = S.top - 1; //指针再减1
    return true;
}

```

- 注意: 此时数据还存留在内存中, 知识逻辑上被删除了 (没有 free);

读取栈顶元素

```

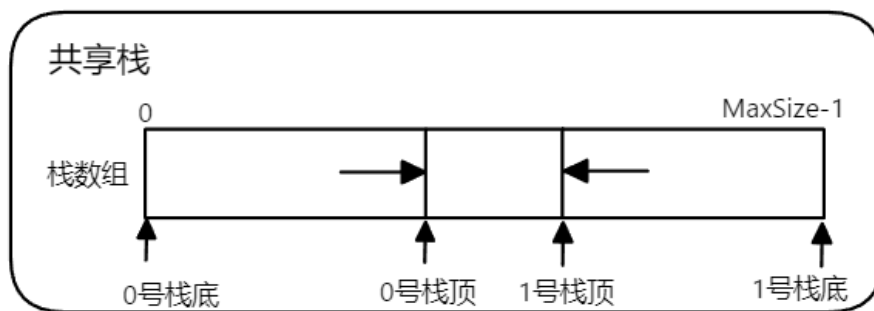
//读栈顶元素
bool GetTop(SqStack S, ElemType &x){
    if(S.top == -1) //栈空, 报错
        return false;
    x=S.data[S.top]; //x记录栈顶元素
    return true;
}

```

7.2.3 共享栈

共享栈

- 图示:



- 代码:

```

//共享栈的定义
#define MaxSize 10
typedef struct{
    ElemType data[MaxSize]; //静态数组存放栈中元素
    int top0; //0号栈顶指针
    int top1; //1号栈顶指针
}SqStack;

//初始化栈
void InitStack(SqStack &S){
    S.top0 = -1; //初始化栈顶指针
    S.top1 = MaxSize;
}

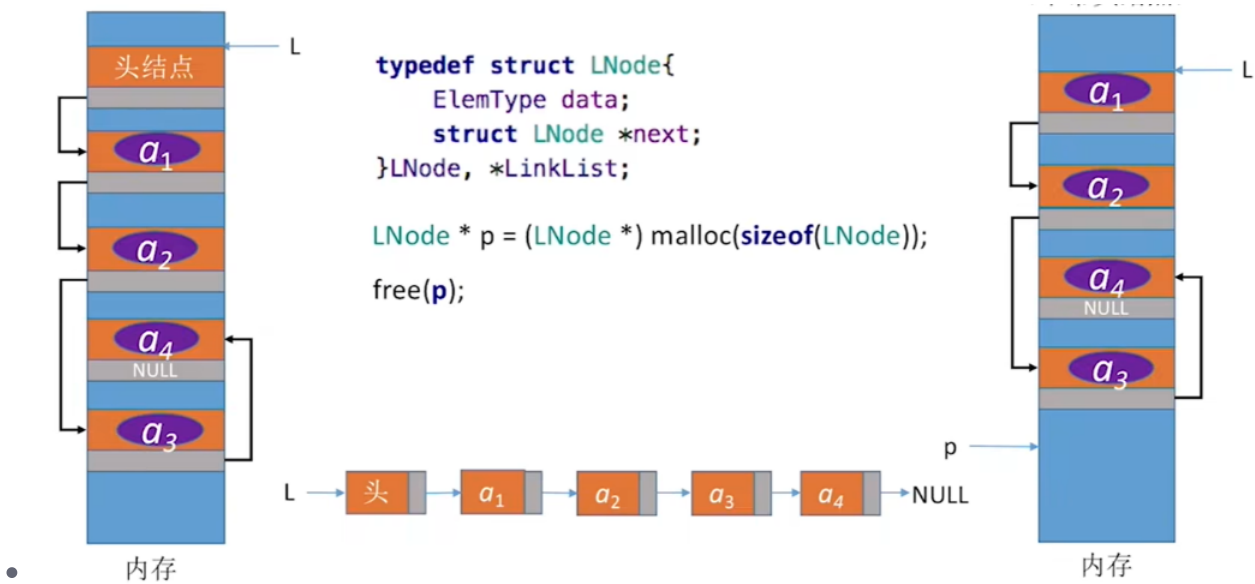
```

- 分析:
 - 栈满的条件: `top0 + 1 == top1`
- 特点:
 - 实现了两个栈, 但是却共享了一个存储空间, 提高资源利用率;

7.3 栈的链式存储实现

概念：栈的链式存储实现

- 概念：
 - 和单链表的定义几乎没有区别，名词变化；
- 图示：



定义

```
typedef struct Linknode{
    ElemType data;           //数据域
    struct Linknode *next;   //指针域
}*LiStack;                  //栈类型定义
```

面试题：栈和队列的区别

- 队列：
 - 是允许在一段进行插入另一端进行删除的线性表；
 - 队列顾名思义就像排队一样，对于进入队列的元素按“先进先出”的规则处理，在表头进行删除在表尾进行插入；
 - 由于队列要进行频繁的插入和删除，一般为了高效，选择用定长数组来存储队列元素，在对队列进行操作之前要判断队列是否为空或是否已满；
 - 如果想要动态长度也可以用链表来存储队列，这时要记住队头和队尾指针的地址；
- 栈：
 - 是只能在表尾进行插入和删除操作的线性表；
 - 对于插入到栈的元素按“后进先出”的规则处理，插入和删除操作都在栈顶进行，与队列类似一般用定长数组存储栈元素；
 - 由于进栈和出栈都是在栈顶进行，因此要有一个 `size` 变量来记录当前栈的大小，当进栈时 `size` 不能超过数组长度，`size+1`. 出栈时栈不为空，`size-1`；

面试题：如何区分队列空还是栈满

- 普通情况下，循环队列队空和队满的判定条件是一样的，都是 `Q.front==Q.rear`；

- 队头指针指向第一个数：队尾指针指向最后一个数的下一个位置，即将要入队的位置；
- 方法一：牺牲一个单元来区分队空和队满；
 - 这个时候 `(Q.rear+1)%MaxSize==Q.front` 才是队满标志；
- 方法二：类型中增设表示元素个数的数据成员；
 - 这样，队空的条件为 `Q.size==0`：队满的条件为 `Q.size==MaxSize`

获取笔记持续更新可加微信：WandD2277