

栈

一、栈的基本概念

• 栈的定义

线性表是具有相同数据类型的 n ($n \geq 0$) 个数据元素的有限序列, 其中 n 为表长, 当 $n=0$ 时线性表示一个空表。若用 L 命名线性表, 则其一般表示为

$$L = (a_1, a_2, \dots, a_i, a_{i+1}, \dots, a_n)$$

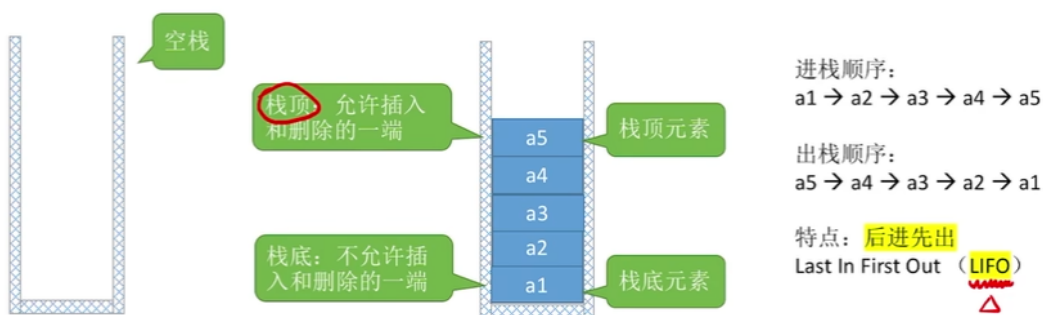
栈 (Stack) 是只允许在一端进行插入或删除操作的线性表

重要术语: 栈顶、栈底、空栈

栈 (Stack) 是只允许在一端进行插入或删除操作的线性表

重要术语: 栈顶、栈底、空栈

逻辑结构: 与普通线性表相同
数据的运算: 插入、删除操作有区别



• 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。

创、销

删除栈顶元素

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。

增、删

不删除栈顶元素

GetTop(S, &x): 读栈顶元素。若栈 S 非空, 则用 x 返回栈顶元素

查: 栈的使用场景中大多只访问栈顶元素

其他常用操作:

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。

• 栈的常考题型

进栈顺序：
a → b → c → d → e

有哪些合法的出栈顺序？

e, d, c, b, a
b, e, d, c, a



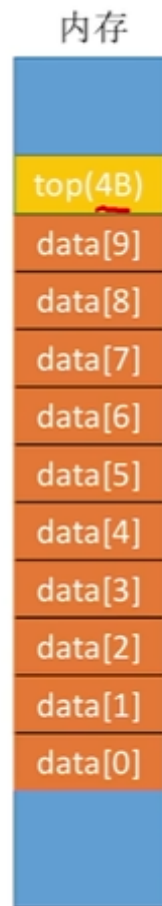
n个不同元素进栈，出栈元素不同排列的个数为 $\frac{1}{n+1}C_{2n}^n$ 。
上述公式称为卡特兰（Catalan）数，可采用数学归纳法证明（不要求掌握）。

$$\frac{1}{5+1}C_{10}^5 = \frac{10*9*8*7*6}{6*5*4*3*2*1} = 42$$

二、栈的顺序存储实现

• 顺序栈的定义

```
1  #define MaxSize 10           //定义栈中元素的最大个数
2  typedef struct
3  {
4      ElemType data[MaxSize]; //静态数组存放栈中元素
5      int top;                //栈顶指针
6  }SqStack; //Sq:sequence
7
8  void testStack()
9  {
10     SqStack s;               //声明一个顺序栈（分配空间）
11     //...后续操作...
12 }
```



顺序存储：给各个数据元素分配连续的存储空间，大小为 $\text{MaxSize} \times \text{sizeof}(\text{ElemType})$

哔哩哔哩

顺序栈的定义

```
#define MaxSize 10
typedef struct{
    ElemType data[MaxSize];
    int top;
} SqStack;
```

Sq: sequence —— 顺序

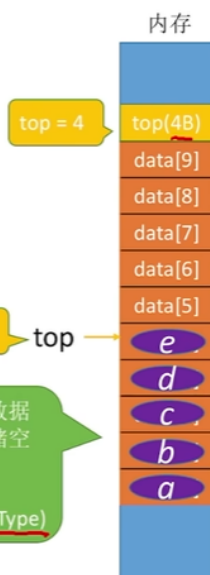
//定义栈中元素的最大个数

//静态数组存放栈中元素
//栈顶指针

```
void testStack() {
    SqStack S; //声明一个顺序栈(分配空间)
    //...后续操作...
}
```

top 指向栈顶元素

顺序存储：给各个数据元素分配连续的存储空间，大小为
 $\text{MaxSize} \times \text{sizeof}(\text{ElemType})$



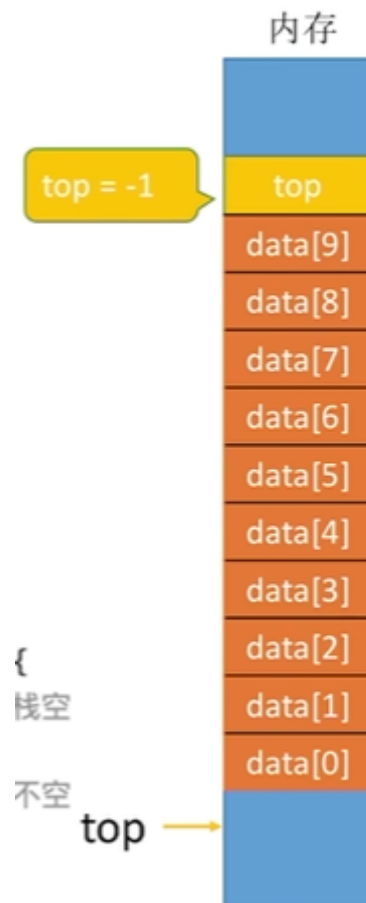
• 初始化操作

```
1  #define MaxSize 10           //定义栈中元素的最大个数
2  typedef struct
3  {
4      ElemType data[MaxSize]; //静态数组存放栈中元素
5      int top;                //栈顶指针
6  }SqStack; //Sq:sequence
7
8  void InitStack(SqStack &S) //初始化栈
9  {
```

```

10     S.top = -1;           //初始化栈顶指针
11 }
12
13 //判断栈空
14 bool StackEmpty(SqStack S)
15 {
16     if(S.top == -1)      //栈空
17         return true;
18     else
19         return false;   //不空
20 }
21
22 void testStack()
23 {
24     SqStack s;           //声明一个顺序栈（分配空间）
25     InitStack(s);
26     //...后续操作...
27 }

```



• 进栈操作

```

1  #define MaxSize 10           //定义栈中元素的最大个数
2  typedef struct
3  {
4      ElemType data[MaxSize]; //静态数组存放栈中元素
5      int top;                //栈顶指针
6  }SqStack;
7
8  //新元素入栈

```

```

9  bool Push(SqStack &S, ElemType x)
10 {
11     if(S.top == MaxSize - 1) //栈满，报错
12         return false;
13
14     S.top = S.top + 1;        //指针先加1
15     S.data[S.top] = x;       //新元素入栈
16     return true;
17 }
18 /*S.top = S.top + 1;
19    S.data[S.top] = x;
20    等价于
21    S.data[++S.top] = x;*/

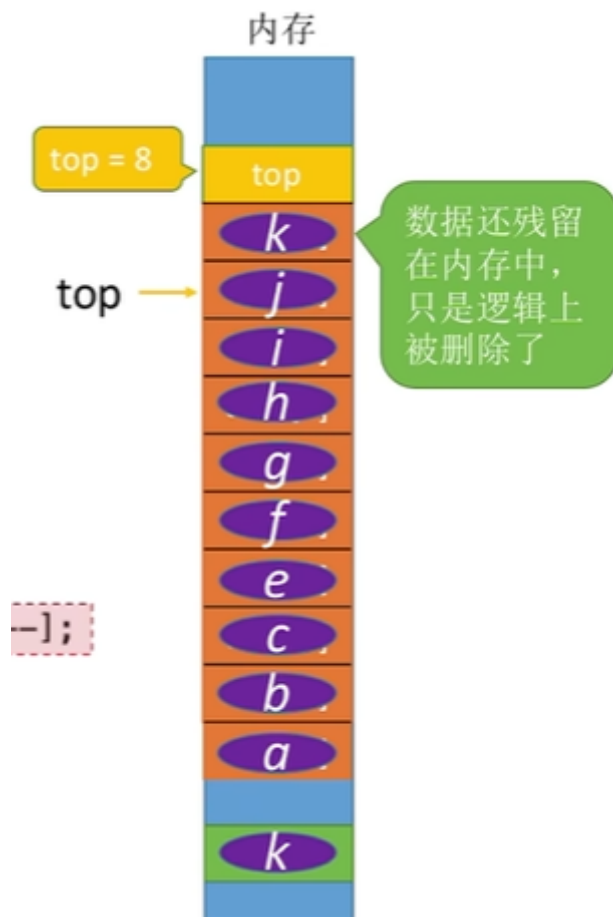
```

• 出栈操作

```

1  #define MaxSize 10           //定义栈中元素的最大个数
2  typedef struct
3  {
4      ElemType data[MaxSize]; //静态数组存放栈中元素
5      int top;                //栈顶指针
6  }SqStack;
7
8  //出栈操作
9  bool Pop(SqStack &S, ElemType &x)
10 {
11     if(S.top == -1)          //栈空，报错
12         return false;
13
14     x = S.data[S.top];       //栈顶元素先出栈
15     S.top = S.top - 1;       //指针再减1
16     return true;
17 }
18 /*x = S.data[S.top];
19    S.top = S.top - 1;
20    等价于
21    x = S.data[S.top--]*/

```



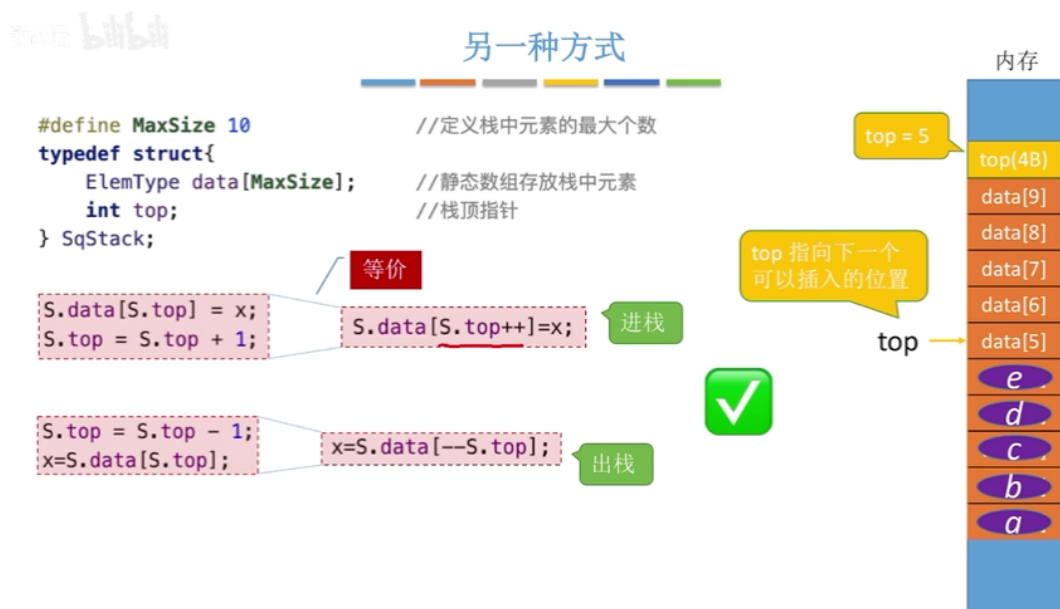
• 读栈顶元素操作

```

1  #define MaxSize 10           //定义栈中元素的最大个数
2  typedef struct
3  {
4      ElemType data[MaxSize]; //静态数组存放栈中元素
5      int top;                //栈顶指针
6  }SqStack;
7
8  //出栈操作
9  bool Pop(SqStack &S, ElemType &x)
10 {
11     if(S.top == -1)          //栈空，报错
12         return false;
13
14     x = S.data[S.top--];     //先出栈，指针再减1
15     return true;
16 }
17
18 //读栈顶元素
19 bool GetTop(SqStack S, ElemType &x)
20 {
21     if(S.top == -1)          //栈空，报错
22         return false;
23
24     x = S.data[S.top];       //x记录栈顶元素
25     return true;
26 }

```

• 另一种方式



栈满条件: $top == MaxSize$

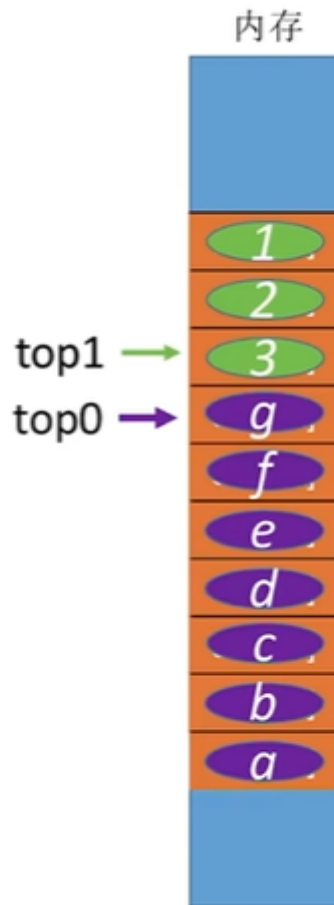
顺序栈的缺点: 栈的大小不可变

• 共享栈

```

1  #define MaxSize 10           //定义栈中元素的最大个数
2  typedef struct
3  {
4      ElemType data[MaxSize];  //静态数组存放栈中元素
5      int top0;                //0号栈顶指针
6      int top1;                //1号栈顶指针
7  } ShStack;
8
9  //初始化栈
10 void InitStack(ShStack &s)
11 {
12     s.top0 = -1;
13     s.top1 = MaxSize;
14 }

```



两个栈共享一片空间

栈满条件: $top0 + 1 == top1$

三、栈的链式存储实现

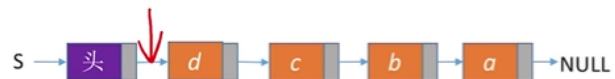
• 链栈的定义

```

1 typedef struct Linknode
2 {
3     ElemType data;           //数据域
4     struct Linknode *next;   //指针域
5 }*Listack;                   //栈类型定义

```

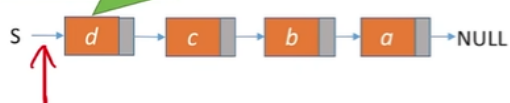
带头结点的初始化



进栈/出栈都只能在栈顶一端进行
(链头作为栈顶)

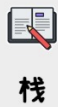
S → NULL

不带头结点的初始化



剩下的创、增、删、查、如何判空、判满和单链表类似

四、知识总结



栈

栈的基本概念

- 定义
 - 一种操作受限的线性表，只能在栈顶插入、删除
 - 特性：后进先出（LIFO）
 - 术语：栈顶、栈底、空栈
- 基本操作
 - 创、销
 - 增、删（元素进栈、出栈，只能在栈顶操作）
 - 查（获得栈顶元素，但不删除）
 - 判空

栈的顺序存储实现

- 顺序存储，用静态数组实现，并须记录栈顶指针
- 基本操作
 - 创、增、删、查（都是 $O(1)$ 时间复杂度）
- 两种实现
 - 初始化时 $top = -1$
 - 入栈 $S.data[++S.top] = x;$
 - 出栈 $x = S.data[S.top--];$
 - 获得栈顶元素 $x = S.data[S.top];$
 - 栈空/栈满条件
 - 初始化时 $top = 0$
 - 入栈 $S.data[S.top++] = x;$
 - 出栈 $x = S.data[--S.top];$
 - 获得栈顶元素 $x = S.data[S.top - 1];$
 - 栈空/栈满条件
- 共享栈
 - 两个站共享同一内存空间，两个栈从两边往中间增长
 - 初始化
 - 0号栈栈顶指针初始时 $top0 = -1$ ；1号栈栈顶指针初始时 $top1 = MaxSize$;
 - 栈满条件 $top0 + 1 == top1;$

栈的链式存储实现

- 用链表存储方式实现的栈
- 两种实现方式
 - 带头结点
 - 不带头结点（推荐）
- 重要的基本操作
 - 创（初始化）
 - 增（进栈）
 - 删（出栈）
 - 查（获取栈顶元素）
 - 如何判空、判满