第3章 栈和队列

教学内容

3.1 栈和队列的定义和特点

3.2 案例引入

3.3 栈的表示和操作的实现

3.4 栈与递归

3.5 队列的的表示和操作的实现

3.6 案例分析与实现

第3章　栈和队列

教学目标

1. 掌握栈和队列的特点，并能在相应的应用问题中正确选用
2. 熟练掌握栈的两种存储结构的基本操作实现算法，特别应注意栈满和栈空的条件
3. 熟练掌握循环队列和链队列的基本操作实现算法，特别注意队满和队空的条件
4. 理解递归算法执行过程中栈的状态变化过程
5. 掌握表达式求值 方法

栈（Stack）

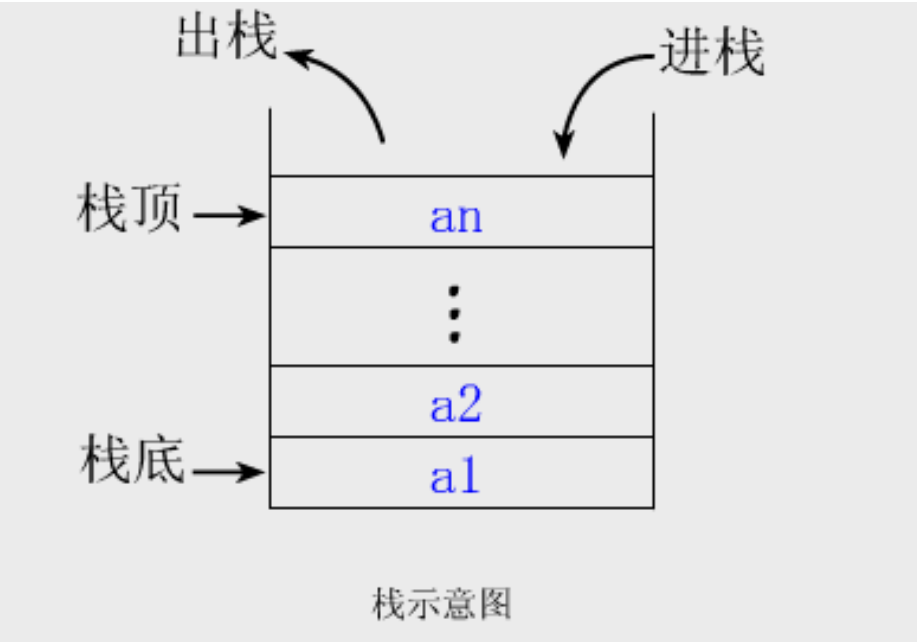
1. 定义
2. 逻辑结构
3. 存储结构
4. 运算规则
5. 实现方式

队列（Queue）

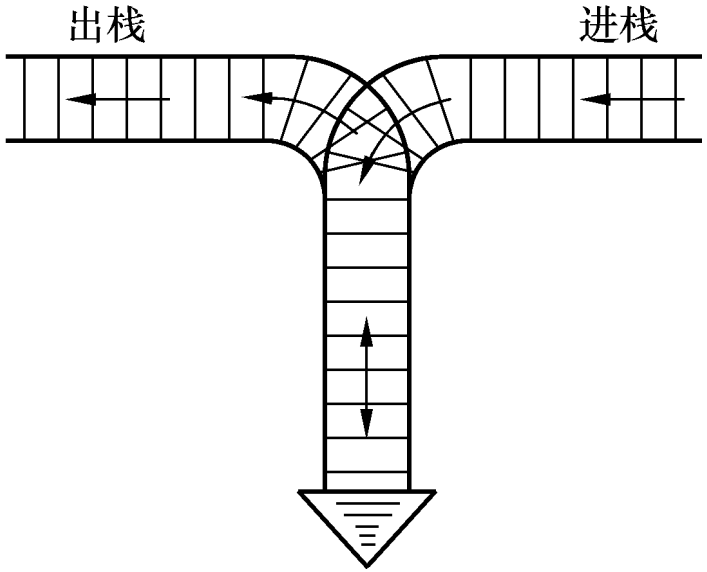
1. 定义
2. 逻辑结构
3. 存储结构
4. 运算规则
5. 实现方式

2020年4月13日

栈



栈示意图



用铁路调度站表示栈

3.1 栈和队列的定义和特点

栈

1. 定义 只能在表的一端（栈顶）进行插入和删除运算的线性表

2. 逻辑结构 与线性表相同，仍为一对一关系

3. 存储结构 用顺序栈或链栈存储均可，但以顺序栈更常见

4.运算规则

只能在**栈顶**运算，且访问结点时依照**后进先出（LIFO）**或**先进后出（FILO）**的原则

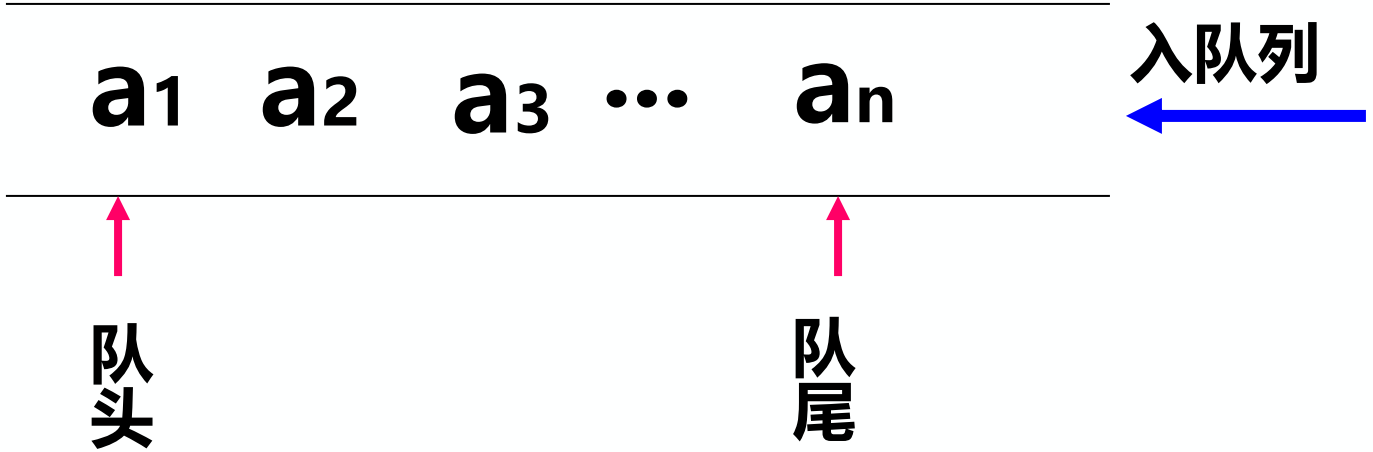
5.实现方式

关键是编写**入栈**和**出栈**函数，具体实现依顺序栈或链栈的不同而不同

基本操作有**入栈、出栈、读栈顶元素值、建栈、判断栈满、栈空**等

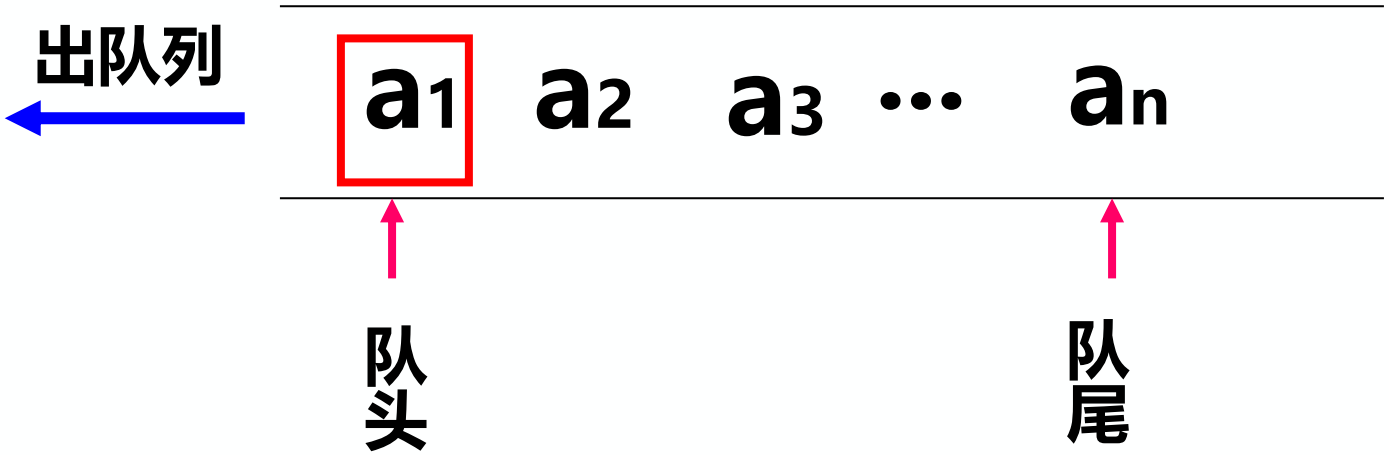
队列是一种先进先出(FIFO) 的线性表. 在表一端插入,在另一端删除

$$q=(a_1,a_2,\cdots a_n)$$

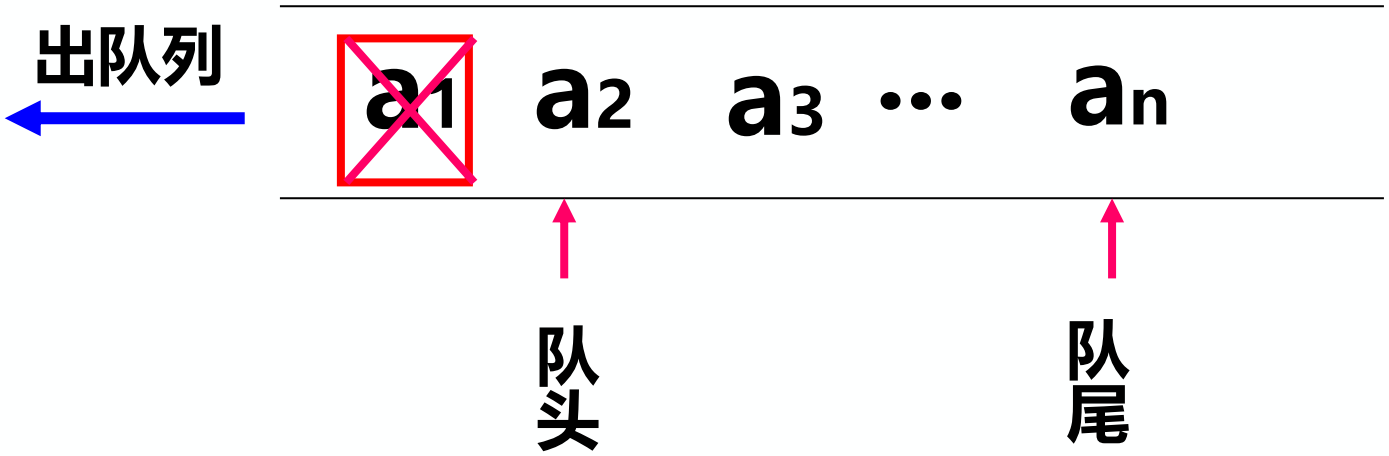


$$q=(a_1,a_2,\cdots a_n)$$

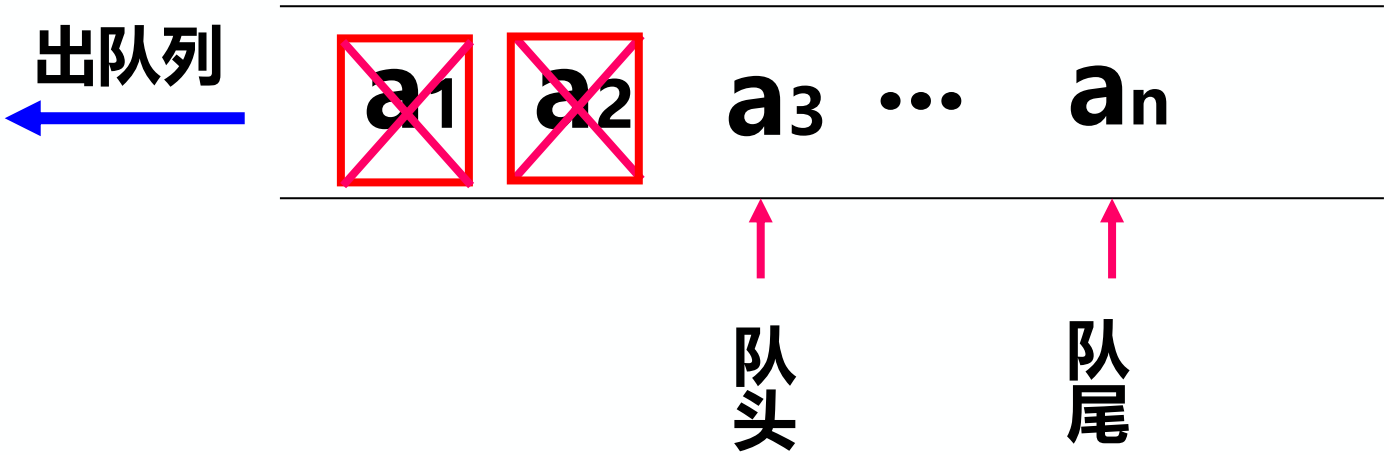
出队列



$$q=(a_1,a_2,\cdots a_n)$$



$$q=(a_1,a_2,\cdots a_n)$$



3.1 栈和队列的定义和特点

队列

1. 定义 只能在表的一端（**队尾**）进行插入，在另一端（**队头**）进行删除运算的**线性表**

2. 逻辑结构 与线性表相同，仍为**一对一**关系

3. 存储结构 用**顺序队列**或**链队**存储均可

4.运算规则 先进先出（FIFO）

5.实现方式 关键是编写入队和出队函数，具体实现依顺序队或链队的不同而不同

栈、队列与一般线性表的区别

栈、队列是一种特殊（操作受限）的线性表

区别：仅在于运算规则不同

一般线性表

逻辑结构：一对一

存储结构：顺序表、链表

运算规则：随机、顺序存取

栈

逻辑结构：一对一

存储结构：顺序栈、链栈

运算规则：后进先出

队列

逻辑结构：一对一

存储结构：顺序队、链队

运算规则：先进先出

3.2 案例引入

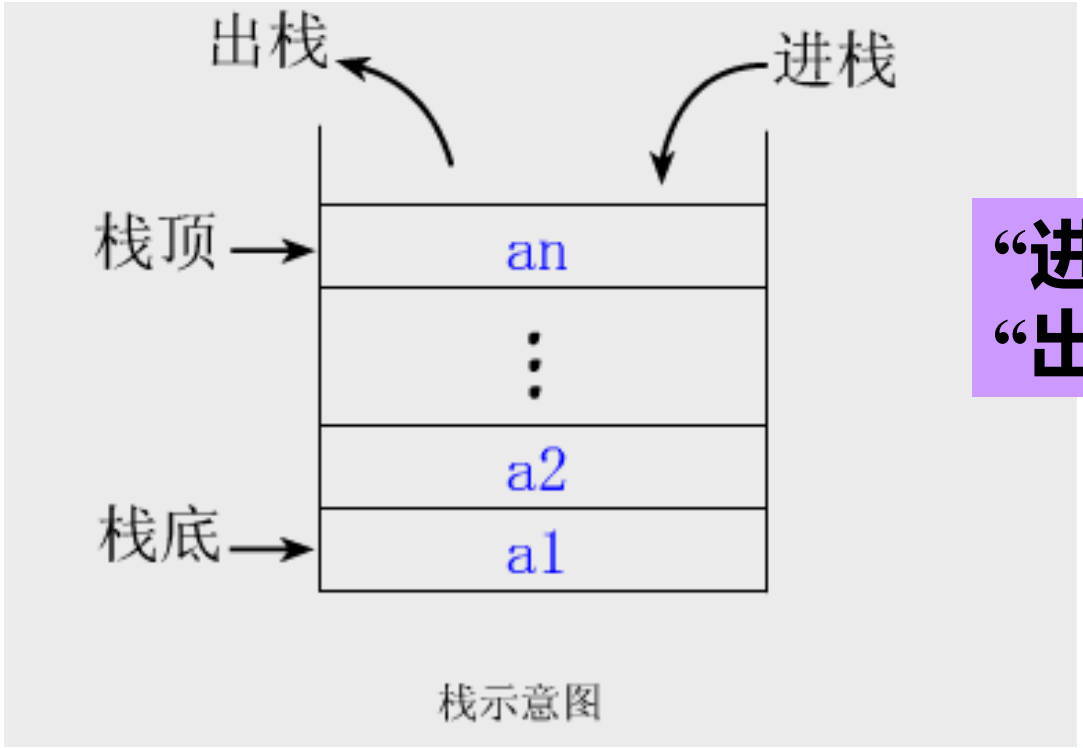
案例3.1： 一元多项式的运算

案例3.2：括号匹配的检验

案例3.3： 表达式求值

案例3.4： 舞伴问题

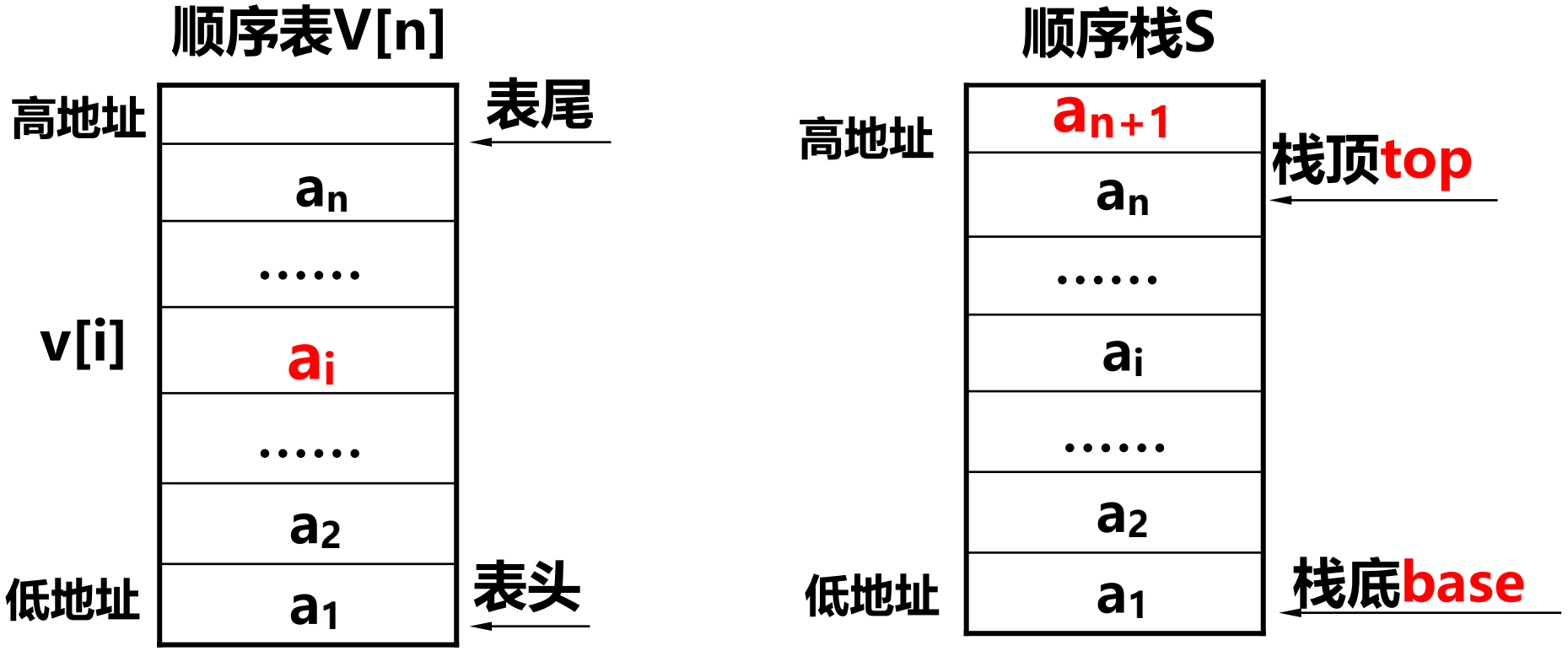
3.3 栈的表示和操作的实现



栈示意图

“进” = 压入=PUSH（）
“出” = 弹出=POP( )

顺序栈与顺序表



写入：v[i]= a_i

读出： x= v[i]

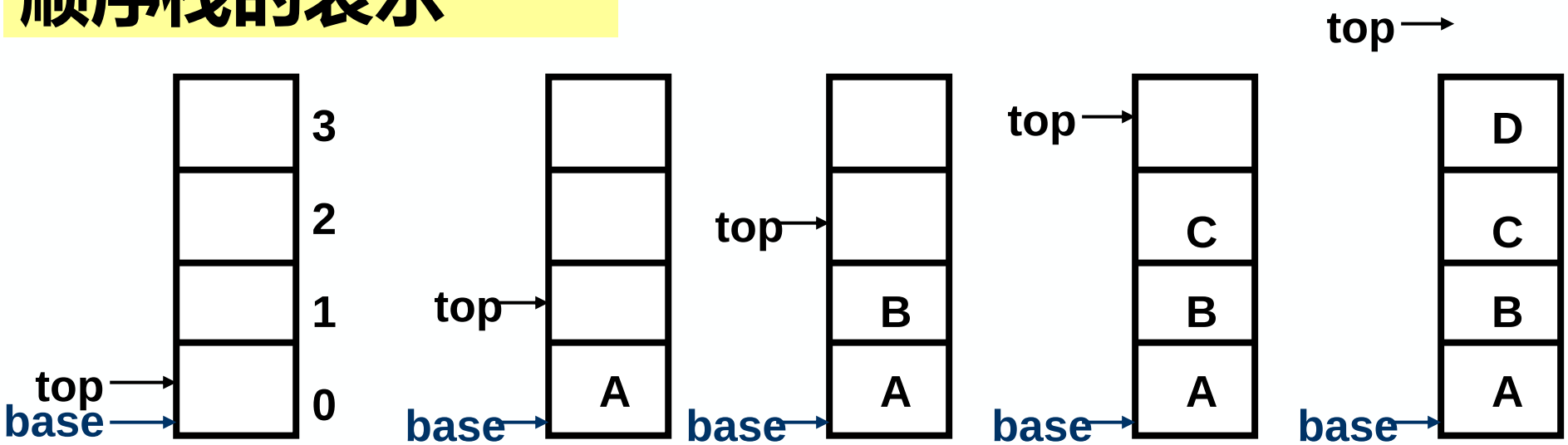
压入：PUSH (a_{n+1})

弹出： POP (x)

前提：一定要预设栈顶指针top！

2020年4月13日

顺序栈的表示



空栈

base == top

是栈空标志

stacksize = 4

top 指示真正的**栈顶元素之上**的下标地址

栈满时的处理方法：

1、**报错**,返回操作系统。

2、**分配更大的空间**，作为栈的存储空间,将原栈的内容移入新栈。

顺序栈的表示

```
#define  MAXSIZE  100
typedef struct
{
        SElemType   *base;
        SElemType   *top;
        int stacksize;
}SqStack;
```

顺序栈初始化

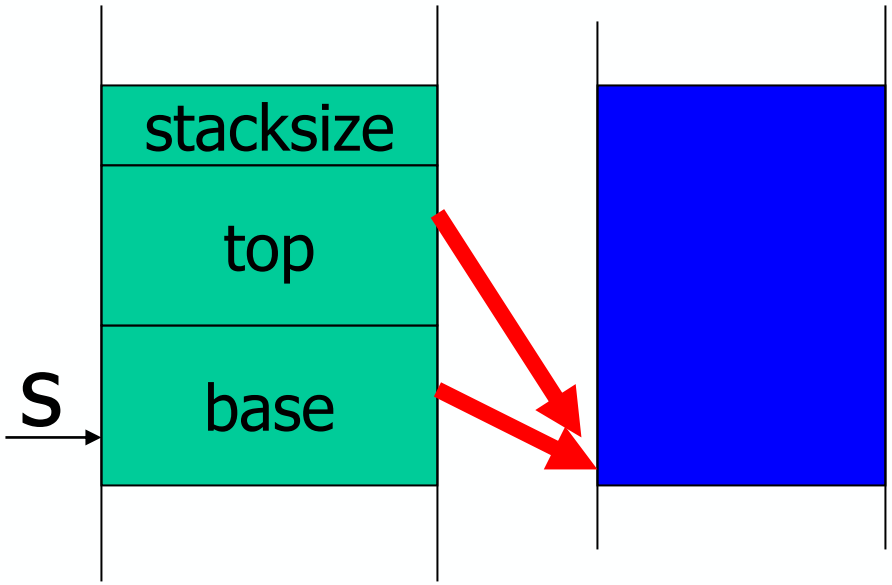
- 构造一个空栈
- 步骤：

(1)分配空间并检查空间是否分配失败，若失败则返回错误

(2)设置栈底和栈顶指针

S.top = S.base;

(3)设置栈大小

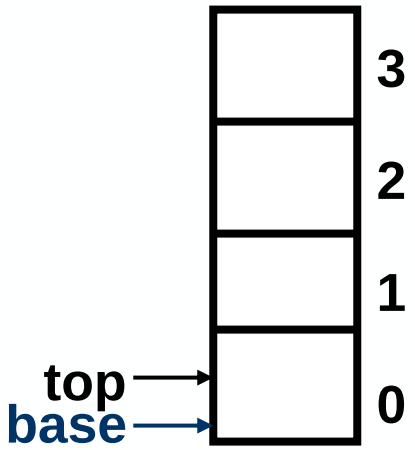


顺序栈初始化

```
Status InitStack( SqStack &S )
{
    S.base =new SElemType[MAXSIZE];
    if( !S.base )        return OVERFLOW;
    S.top = S.base;
    S.stackSize = MAXSIZE;
    return OK;
}
```

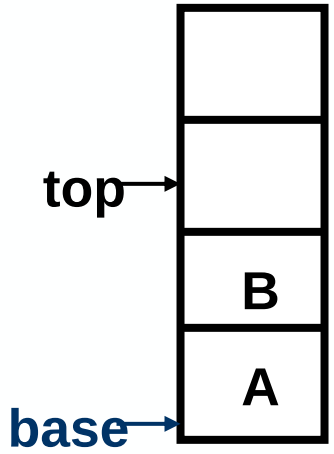
判断顺序栈是否为空

```
bool StackEmpty( SqStack S )
{
   if(S.top == S.base) return true;
   else return false;
}
```



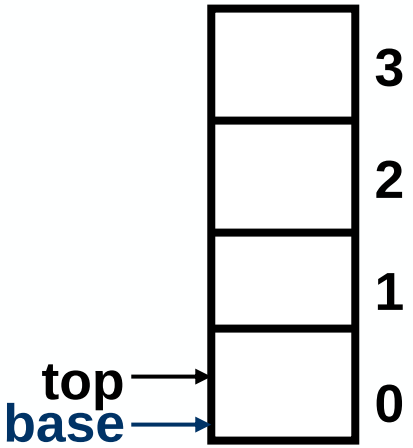
求顺序栈的长度

```
int StackLength( SqStack S )
{
    return S.top – S.base;
}
```



清空顺序栈

```
Status ClearStack( SqStack &S )
{
    if( S.base ) S.top = S.base;
    return OK;
}
```

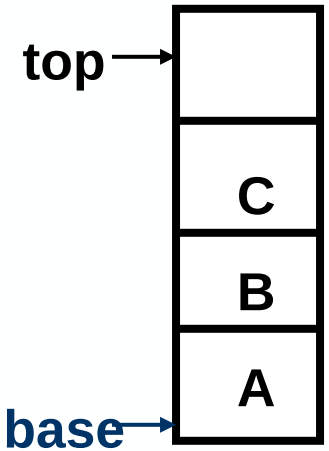


销毁顺序栈

```
Status DestroyStack( SqStack &S )
{
   if( S.base )
   {
        delete S.base ;
        S.stacksize = 0;
        S.base = S.top = NULL;
   }
  return OK;
}
```

顺序栈进栈

(1)判断是否栈满，若满则出错
(2)元素e压入栈顶
(3)栈顶指针加1

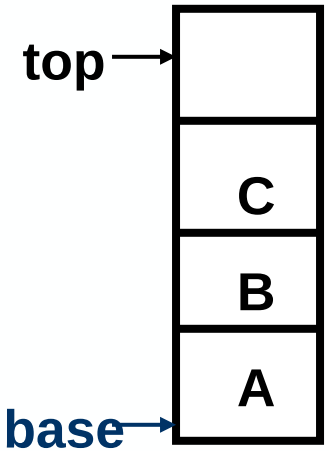



```
Status Push( SqStack &S, SElemType e)
{
   if( S.top - S.base== S.stacksize - 1 ) // 栈满
      return ERROR;
   *S.top++=e;
   return OK;
}
```

*S.top=e;
S.top++;

顺序栈出栈

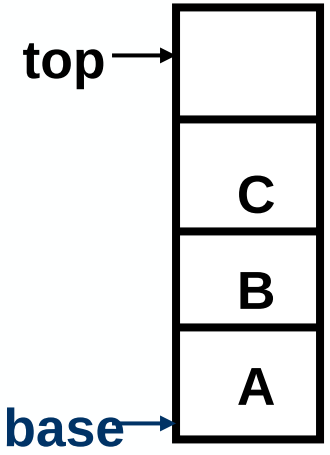
(1)判断是否栈空，若空则出错
(2)获取栈顶元素e
(3)栈顶指针减1




```
Status Pop( SqStack &S, SElemType &e)
{
    if( S.top == S.base) // 栈空
        return ERROR;
    e =  *--S.top;
    return OK;
}
```

--s.top;
e=*S.top;

取顺序栈栈顶元素



(1)判断是否空栈，若空则返回错误
(2)否则通过栈顶指针获取栈顶元素

```
Status GetTop( SqStack S, SElemType &e)
{
   if( S.top == S.base )     return ERROR;  // 栈空
   e = *( S.top – 1 );
   return OK;
}
```

e = *(S.top --);
???

练习

1.如果一个栈的输入序列为123456，能否得到435612和135426的出栈序列?

435612中到了12顺序不能实现；

135426可以实现。