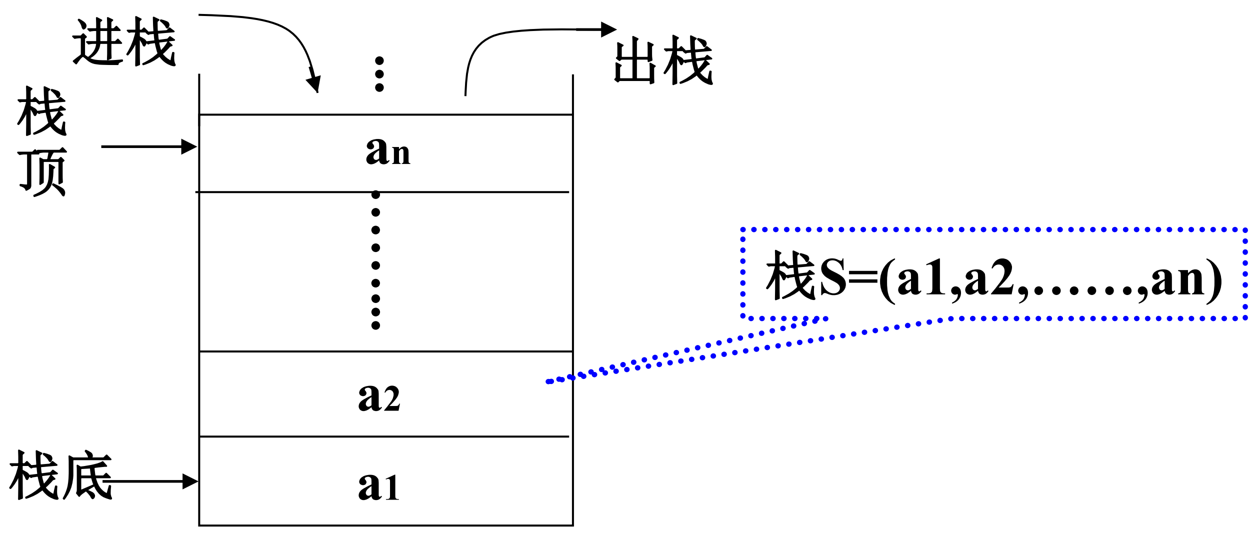
第3章 栈

3.1 ADT栈（stack）

1、栈的定义和特点

- 定义：限定仅在表首进行插入或删除操作的线性表，表首—栈顶，表尾—栈底，不含元素的空表称空栈
- 特点：先进后出（FILO）或后进先出（LIFO）



3.1 ADT栈(Stack)

2、ADT栈上定义的常用的基本运算：

(1) StackEmpty(S): 判断栈空

(2) StackFull(S):判断栈满

(3) StackTop(S): 返回栈顶元素

(4) Push(x, S):将元素x入栈

(5) Pop(S):出栈，删除并返回栈S的栈顶元素

3.1 ADT栈(Stack)

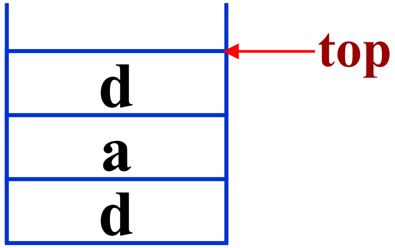
- **3、栈应用的简单例子：**

  - **(1)程序编译时的表达式或字符串的括号匹配问题。**

例如，算术表达式(x*(x+y)-z)，其中位置1和4处有左括号，而位置8和11处有右括号，满足配对要求。

但算术表达式 (x+y)*z)(，其中位置8处的右括号没有可与之配对的左括号，而位置9处的左括号没有可与之配对的右号。

(2)回文游戏：顺读与逆读字符串一样（不含空格）

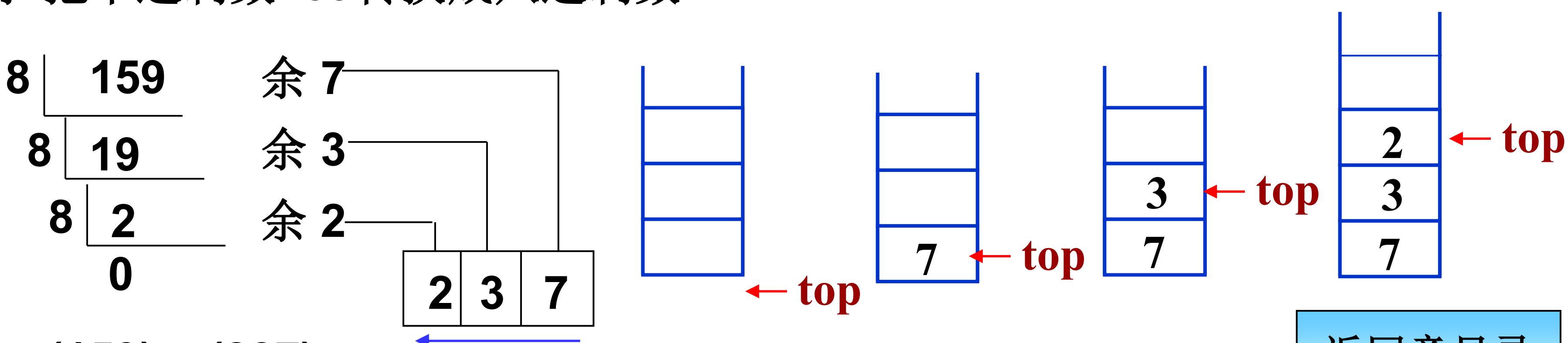


字符串：“madam im adam”

1.读入字符串
2.去掉空格（原串）
3.压入栈
4.原串字符与出栈字符依次比较
  若不等，非回文
  若直到栈空都相等，回文

(3)多进制输出：

例 把十进制数159转换成八进制数

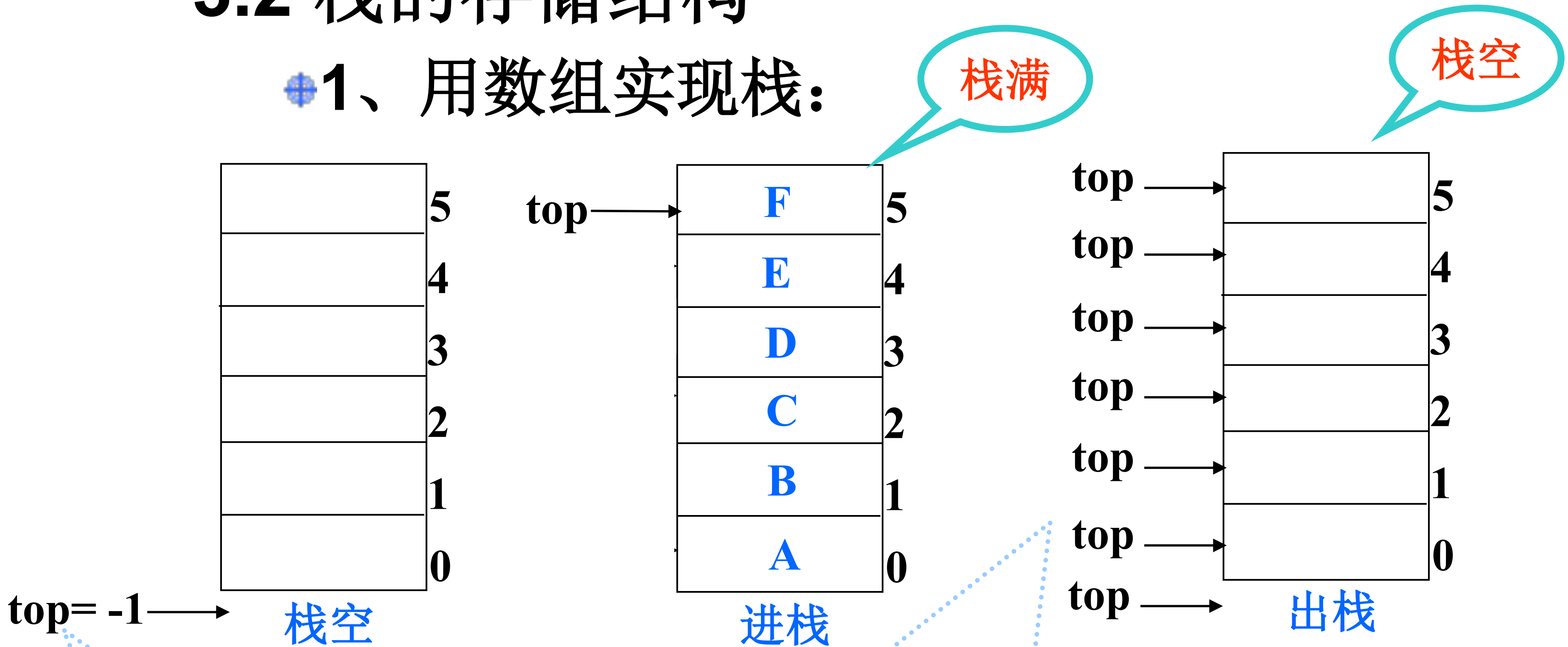


$(159)_{10}=(237)_{8}$

返回章目录

3.2 栈的存储结构

- 1、用数组实现栈：



栈顶指针top,指向实际栈顶后的空位置，初值为-1

设数组维数为M
top= -1,栈空，此时出栈，则下溢（underflow）
top=M-1,栈满，此时入栈，则上溢（overflow）

(1)用数组实现的栈结构Stack定义：

```
typedef  struct astack  *Stack;
typedef  struct astack {
    int  top；            /* 栈顶位置，当栈为空时，top=-1*/
    int  maxtop；         /*栈顶位置的最大值 */
    StackItem   *data;    /*栈元素数组*/
} Astack；
```

- 入栈算法

```
void Push(StackItem x, Stack S)
{
   if( StackFull(S) Error("Stack is full");
   else S->data[++ S->top] = x;
}
```

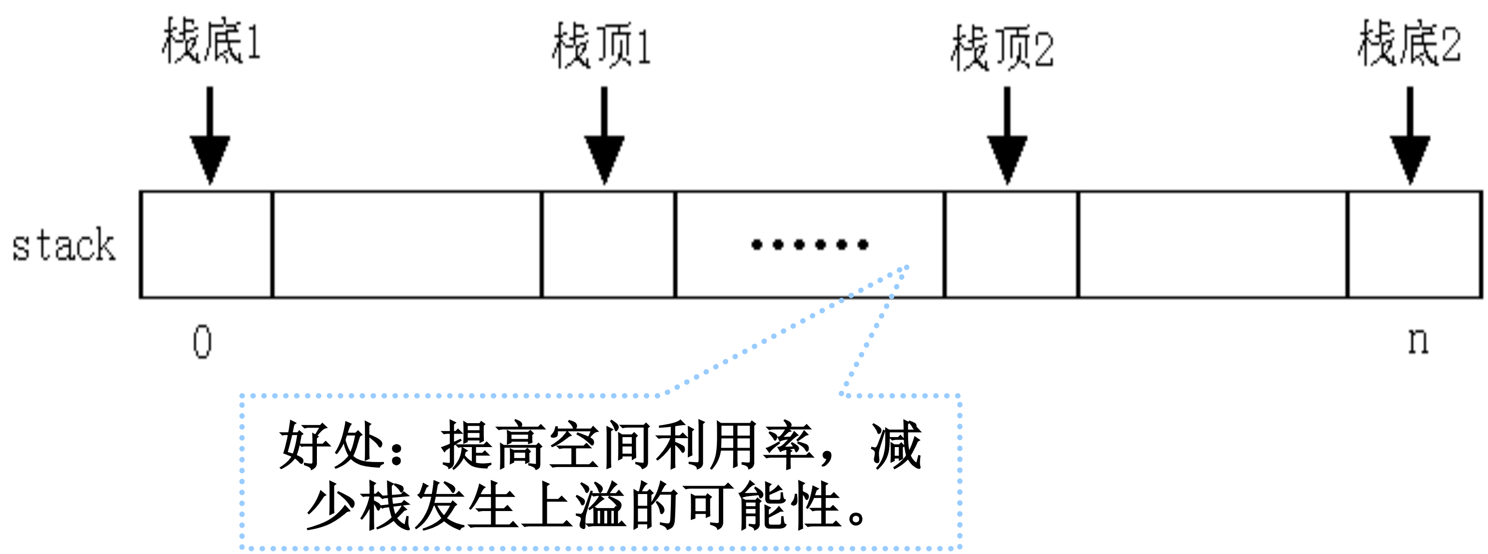
- 出栈算法

```
StackItem Pop(Stack S)
{
   if(StackEmpty(S)) Error("Stack is empty");
   else return S->data[S->top--] ;
}
```

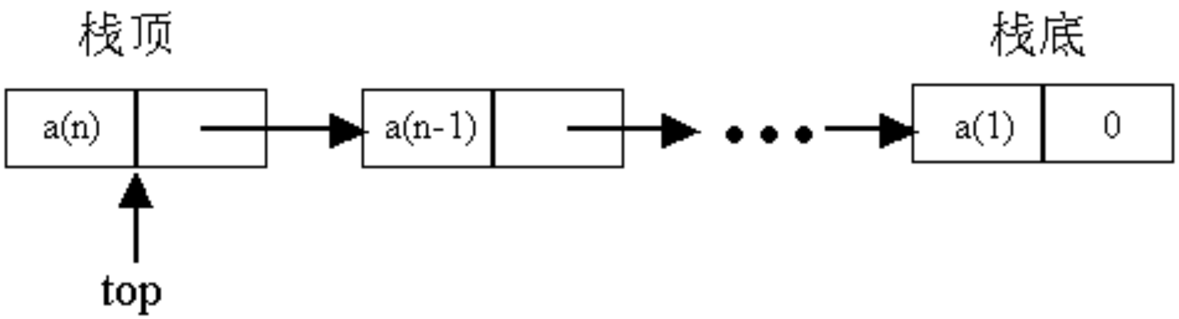
- **(2)栈的数组实现的优缺点**
  - 优点：所列的7个基本运算都可在O(1)的时间里完成，效率高。
  - 缺点：为了使每个栈在算法运行过程中不会溢出，通常要为每个栈预置一个较大的栈空间。另一方面，由于各个栈的实际大小在算法运行过程中不断变化。经常会发生其中一个栈满，而另一个栈空的情形，空间利用率低。

(3) 两个栈共用一个数组

利用栈底位置不变的特性，可以将2个栈的栈底分别设在数组stack的两端。然后各自向数组stack的中间伸展，如下图所示。



2、链栈—用指针实现栈



- (1)链栈的结点类型定义：

```
typedef struct snode *slink;
typedef struct  snode
{  StackItem  element;
    slink  next;
}StackNode;
```

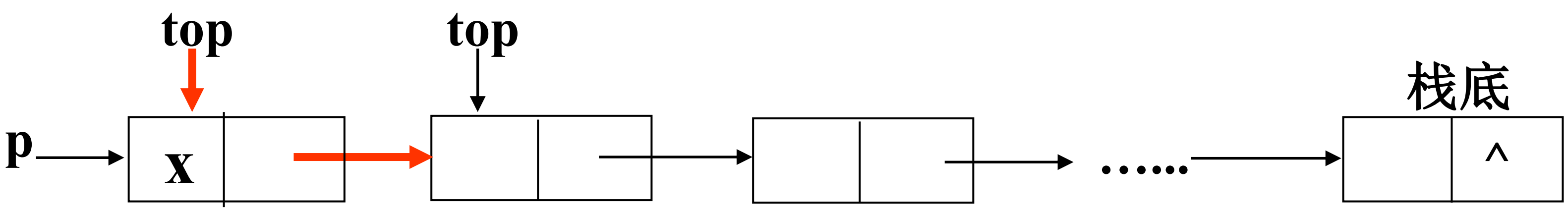
2、链栈—用指针实现栈

- (2)用指针实现的链栈定义：

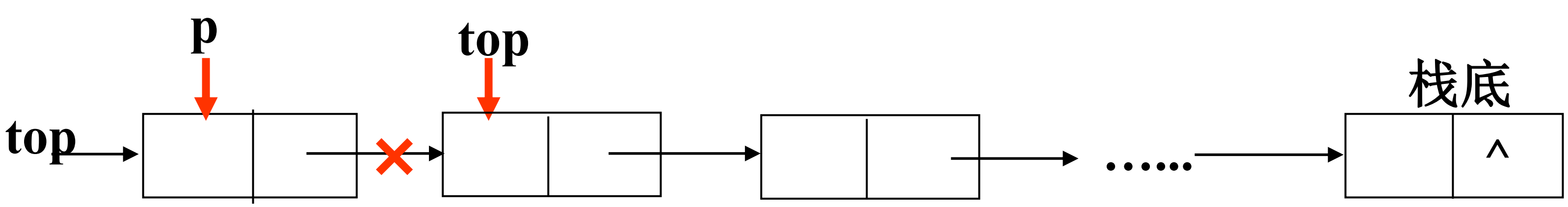
```
typedef struct lstack *Stack;
typedef struct  lstack
{
   slink  top; //栈顶结点指针
}Lstack;
```

(2)入栈、出栈算法实现及演示:

- 入栈算法



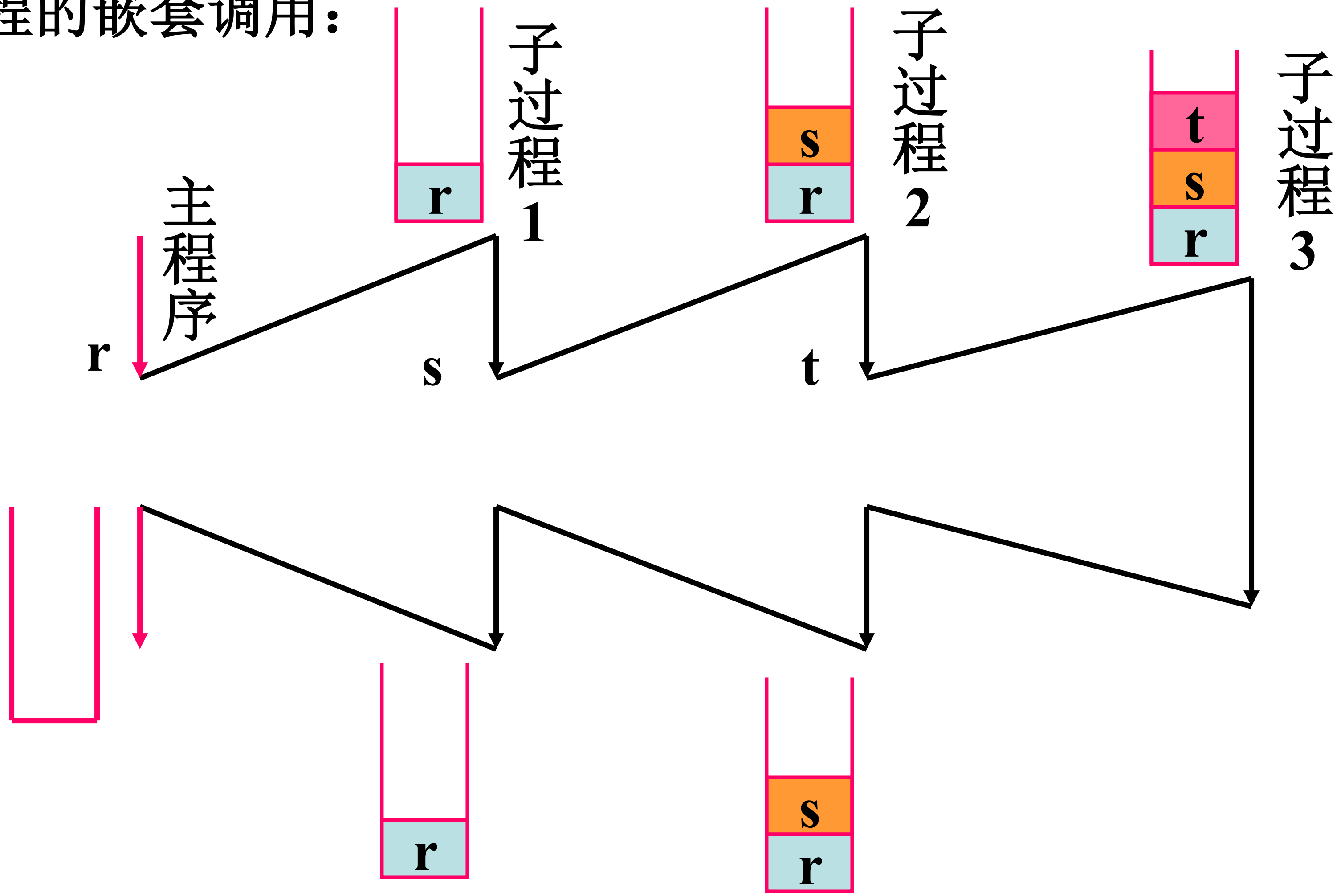
- 出栈算法



返回章目录

3.3 栈的应用

1、过程的嵌套调用：



2、递归过程及其实现：

- 递归：函数直接或间接的调用自身叫递归
- 实现：建立递归工作栈

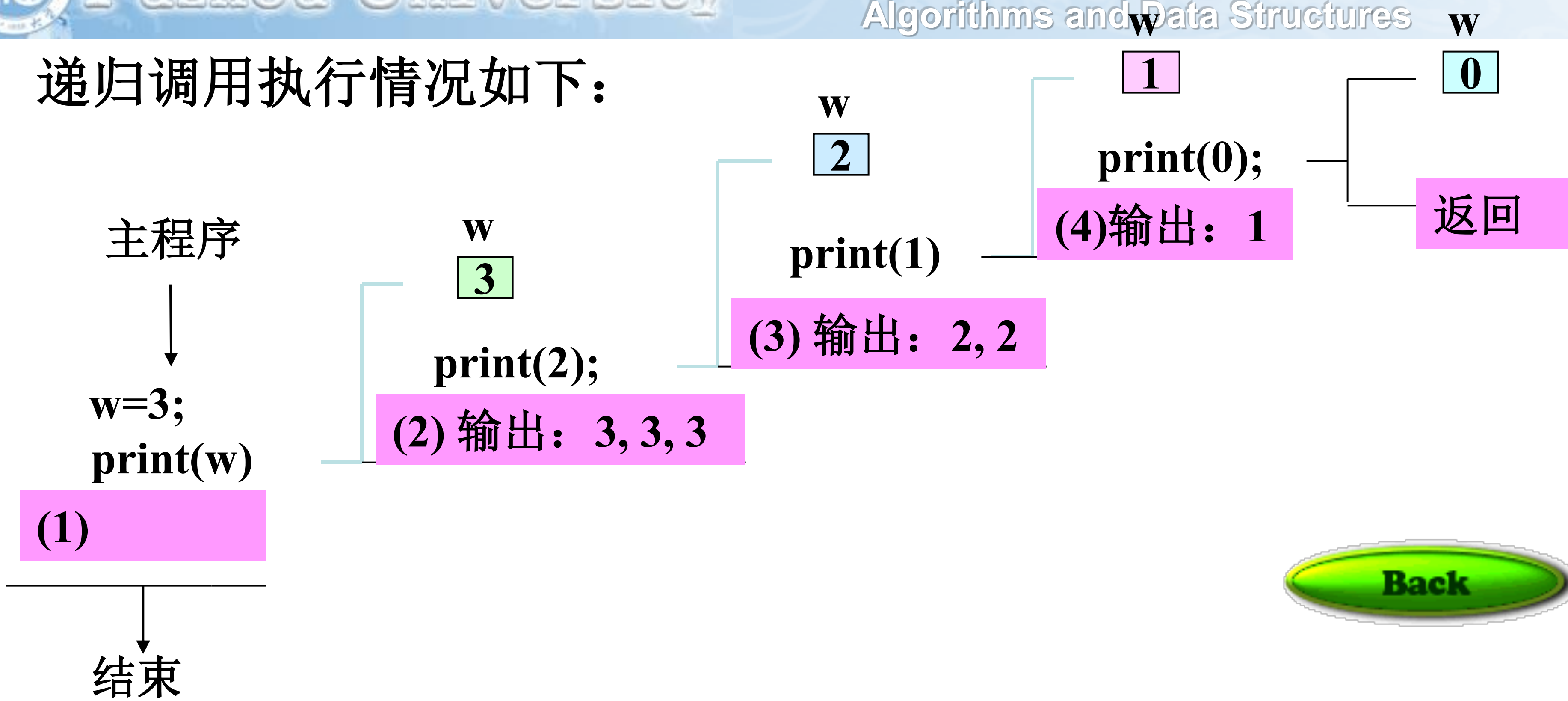
例 递归的执行情况分析

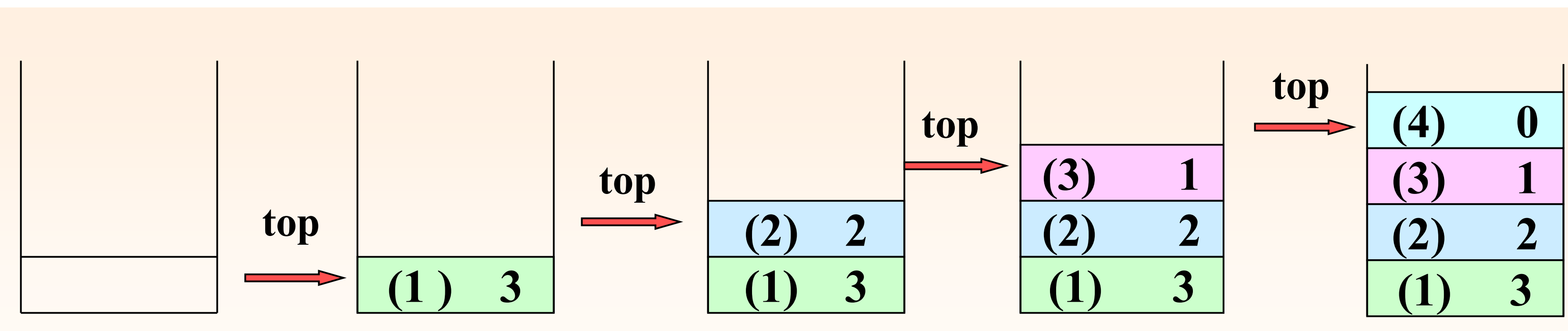
Click Here

```
void  print(int  w)
{     int  i;
      if ( w!=0)
      {     print(w-1);
            for(i=1;i<=w;++i)
                  printf("%3d,", w);
            printf("/n");
      }
}
```

运行结果：
1，
2， 2，
3， 3， 3，

递归调用执行情况如下：


主程序
w=3;
print(w)
(1)
结束
w
3
print(2);
(2) 输出：3, 3, 3
w
2
print(1)
(3) 输出：2, 2
w
1
print(0);
(4)输出：1
w
0
返回
Back


top
(1) 3
top
(2) 2
(1) 3
top
(3) 1
(2) 2
(1) 3
top
(4) 0
(3) 1
(2) 2
(1) 3

3、算术表达式求值

1、算术表达式的定义

在计算机中，表达式都是由**操作数(operand)**、**运算符(operator)**和**界限符(delimiter)**组成。

只含二元运算符的算术表达式可定义为：

表达式::= 操作数 运算符 操作数
操作数::= 简单变量 ｜ 表达式
简单变量::= 标识符 ｜ 无符号整数

例1：**Exp = 3*5+(6-8/4)*7#**

3、算术表达式求值

2、算术表达式的表示方式

假设 Exp = S1 + OP + S2

- S1 + OP + S2 称为表达式的中缀表示法（简称中缀式）
- S1 + S2 + OP 称为表达式的后缀表示法（简称后缀式）
- OP + S1 + S2 称为表达式的前缀表示法（简称前缀式）

例2：若Exp= a×b+(c-d/e)×f

后缀式为：ab×cde/-f×+
前缀式为：+×ab×-c/def

动画演示>>

>>

3、算术表达式求值

a	×	b	+	(	c	−	d	/	e	)	×	f

下一步

<<

3、算术表达式求值

3、后缀表达式求值

后缀式的求值规则：

——“先找运算符，后找操作数”

例3：对后缀式 Exp=ab×cde/-f×+# 求值

动画演示>>

3、算术表达式求值

如何从后缀式求值?

下一步

>>

3、算术表达式求值

3、后缀表达式求值

利用栈进行后缀表达式求值的基本思想：

1）从左到右读入后缀表达式，

2）若读入的是一个操作数，就将它压入栈；

3）若读入的是一个运算符op，就从栈中弹出两个操作数，设为x和y，计算表达式x op y的值，并将计算结果压入栈；

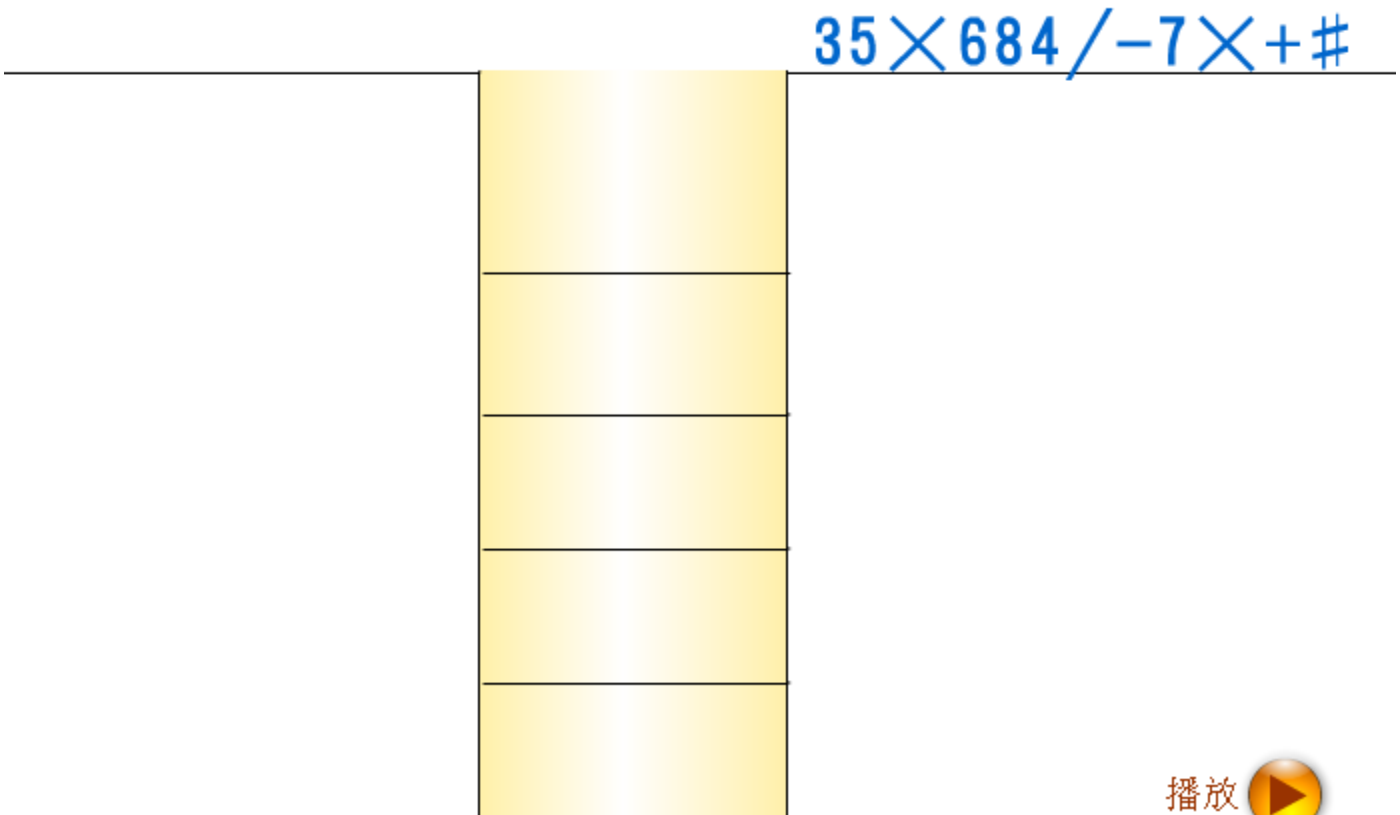
对整个后缀表达式读入结束时，栈顶元素就是计算结果。

例4：求后缀表达式 3 5×6 8 4/-7×+# 的值

动画演示>>

>>

3、算术表达式求值



播放

<<

3、算术表达式求值

4、原表达式向后缀式的转换

例5：

（1）原表达式：a×b/c×d-e+f

后缀式：ab×c/d×e-f+

（2）原表达式：a+b×c-d/e×f

后缀式：abc×+de/f×+

给每个运算符赋以一个优先级，如下：

运算符	#	(	)	+	-	×	/
优先级	-1	3/0	0	1	1	2	2

3、算术表达式求值

4、原表达式向后缀式的转换

利用栈实现原表达式向后缀式转换的基本思想：

1) 设立运算符栈，预设运算符栈的栈底为“#”；

2) 若当前字符是操作数，则直接输出到后缀式；

3) 若当前字符为运算符且优先级大于栈顶运算符，则进栈，否则弹出栈顶运算符输出到后缀式；

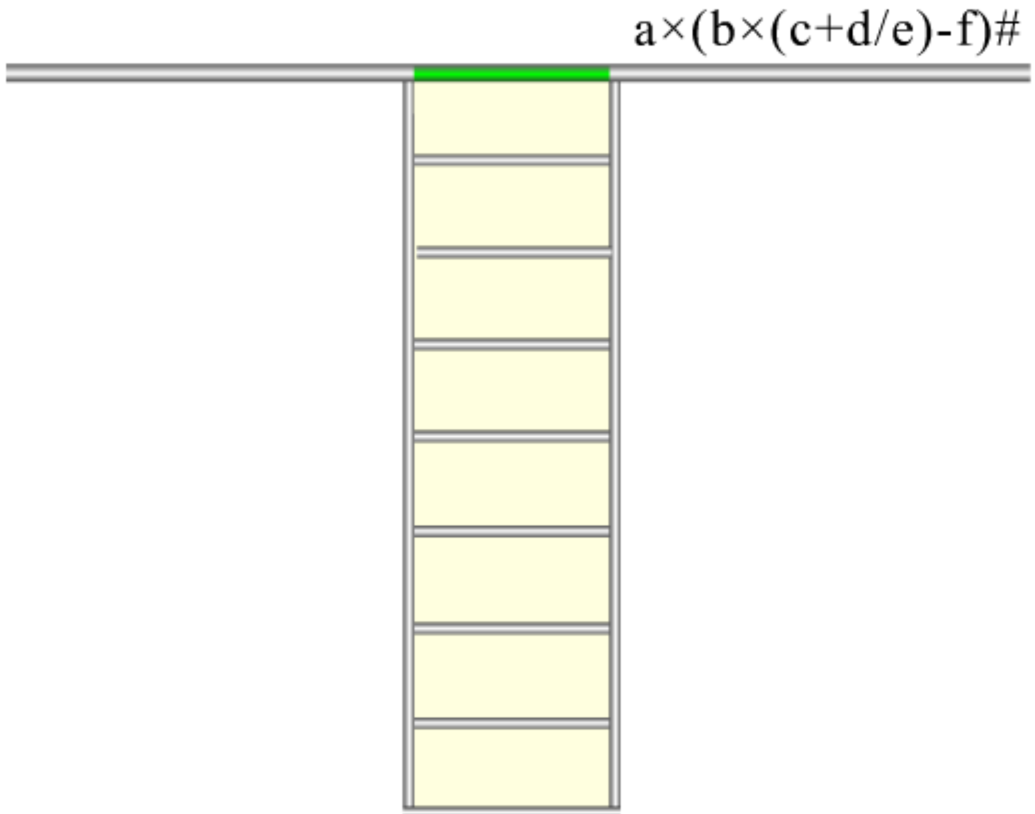
例6：求表达式 a×(b×(c+d/e)-f)# 的后缀式

动画演示>>

>>

3、算术表达式求值

表达式“a×(b×(c+d/e)-f)#”转换成后缀式的演算过程如下所示:

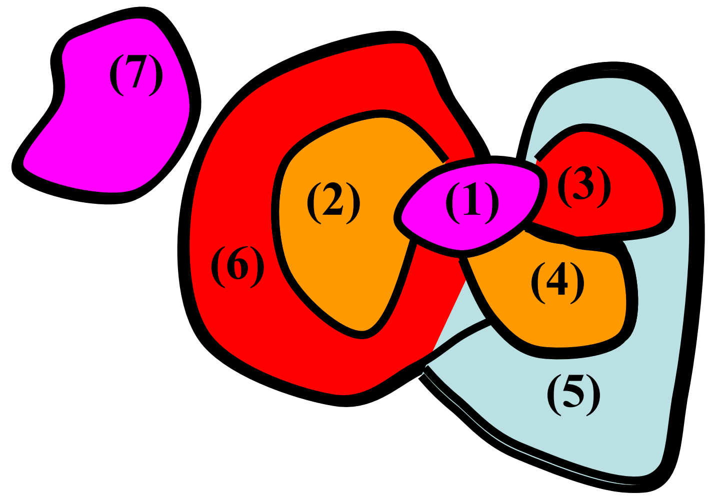


<<

4、地图四染色问题

R [7][7]

	1	2	3	4	5	6	7
1	0	1	1	1	1	1	0
2	1	0	0	0	0	1	0
3	1	0	0	1	1	0	0
4	1	0	1	0	1	1	0
5	1	0	1	1	0	1	0
6	1	1	0	1	1	0	0
7	0	0	0	0	0	0	0



1	2	3	4	5	6	7
1	2	3	2	4	3	1

1# 紫色	2# 黄色
3# 红色	4# 绿色

5、等价类划分问题

(1)问题的提出

给定集合S及一系列形如“x等价于y”的等价性条件，要求给出S的满足所列等价性条件的等价类划分。其中x和y是S中的元素。

- 复习:

(1)集合上的等价关系和集合关于某一等价关系的等价类划分等概念;

(2)举出3个你熟悉的等价关系和等价类划分。

(2) 问题的数学化

我们总可以用整数来表示集合中的元素。因此，如果集合S中共有n个元素，则可将集合S表示为｛1，2，…，n｝，而元素i和j的等价性条件可表示为i≡j，1≤i，j≤n。

这样，问题可一般地表述为：已知S=｛1，2，…，n｝上的一个等价关系由 r 个等价性条件{i_t≡j_t, 1≤i_t，j_t≤n, t=1,2,3,…,r}来表示。

要求该等价关系所确定的等价类划分。

(3) 举例

给定集合 S =｛1，2，…，7｝，及等价性条件：1≡2，5≡6，3≡4，1≡4。则集合S的等价类划分如下：首先将S的每一个元素看成一个等价类。然后顺序地处理所给的等价性条件。每处理一个等价性条件，就得到一个相应的等价类划分：

1≡2　｛１，２｝｛３｝｛４｝｛５｝｛６｝｛７｝；
5≡6　｛１，２｝｛３｝｛４｝｛５，６｝｛７｝；
3≡4　｛１，２｝｛３，４｝｛５，６｝｛７｝；
1≡4　｛１，２，３，４｝｛５，６｝｛７｝。

最终所得到的集合S的等价类划分为：｛１，２，３，４｝｛５，６｝｛７｝。

返回章目录

THE END