案例3.3： 表达式求值

算术四则运算规则

(1) 先乘除,后加减

(2) 从左算到右

(3) 先括号内,后括号外

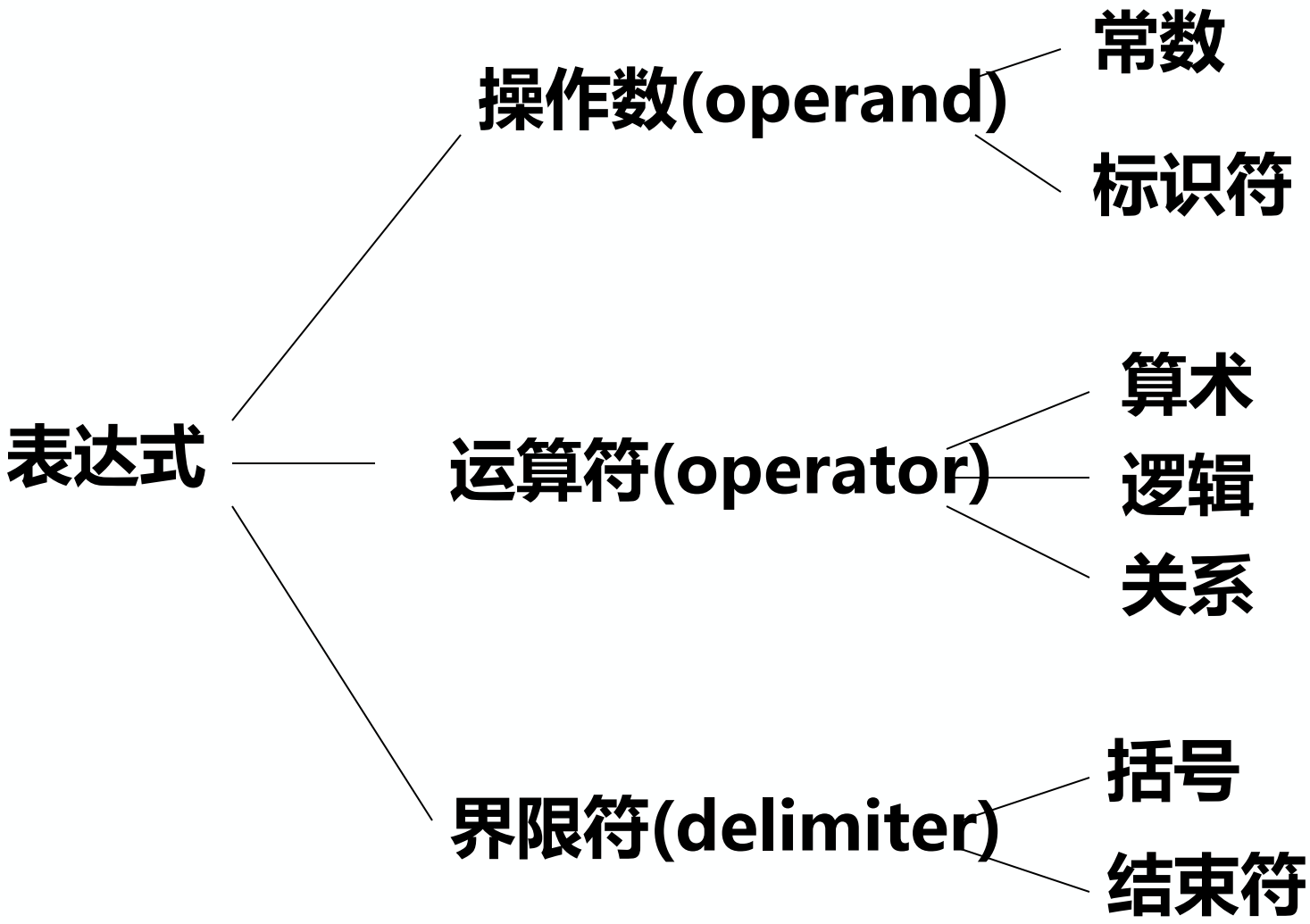

表达式
操作数(operand)
常数
标识符
运算符(operator)
算术
逻辑
关系
界限符(delimiter)
括号
结束符

表3.1　算符间的优先关系

θ_1 \ θ_2	+	-	*	/	(	)	#
+	>	>	<	<	<	>	>
-	>	>	<	<	<	>	>
*	>	>	>	>	<	>	>
/	>	>	>	>	<	>	>
(	<	<	<	<	<	=	x
)	>	>	>	>	x	>	>
#	<	<	<	<	<	x	=

2020年4月26日

【算法步骤】

设定两栈： OPND-----操作数或运算结果　OPTR------运算符

① 初始化OPTR栈和OPND栈，将表达式起始符“#”压入OPTR栈。

② 扫描表达式，读入第一个字符ch，如果表达式没有扫描完毕至“#”或OPTR的栈顶元素不为“#”时，则循环执行以下操作：

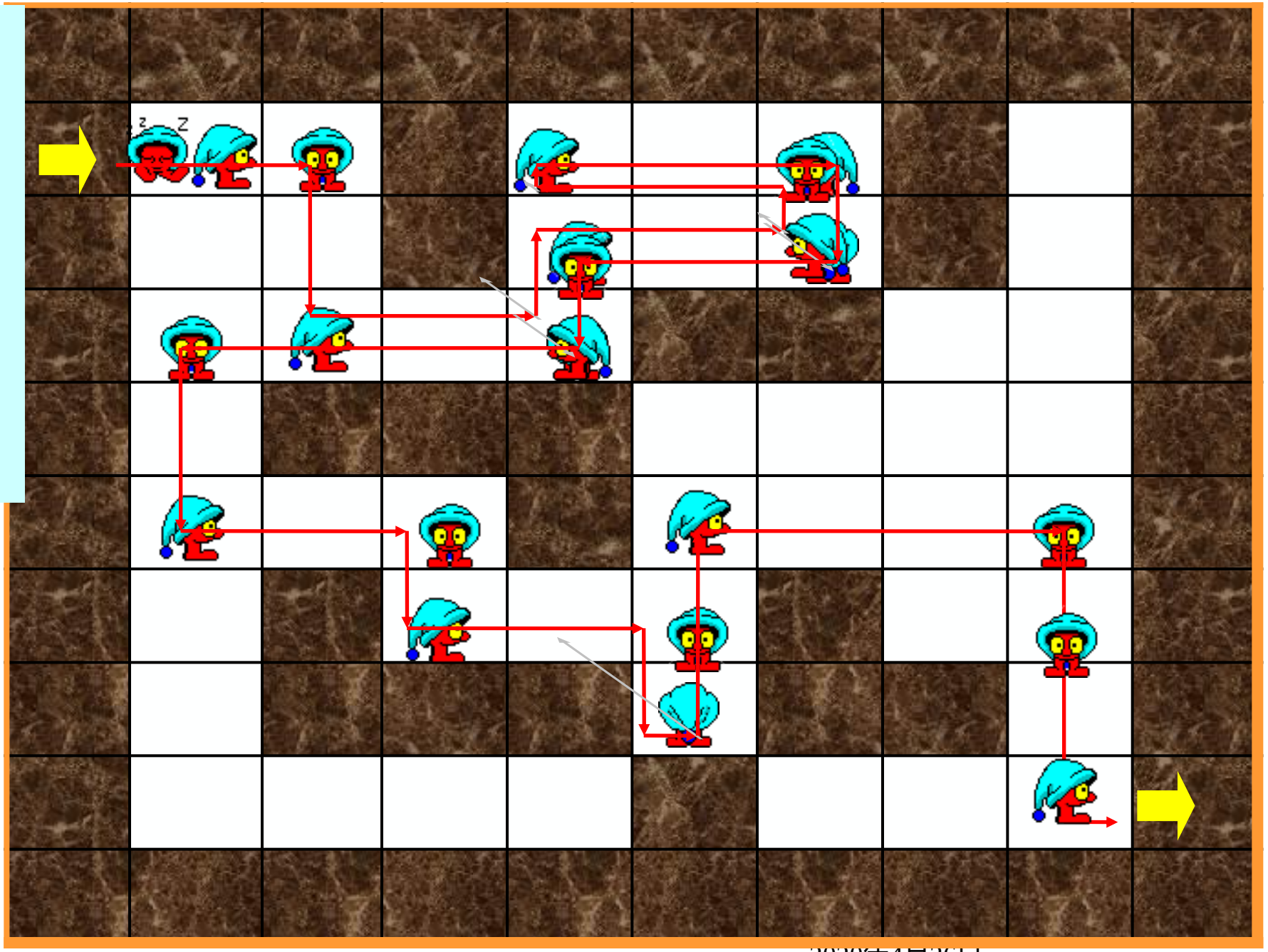
- 若ch不是运算符，则压入OPND栈，读入下一字符ch；
- 若ch是运算符，则根据OPTR的栈顶元素和ch的优先级比较结果，做不同的处理：
  - 若是小于，则ch压入OPTR栈，读入下一字符ch；
  - 若是大于，则弹出OPTR栈顶的运算符，从OPND栈弹出两个数，进行相应运算，结果压入OPND栈；
  - 若是等于，则OPTR的栈顶元素是“(”且ch是“)”，这时弹出OPTR栈顶的“(”，相当于括号匹配成功，然后读入下一字符ch。

③ OPND栈顶元素即为表达式求值结果，返回此元素。

```
OperandType  EvaluateExpression( ) {
 InitStack (OPTR);  Push (OPTR，'#') ;
 InitStack (OPND);  ch = getchar( );
 while (ch!= '#' || GetTop(OPTR)! = '#') {
  if (! In(ch)){Push(OPND,ch); ch = getchar(); }  // ch不是运算符则进栈
  else
    switch (Precede(GetTop(OPTR),ch)) { //比较优先权
      case '<' :   //当前字符ch压入OPTR栈，读入下一字符ch
          Push(OPTR, ch);  ch = getchar();  break;
      case '>' :   //弹出OPTR栈顶的运算符运算，并将运算结果入栈
          Pop(OPTR, theta);
          Pop(OPND, b);  Pop(OPND, a);
          Push(OPND, Operate(a, theta, b)); break;
      case '=' :   //脱括号并接收下一字符
          Pop(OPTR,x);    ch = getchar();   break;
      } // switch
 } // while
 return GetTop(OPND);} // EvaluateExpression
```

OPTR	OPND	INPUT	OPERATE
#		3*(7-2)#	Push(opnd,’3’)
#	3	*(7-2)#	Push(optr,’*’)
#,*	3	(7-2)#	Push(optr,’(’)
#,*,(	3	7-2)#	Push(opnd,’7’)
#,*,(	3,7	-2)#	Push(optr,’-’)
#,*,(,—	3,7	2)#	Push(opnd,’2’)
#,*,(,—	3,7,2	)#	Operate(7-2)
#,*,(	3,5	)#	Pop(optr)
#,*	3,5	#	Operate(3*5)
#	15	#	GetTop(opnd)

迷宫求解



迷宫求解

求解思想：回溯法

从入口出发，按某一方向向未走过的前方探索

✓若能走通，则到达新点，否则试探下一方向；

✓若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探

直到所有可能的通路都探索到，或找到一条通路，或无路可走又返回到入口点。

迷宫求解

需要解决的问题：

1、表示迷宫的数据结构
2、试探方向
3、栈的设计
4、防止重复到达某点，避免发生死循环

2020年4月26日

迷宫求解

1、表示迷宫的数据结构

表示一个m行n列迷宫：

用maze[m][n]表示，$0 \le i < m, 0 \le j < n$

maze[i][j] = 0　　通路

maze[i][j] = 1　　不通

改进：

用maze[m+2][n+2]表示

且maze[i][j]=1，$i=0$或$m+1$, $j=0$或$n+1$

入口坐标为（1，1），出口坐标为（m，n）

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	0	0	0	1	0	1
2	1	0	0	1	0	0	0	1	0	1
3	1	0	0	0	0	1	1	0	0	1
4	1	0	1	1	1	0	0	0	0	1
5	1	0	0	0	1	0	0	0	0	1
6	1	0	1	0	0	0	1	0	0	1
7	1	0	1	1	1	0	1	1	0	1
8	1	1	0	0	0	1	0	0	0	1
9	1	1	1	1	1	1	1	1	1	1

迷宫求解

迷宫的定义：

```
#define  m   8  /*迷宫的实际行*/
#define  n   8  /*迷宫的实际列*/
int maze [m+2][n+2] ;
```

2、试探方向

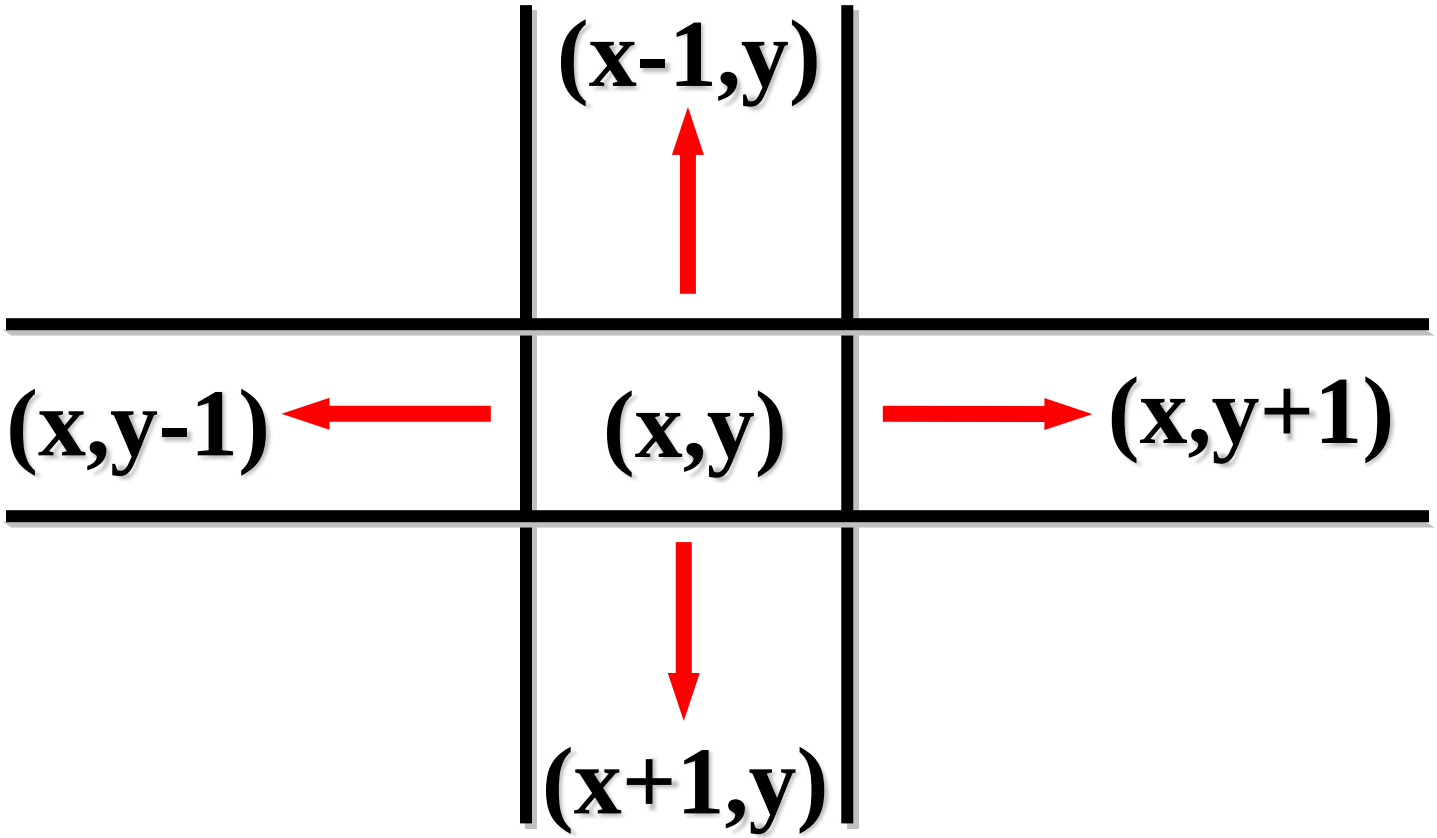
表示位置的类型PosType定义如下：

```
typedef  struct
{
   int x，y;
} PosType ;
```

迷宫求解

试探顺序规定为：从正东沿顺时针方向

与点(x，y)相邻的4个点及坐标



迷宫求解

3、栈的设计

栈中每个元素的组成：

通道块在路径上的序号

坐标位置

前进方向（东为1，南为2，西为3，北为4）

栈元素的类型定义：

```
typedef struct {
    int ord;
    PosType seat;
    int di;
}SElemType;
```

迷宫求解

4、防止重复到达某点

走过不通之处要加以标记（MarkPrint操作）

2020年4月26日

案例3.4： 舞伴问题

【案例分析】

- 设置两个队列分别存放男士和女士入队者
- 假设男士和女士的记录存放在一个数组中作为输入，然后依次扫描该数组的各元素，并根据性别来决定是进入男队还是女队。
- 当这两个队列构造完成之后，依次将两队当前的队头元素出队来配成舞伴，直至某队列变空为止。
- 此时，若某队仍有等待配对者，则输出此队列中排在队头的等待者的姓名，此人将是下一轮舞曲开始时第一个可获得舞伴的人。

【数据结构】

```
//- - - - - 跳舞者个人信息- - - - -
typedef struct
{
  char name[20];                //姓名
  char sex;                     //性别，'F'表示女性，'M'表示男性
}Person;
//- - - - - 队列的顺序存储结构- - - - -
#define MAXQSIZE 100            //队列可能达到的最大长度
typedef struct
{
  Person *base;                 //队列中数据元素类型为Person
  int front;                    //头指针
  int rear;                     //尾指针
}SqQueue;
SqQueue Mdancers,Fdancers;      //分别存放男士和女士入队者队列
```

【算法步骤】

① 初始化Mdancers队列和Fdancers队列。

② 反复循环，依次将跳舞者根据其性别插入Mdancers队列或Fdancers队列。

③ 当Mdancers队列和Fdancers队列均为非空时，反复循环，依次输出男女舞伴的姓名。

④ 如果Mdancers队列为空而Fdancers队列非空，则输出Fdancers队列的队头女士的姓名。

⑤ 如果Fdancers队列为空而Mdancers队列非空，则输出Mdancers队列的队头男士的姓名。

队列的其它应用

【例】汽车加油站

结构：入口和出口为单行道，加油车道若干条n

每辆车加油都要经过三段路程，三个队列

- 1.入口处排队等候进入加油车道
- 2.在加油车道排队等候加油
- 3.出口处排队等候离开

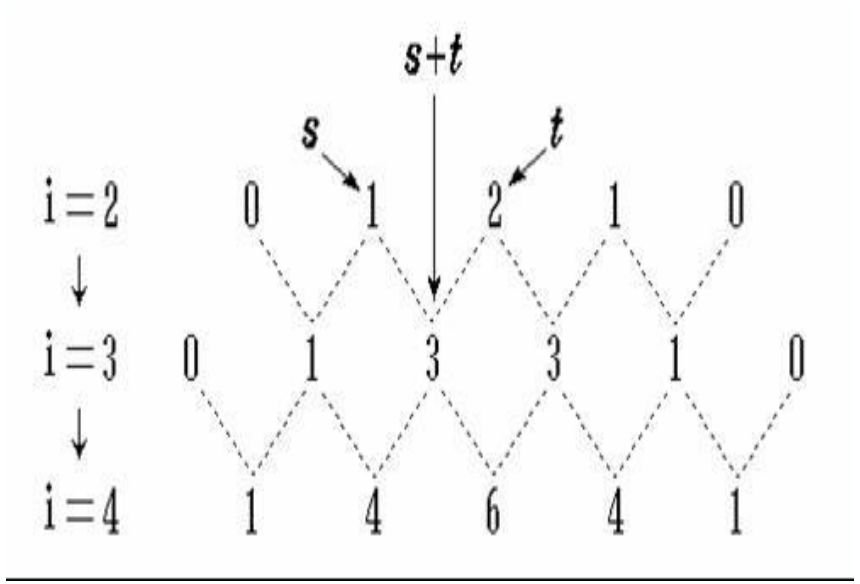
若用算法模拟，需要设置n+2个队列。

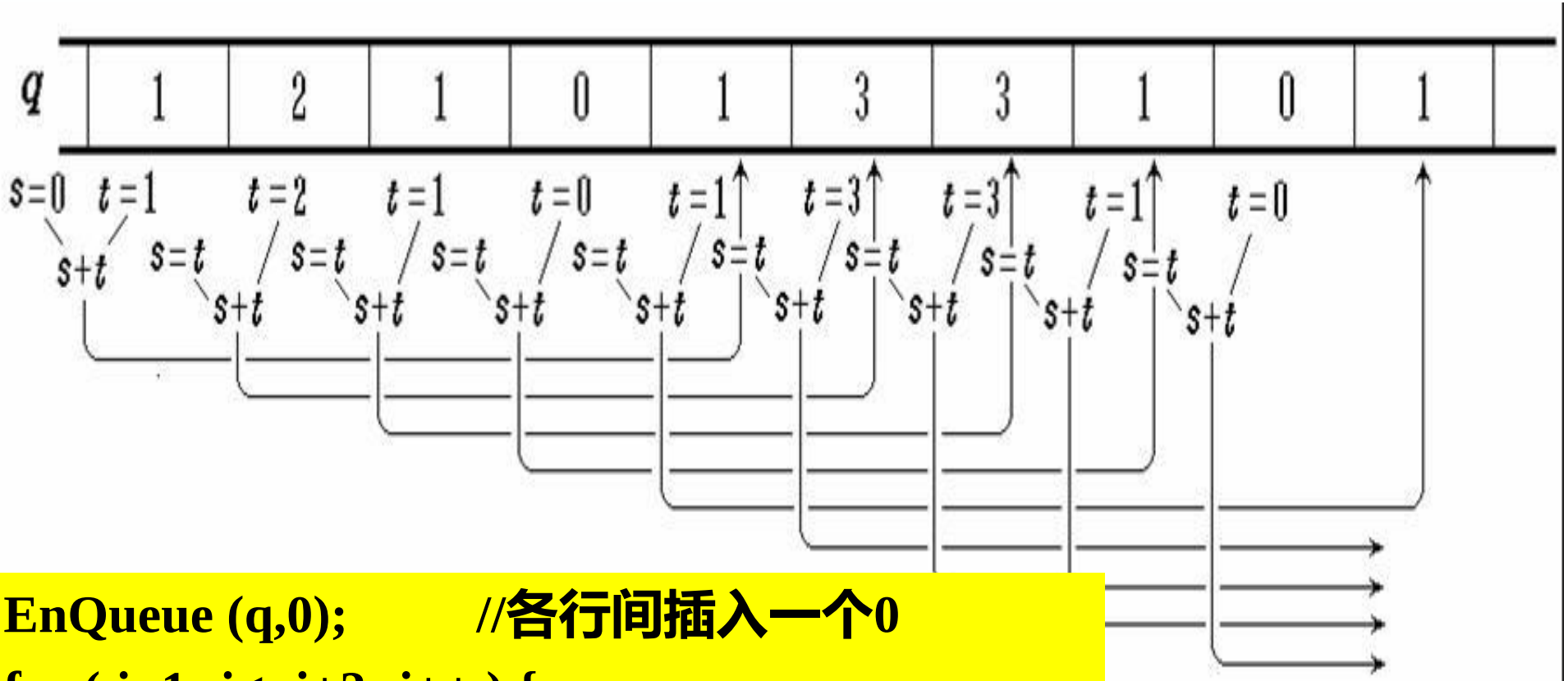
【例】模拟打印机缓冲区

- ✓ 在主机将数据输出到打印机时，主机速度与打印机的打印速度不匹配
- ✓ 为打印机设置一个**打印数据缓冲区**，当主机需要打印数据时，先将数据依次写入缓冲区，写满后主机转去做其他的事情
- ✓ 而打印机就从缓冲区中按照**先进先出**的原则依次读取数据并打印

【例】打印杨辉三角形

```
                1     1                    i = 1
             1     2     1                    2
          1     3     3     1                 3
       1     4     6     4     1              4
    1     5     10    10    5     1           5
 1     6     15    20    15    6     1        6
```





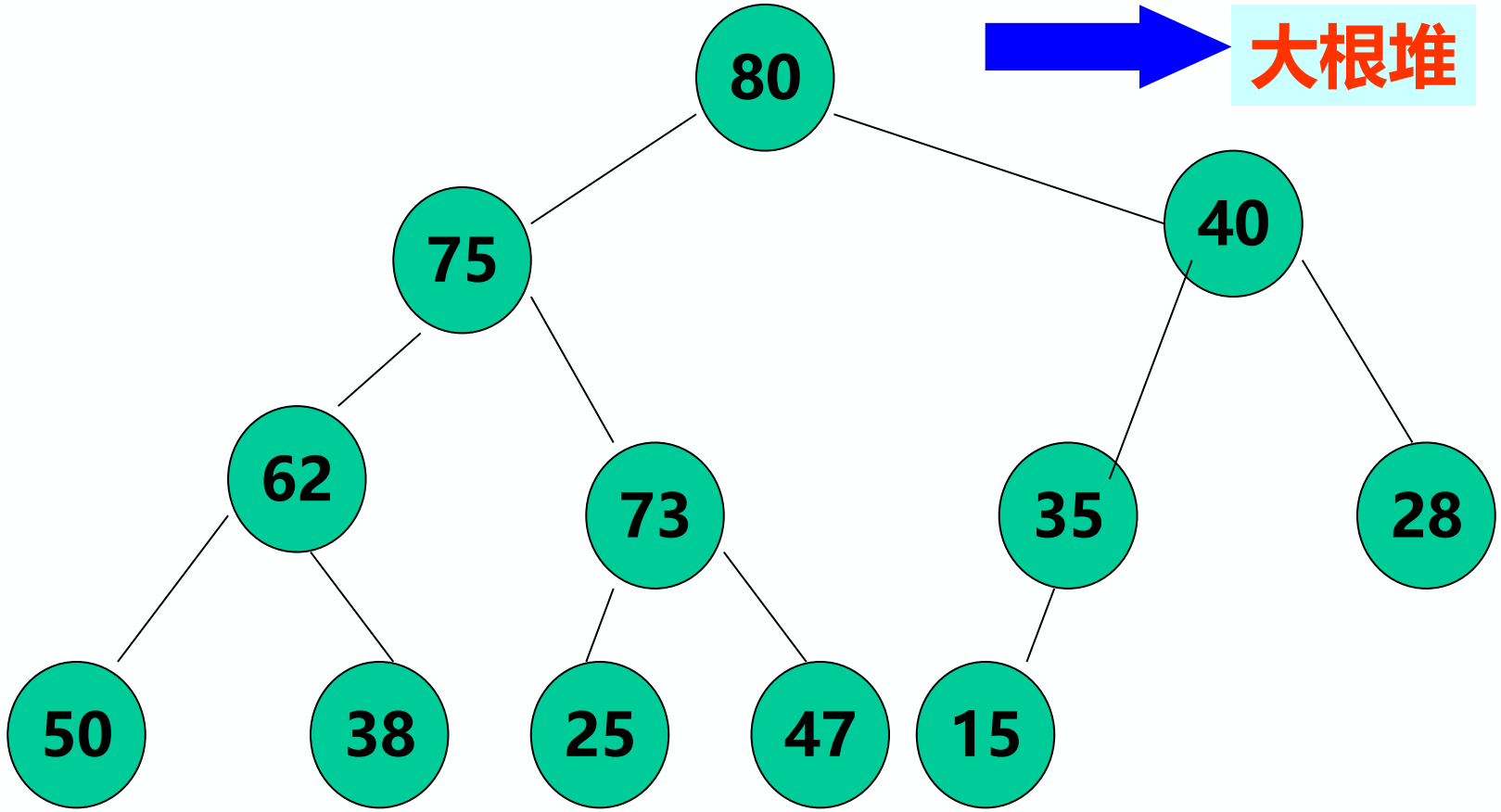

```
EnQueue (q,0);          //各行间插入一个0
for ( j=1; j<=i+2; j++ ) {
    DeQueue (q,t );       //一个系数出队
    EnQueue (q, s+t ); //计算下一行系数，并进队
    s = t;
    if ( j != i+2 ) cout << s << ' ';       //第i+2个0 }
```

优先级队列(priority_queue)---堆

- 每次从队列中取出的是具有最高优先权的元素
- 任务优先权及执行顺序的关系

任务编号	1	2	3	4	5
优先权	20	0	40	30	10
执行顺序	3	1	5	4	2

数字越小，优先权越高



2020年4月26日

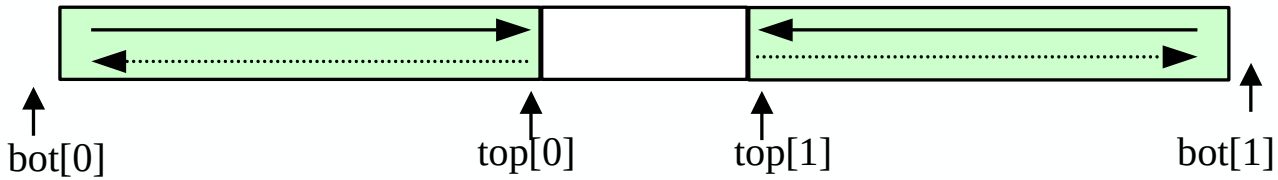
小结

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. 熟练掌握栈的顺序栈和链栈的进栈出栈算法，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进队出队算法，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

算法设计题

（1）将编号为0和1的两个栈存放于一个数组空间V[m]中，栈底分别处于数组的两端。当第0号栈的栈顶指针top[0]等于-1时该栈为空；当第1号栈的栈顶指针top[1]等于*m*时，该栈为空。两个栈均从两端向中间增长。试编写双栈初始化，判断栈空、栈满、进栈和出栈等算法的函数。双栈数据结构的定义如下：

```
typedef struct{
  int top[2], bot[2];  //栈顶和栈底指针
   SElemType *V;       //栈数组
  int m;               //栈最大可容纳元素个数
}DblStack;
```



Back

答案

```
//初始化一个大小为m的双向栈s
Status Init_Stack(DblStack &s,int m)
{
  s.V=new SElemType[m];
  s.bot[0]=-1;
  s.bot[1]=m;
  s.top[0]=-1;
  s.top[1]=m;
  return OK;
}
```

答案

```
//判栈i空否, 空返回1, 否则返回0
int IsEmpty(DblStack s,int i)
{return s.top[i] == s.bot[i]; }

//判栈满否, 满返回1, 否则返回0
int IsFull(DblStack s)
{  if(s.top[0]+1==s.top[1]) return 1;
   else return 0;}
```

答案

```
void Dblpush(DblStack &s,SElemType x,int i)
{
 if( IsFull (s ) )  exit(1);
            // 栈满则停止执行
   if ( i == 0 ) s.V[ ++s.top[0] ] = x;
   //栈0情形：栈顶指针先加1, 然后按此地址进栈
   else s.V[--s.top[1]]=x;
   //栈1情形：栈顶指针先减1, 然后按此地址进栈
}
```

答案

```
int Dblpop(DblStack &s,int i,SElemType &x)
{if ( IsEmpty ( s,i ) ) return 0;
    //判栈空否, 若栈空则函数返回0
  if ( i == 0 ) s.top[0]--;   //栈0情形：栈顶指针减1
  else s.top[1]++; //栈1情形：栈顶指针加1
  return 1;
}
```

算法设计题

（10）已知f为单链表的表头指针，链表中存储的都是整型数据，试写出实现下列运算的递归算法：

① 求链表中的最大整数；

② 求链表的结点个数；

③ 求所有整数的平均值。

提示

```
int GetMax(LinkList p){//求链表中的最大整数
        if(!p->next)        return p->data;
        else
        {
                int max=GetMax(p->next);
                return  p->data>=max ? p->data:max;
        }
}
void main( ){
    LinkList L;
        CreatList(L);
        cout<<"链表中的最大整数为： "<<GetMax(L->next)<<endl;
        ……}
```