**栈**

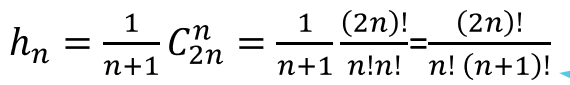
**1. 栈的基本概念**

**栈：**是限制在一端进行插入和删除操作的线性表，先进后出

**栈顶：**用栈顶指针来指示栈顶元素

**栈底：**固定端，也称为表头

**空栈：**没有元素时称为空栈

****

**n个元素进栈，可能的出站序列种数**

**2. 顺序栈**

**动态顺序栈、静态顺序栈**

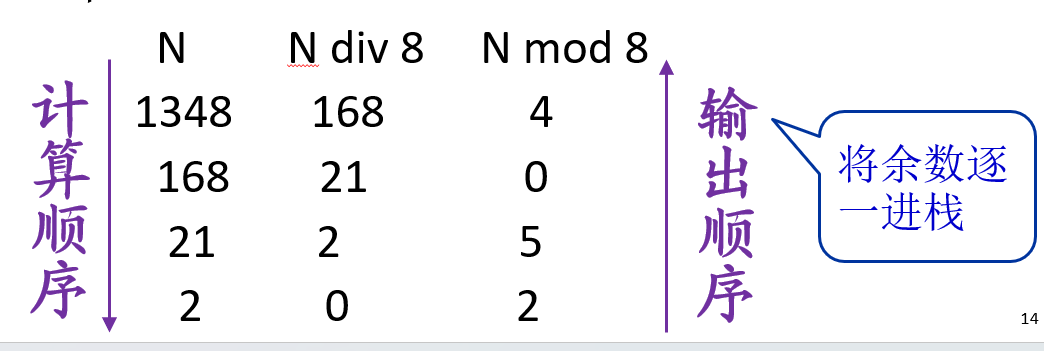
**栈底指针base：**指向栈底固定不变

**栈顶指针****top：**指向数组中的下一个空闲存储位置

**作为栈空的标记：top==base**

**栈的应用场景：**

数制转换、括号匹配



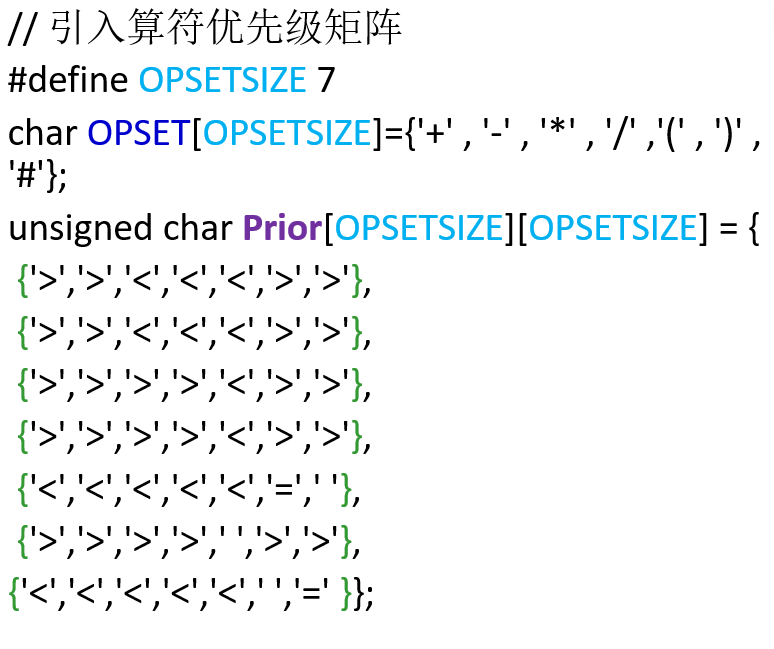
**3. 链式栈**

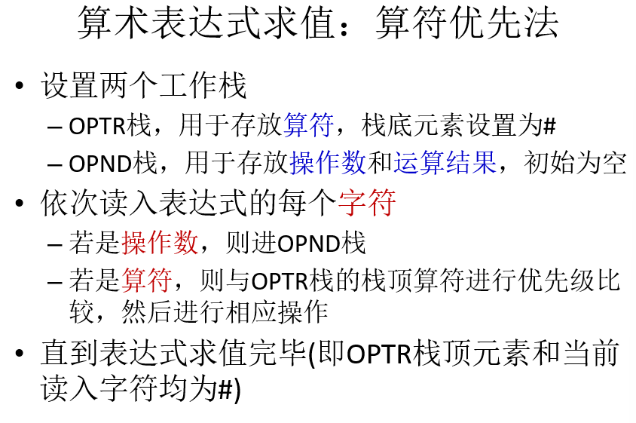
**数据结构LinkedStack：头节点**

data数据

next指向下一个栈位

**链式栈不设base，由结点的next是否为NULL确定是否到头/为空栈**

**4. 栈的应用举例**



**利用栈计算中缀表达式（一般表达式）**

* ​**​数字​**​：读取完整数字（处理多位数），压入 OPND。
* ​**​左括号 (​**​：压入 OPTR。
* ​**​右括号 )​**​：弹出 OPTR栈顶运算符并计算，直到遇到 (。
* ​**​运算符​**​： 若当前算符优先级 ≤ 栈顶，先计算栈顶运算并将结果压入OPND，**直到 优先级大于栈顶**，再压入当前运算符，读下一个字符

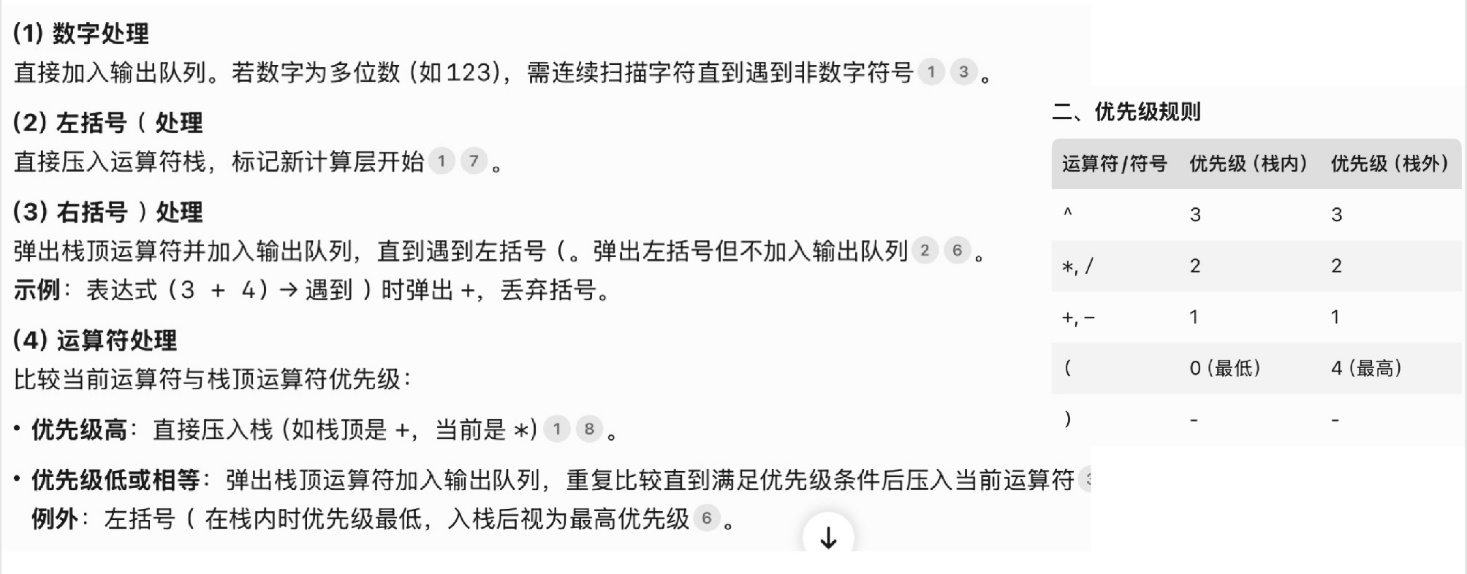
若大于，直接压栈，读下一个字符

**\*、/**> **+、-**> 左括号**(** > 右括号**)**、**#** 最低优先级

**利用栈计算后缀表达式（逆波兰表达式）**

一个工作栈

遇到操作数，进栈；遇到运算符，连续从栈中退出两个操作数，进行计算

**中缀表达式转换成后缀表达式**

* 1. 若ch是**操作数**，则**直接输出**，读下一个字符ch；
  2. 若ch是**运算符**，比较ch的优先级和栈顶运算符的优先级：
     1. 若ch的优先级高，则ch进栈，读下一个字符ch；
     2. 若ch的优先级低，则栈顶运算符退栈并输出，直到ch优先级大于栈顶，入栈，读下一个字符；
     3. 若两者的**优先级相等**，则**栈顶运算符退栈**，**若退出的是“(”，读下一个字符ch**

**迷宫寻路（栈版本-深度优先搜索）**

二维数组储存迷宫信息

栈存储路径信息：每个元素包含：行、列、方向

寻路过程： 若当前位置可通，则入栈，纳入路径，选方向1继续前进

若当前位置不可通，则弹栈后退，选择栈中记录方向的下一个方向继续前进

若回退后发现四周均不通，则将当前位置从路径中删除，继续回退

**递归的Master定理：**

|  |  |  |
| --- | --- | --- |
| 条件 | 复杂度 | 理解 |
| logb(a) > d **递归主导​** | **O(nlogb(a))​** | 拆分子问题的代价远高于递归外的操作，时间复杂度由拆分次数主导 |
| logb(a) = d **内外均衡​** | **O(nd·logn)​** | 递归拆分与外部操作复杂度相当，每一层需叠加O(n^d)操作，共logn层 |
| logb(a) < d **外部操作主导** | ​**​O(nd)​** | 递归外操作复杂度远高于拆分，时间复杂度由外部操作主导 |

**队列**

**1.队列的基本概念**

队列：是运算受限的线性表，只允许在表的一端进行插入，而在另一端进行删除，先进先出

**队头**（front）：允许进行删除的一端

**队尾**（rear）：允许进行插入的一端

空队列：队列中没有元素

**2. 队列的链式表示和实现**

**链队列LinkedQueue**：

front队头指针（指向头节点）

rear队尾指针，指向当前最后一个元素

**结点QNode：** data数据

next指向下一个结点

注意，**链队列设有头节点，front指向头结点**

**队空判断： rear==front**

**队满判断： rear==MAXQUEUESIZE**

**3. 队列的顺序表示和实现**

**(静态)顺序队列SqQueue**： Queue\_array存放结点的数组

queueSize队列空间的大小，可容纳的结点数

front、rear队头、队尾指针（数组下标，初始为0）

**结点QNode：** data数据

next指向下一个结点

**没有头节点！！！！！！**

队头指针指向队头元素，队尾指针指向**下一个空闲空间**

**假溢出：**尾指针达到数组的上界导致入队失败，但由于出入队过程中头尾指针只能后移，实际上头指针前还有空闲空间

**循环队列**

将为队列分配的向量空间看成为一个首尾相接的圆环

**循环队列CircularQueue**：ElemType \*base 执行动态分配的空间

front、rear

**结点QNode：** data数据

next指向下一个结点

**没有头节点！！！！！！**

**当队头、队尾指针到达MAXQUEUESIZE-1时，其加1操作的结果是指向0**

**即加一替换为（x= (x+1) % MAXQUEUESIZE）**

队空条件：rear==front 队满条件：rear==front 无法区分

* + 区分队空和队满的方法：

（1）CircularQueue中增加一个**队列长度**成员

（2）少用一个元素空间，以“尾指针的下一个位置为头指针”为队满标志，即**(rear+1) % MAXQUEUESIZE == front**

**4. 队列的应用举例**

**杨辉三角（二项式系数生成）**

C(n,i) = C(n-1,i) + C(n-1,i-1)

next\_row\_i = current\_row\_i + current\_row\_i-1

0 1 1 0 1 2 1 0 1 3 3 1 0 (每生成完一行后添加一个0)

**需要的队列长度为r+2**，r为所需要获取的最大行数

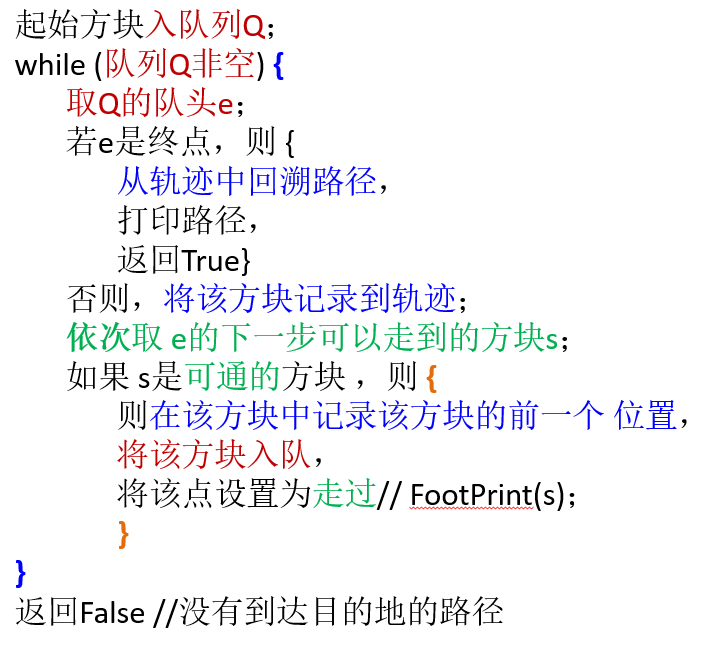
**迷宫寻路（队版本-广度优先搜索）**

typedef struct{

PosType seat；//该通道块在迷宫的坐标位置

PosType former；//从那个块走来

Int di；

}ElemType；

起始方块入队

while（队列非空）{

取队头e进行检查

若e是终点，按轨迹打印路径；

若不是，加入轨迹；

}

遍历e下一步能走到的方块s：

若s可通（周围标记过的点可认为不通），则将该点标记为走过，将该方块入队，将该点设置为走过

**运动会日程安排**

目标：使同一子集的元素没有冲突关系

**数据结构：**

用队列sq存放n个集合元素

用矩阵conflictMatrix[n][n]表示元素之间的冲突关系

用数组result[n]存放每个元素的子集/分组编号

用工作数组clash[n]记录与第k组已入组元素有冲突的元素情况

**算法流程：**

1.clash数组初始化为冲突矩阵的第0行。

2.找到clash数组中未被划分过下一个值为0的位置k，将冲突矩阵的第k行累加到clash数组上重复此行操作，直到clash中没有值0且未被划分位置。（划分组号记录在result中）

3.对下一个未被划分的元素（位置为k），clash数组初始化为冲突矩阵的第k行，执行上面的操作。

贪心法：

项目队列，依次与当前子集中的元素匹配，检查有没有冲突关系：

若与子集中现有的所有元素都不冲突，则出队，放入子集；

若有冲突，则放至队尾。

完成一次队列元素的遍历后，再开一个子集，直到队列为空。

Clash冲突数组叠加：

取对头元素e

将R的第e行赋值给clash

取队列下一个元素e，若与clash数组不冲突，出队，将R的第e行叠加到clash上

若冲突，则放到队尾

**离散事件模拟**

模拟银行业务流程

处理客户到达事件：将客户排到**当前最短**的队，如排在队头，生成该客户离开事件；生成下一客户到达事件

处理当前离开事件：删除相应队列头，之后，为当前队列的队头客户生成下一个的离开事件

生成的事件按**发生时间顺序**插入事件队列