

# 数据结构

# Data Structure

2017年秋季学期  
刘鹏远

# 栈的应用

## 应用一：数制转换问题 教材P48

基本转换原理： $n = (n \text{ div } d) * d + n \text{ mod } d$

整除部分 + 余数

d为2,8,16等，代表d进制

对整除部分进行div，然后保留余数

依次进行，得到每次的余数即为转换后的结果

保存的余数从先到后依次表示转换后的d 进制数的低位到高位，而输出是由高位到低位

例如  $(1348)_{10} = (2504)_8$ ，其运算过程如下：

n	n div 8	$n\%8$	
1348	168	4	$8^0$
168	21	0	$8^1$
21	2	5	$8^2$
2	0	2	$8^3$

计算，直到  $n==0$ ，将余数后进先出

**提问：**用什么结构/方法实现？

一：结果依次放顺序表，倒序输出； 二：栈(考试答案)

```
Status conversion(int n, int d)
{    //这里先声明并初始化一个栈S
    while(n) {
        Push(&S, n%d);
        n = n/d;    }
    while(!Is_empty(S)) {
        Pop(&S, &n);
        printf("%d",n);    }
    return OK;}
```

## 应用二：括号匹配问题

先考虑最简单仅有一种括号的情况：

对 $(a*(b+c)+d)+(e-b)$  括号均匹配，合法

对 $(a+b))(c+d)+a$  括号不全匹配，非法

对 $)a-b)+(c-d)$  不全匹配，非法

请问大家会怎么来写判断是否匹配的程序？

## 思路：

0、从左到右观察括号序列

1、如出现右括号，则其前面保存的一定是与之匹配的左括号

2、此时可以消除这两个括号。

重复一直到表达式尽头...或者不匹配

后出现的左括号，先匹配

处理策略：

1、遇到左括号入栈；

2、遇到右括号将左括号出栈进行匹配；

有右括号但栈空，或表达式结束时栈为空，均为不匹配，否则为匹配。

{ ( [ ] [ ] ) }，这类多种括号的情况，处理思路类似

问题：换种思路：从右向左看，如何？



```
Status match(char *s){//也可以改成从键盘输入字符
//这里先声明并初始化一个栈S
for(i=0;s[i]!='\0';i++)    {
    if(s[i] == '(')
        if(!Push(&S, '(')) return OVERFLOW;
    if(s[i] == ')')
        if(!Pop(&S, &c)) return ERROR;
}if(!Is_empty(S)) return ERROR;
return OK;}//返回OK表示匹配，其他为不匹配
```

如果是多种括号  
( [ { } ] )

```
if(s[i] == ')' || s[i] == ']' || s[i] == '}')
```

后续代码则请大家自行考虑如何解决

## 应用三----行编辑程序（最简单的功能）

每次终端接收单个字符，考虑用户可能输错，因此可以用后续的输入字符来对前面进行修改，每行输入完毕显示该行。

行编辑程序的字符可分为三类：

普通符、回车符\n、以及两个编辑符#，@

#表示删除前一字符      @是删除当前行

如果用户依次输入：

whli##ilr#e(s# \*s)回车

则输出：

while( \*s)

继续输入：

outcha@putchar(\*s=#++);回车

则输出：

putchar(\*s++);

处理思路：

普通字符依次入栈

遇到'#'则出栈一个元素

遇到'@'则清空当前栈

遇到'\n'则退栈至空输出。

重复以上。

```
void line_edit(){  
    //这里先声明并初始化一个栈S，声明其他变量  
    while((ch=getchar())!=EOF){//当ch非输入结束符: ^z+回车  
        while(ch!=EOF && ch!='\n'){  
            switch(ch){  
                case '#': Pop(&S, &c);break;  
                case '@': Clear(&S);break;  
                default: Push(&S, ch);break;}  
            ch=getchar(); }  
        Traverse(S);//从栈底到栈顶，最好用顺序栈  
        Destroy(&S); } }
```

## 应用四---表达式求值

不失一般性，考虑四则运算的算数表达式：

$$3+9*(10-3)/2+20$$

只要懂得运算规则即可

但计算机/程序/计算器其实都需要解析这个表达式才能求值

一个表达式由操作数(亦称运算对象)、操作符(亦称运算符)和分界符(如=)组成。

表达式中相邻两个操作符的计算次序为：

- a) 优先级高的先计算；                      \* / 优先级大于 + -
- b) 优先级相同的自左向右计算；
- c) 当使用括号时从最内层括号开始计算。

操作/运算符的优先关系是确定的，见教材P53的表  
任何两个运算符 $op_1, op_2$ 之间关系只有三种： $> < =$

假设输入的是一个合法的表达式

限定4种双目操作符，一种括号 '(' , ')'



如何来做呢？

计算机来计算中缀表达式的值可用利用两个栈，

一个存运算符

一个存操作数

操作符 ch	#	(	^	*, /, %	+, -	)
isp (栈内)	0	1	7	5	3	8
icp (栈外)	0	8	6	4	2	1

同时还要利用运算符优先级比较。见P53.

注意P53表中，优先级与两个运算符出现先后顺序相关

同一个运算符在栈内栈外的优先级不同。 或用上表

该表可做成函数，输入为字符，内部利用case，输出优先级

**策略：**0、入栈界限符'#'

1、操作数先放到操作数栈N内保存

2、如是符op1，

比较O的栈顶，如比栈顶优先级高，入栈O

如op1优先级<栈顶op2，说明op2应该先计算，N连续出栈两次，O出栈，用这两个数进行op2的计算，入栈N

如果优先级相同，O出栈(是左右括号)

重复以上过程直到栈空。

N内的栈顶元素，即为表达式的值

实例：

$$5+(7-3)*2$$

根据以上思路，给定N,O两个栈，画一下整个求值过程。

请同学们自己在纸上先尝试画一下

算法伪码(余P53略有不同):

```
float evaluate_expression(){//也可一个字符串参数，即表达式
    init(N);init(O);push(O,'#');    //'#'表示表达式开始与结束
    c=getchar();
    while(!is_empty(O)){//栈不空就继续
        if(!is_op(c, OP)){push(c,N); c=getchar();}//操作数入栈
        else switch(compare_prior(get_top(O), c)){
            case '<':push(c, O); c=getchar();break;//栈顶低
            case '=':pop(op1, O); c=getchar();break;//括号和#
```

```
case '>': //注意这里没有getchar()  
    pop(op1, O); pop(b,N);pop(a,N);  
    push(compute(a,op1,b), N);  
    break;
```

```
}
```

```
}
```

```
return(get_top(N));
```

```
//假设表达式是正确表达式
```

实际上，有三种算数表达式：

中缀(infix)表达式

<操作数> <操作符> <操作数>，如  $A+B$ ；

前缀(prefix)表达式

<操作符> <操作数> <操作数>，如  $+AB$ ；

后缀(postfix)表达式

<操作数> <操作数> <操作符>，如  $AB+$ ；

其中，后缀表达式，即操作符在两个操作数的后面，也叫逆波兰表达式，这种逆波兰表示是波兰逻辑学家Jan Lukasiewicz于1950年左右发明的。

Infix Expression	Prefix Expression	Postfix Expression
$A + B$	$+ A B$	$A B +$
$A + B * C$	$+ A * B C$	$A B C * +$
$(A + B) * C$	$* + A B C$	$A B + C *$
$(A + B) * (C + D)$	$* + A B + C D$	$A B + C D + *$

发现点儿啥没？

- 1、操作数顺序不变，操作符位置不同
- 2、后两种无需括号

考虑计算：3 9 10 4-\*2/+20+

无需括号，不用考虑优先级，方便

1、大家用纸计算下其值

2、还原成中缀表达式

**后缀表达式求值：**

人算策略：从左到右找操作符及其前两个数，计算放原地

机算策略：数入栈，遇到操作符出栈2个数，运算结果入栈



过程：

顺序扫描表达式的每一项，根据它的类型做如下相应操作：

- a) 若该项是操作数，则将其入栈；
- b) 若该项是操作符 $\langle op \rangle$ ，则连续从栈中退出两个操作数 $Y$ 和 $X$ ，形成运算指令 $X\langle op \rangle Y$ ，并将计算结果入栈。

当表达式的所有项都扫描并处理完后，栈顶存放的就是最后的计算结果。

```
int ComputePostExp(char *s)//也可以改成从键盘输入字符
    for(i=0;s[i]!='\0';i++)    {
        if(IsNum(s[i])) Push(&S, s[i]);
        else {
            Pop(&S, &b);
            Pop(&S, &a);
            n = Compute(a, b, s[i]);
            Push(&S, n);    }
    }    GetTop(S, &v); return v;
} //Compute函数自行实现
```

问题来了---如何得到后缀表达式？中缀--->后缀表达式转换

## 算法策略：

对数字直接输出

对操作符则

如是“（”，进栈

如是“）”，则栈内元素依次出栈输出，直到“（”出栈

否则与栈顶元素进行优先级比较

该操作符优先级低，则出栈，直到不低于或成为空栈

该操作符入栈

那么，为什么这个策略是正确的呢？留给大家思考

例子：  $a + b * (c - d * e) - f / g$

转为后缀表达式：  $abcde*-*/+fg/-$

大家对照前面策略，画一下转换过程，注意如同级，则先出现的优先级高

编译器遇到公式，转换完以后，在程序实际运行时，利用后缀表达式求值，就相对简单也省时间了。

## 算法伪码

Status MidTrans2Post(char \*s)//也可以改成从键盘输入字符

//先声明或初始化一个栈S，以及其他变量

```
for(i=0;s[i]!='\0';i++) {
```

```
    if(IsNum(s[i])) printf("%c", s[i]);//IsNum函数自行编写
```

```
    else if(s[i]=='(') Push(&S, s[i]);
```

```
    else if(s[i]==')')
```

```
        while(1) {
```

```
            Pop(&S, &op);
```

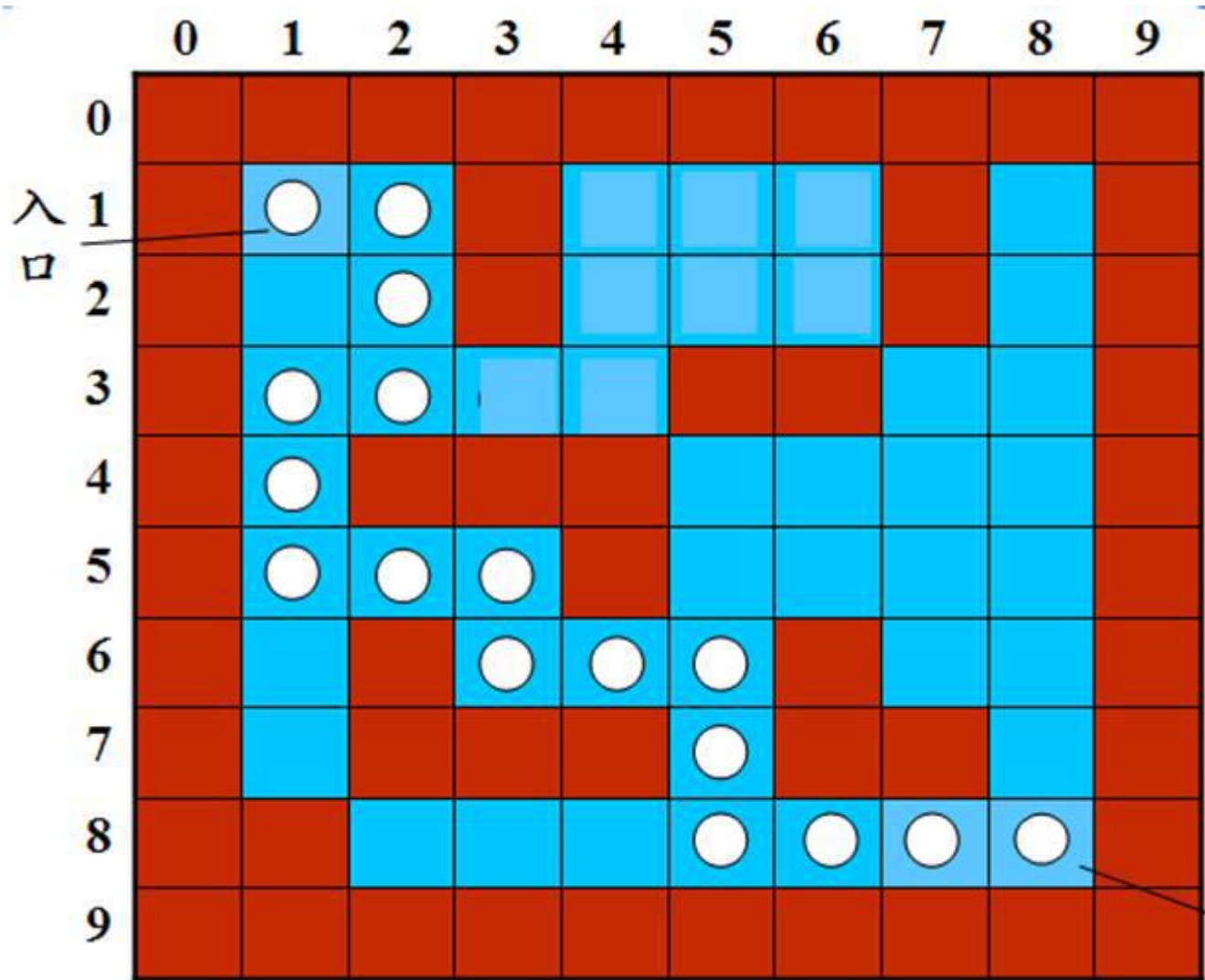
```
            if(op!='(') printf("%c", op);
```

```
            else break;}//两个括号之间的符均出栈
```

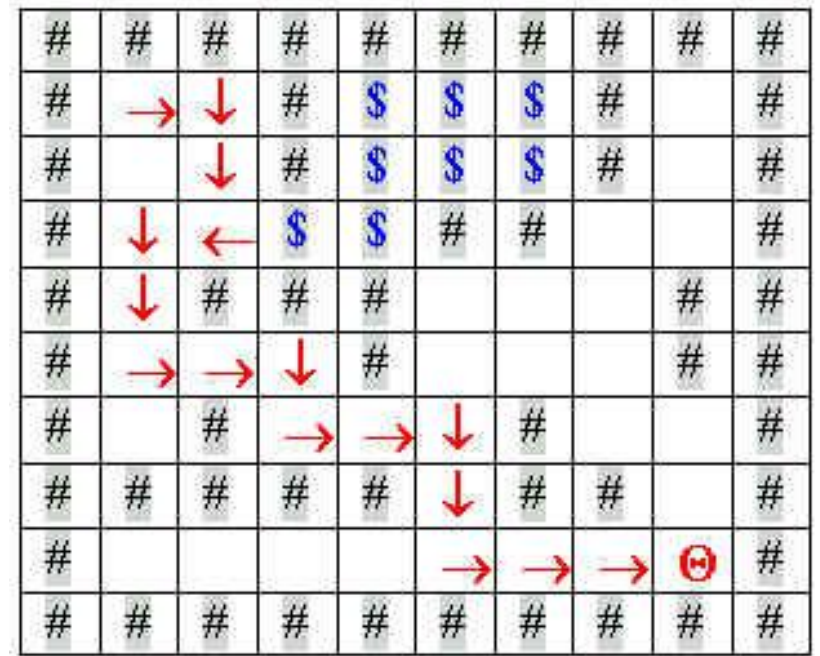
```
else {  
    if(!s_empty(S)) Push(&S, s[i]);  
    else{  
        GetTop(S, &op);  
        while(CompareOP(op, s[i]) &&!s_empty(S)){  
            Pop(&S, &op); printf("%c",op);GetTop(S, &op);}  
        Push(&S, s[i]);}  
    }
```

```
}while(!s_empty(S){Pop(&S, &op); printf("%c",op);}  
return OK;}//CompareOP函数自定义，前者大为1，否则为0
```

# 应用---迷宫求解



出口



开始就是墙的，不能走  
曾走不通地方，不能走  
除非走到无路，否则不能回头

“入口”到“出口”的简单路径（所经过的不重复通道方块）

## 数据表示与定义:

- 1) **迷宫**表示: 可用二维数组, 自行定义
- 2) **方格**: 坐标, 标记(墙标记#、探过不通标记\$, 当前路径标记-, 出口G, **空白格o**)
- 3) **当前路径 (栈)**: 路径中各方块的位置;
- 4) **行进下一方格**:按四个方向按一定次序试探相邻方格。

[illegible]



```
typedef struct{
    int x; int y;
}ElemType;
typedef struct{
    ElemType pos;//方格坐标 (x,y)
    char data;//方格标记
    int di;//方向标记
}GridType;
GridType maze[m][n];//迷宫数组， 可设为全局变量
```

## 行进下一方格的探索结果有三种可能:

- a. 探索方格为空白格，则：1、将该方格标记更改为-， 2、该位置压入路径栈
- b. 探索方格为出口，则1、该位置入路径栈，栈内即为路径通路。游戏结束
- c. 探索方格为其他标记，则继续顺次探索

如行进下一**方格**探索了所有4方向仍没有遇到空白格，当前**方格**标记改为\$，出栈，路径回退一步（**一直走到道路尽头，不撞南墙不回头**）

[illegible]

给定方格，给定方向，得到一个探索的方格pos。

di可取值1,2,3,4，依次表示：东，南，西，北，初值均为1。

ElemType NextGrid(ElemType pos)

```
{    if maze[pos].di==1
        return x+1,y;
        //当前pos坐标为x,y
        ....//其余自行完成
```

}//伪码

x-1, y-1	x, y-1	x+1, y-1
x-1, y	x, y	x+1, y
x-1, y+1	x, y+1	x+1, y+1

```
int GoStep(ElemType *pos){//伪码
    NewPos = NextGrid(*pos);
    if(maze[NewPos].data='o'){//空白格
        *pos = NewPos;
        return 1;
    }
    return 0;
}
//对某方格，试探走一步
//成功返回1， 否则返回0， 作为探索结果
```

```
Status MazePath(ElemType start, end){//伪码
    init_stack(S); push(S, start);
    if(maze[start].data!='o' && maze[start].data=='G') return FALSE;
    maze[start].data = '-';//初始工作完毕
    do{GetTop(S, &CurPos);//栈顶元素作为当前方格位置
        while(maze[CurPos].di<=4){//四个方向探索
            if(GoStep(&CurPos)) {//探索一步成功
                if(CurPos==end){push(S, CurPos);break;}
                push(S, CurPos); maze[CurPos].data='-';}
            else maze[CurPos].di++;//否则该方格方向++}
```

```
    if(maze[CurPos].di>4){//确实没有前进的路则
        pop(S, &CurPos);//退栈，回退一步
        maze[CurPos].data='$';//标记不通}
}while(!S.Is_empty()&&CurPos!=end);
if(S.Is_empty()) return 0;
return 1;
} //算法与教材P51不尽相同
//可最后打印整个迷宫，或中间即时显示迷宫
//兴趣同学自行实现，需要实现判断pos_type变量相等的函数
```

思考：为何用栈？直接用线性表不就完全可以嘛？

上机及作业：

利用栈解决

1) 数制转换问题

2) 括号匹配问题（一种括号上机，多种括号回去做）

3) 行编辑程序

4) 中缀转后缀

5) 后缀表达式求值

要求：1)，4)，5) 分别单建工程；

2),3) 建一个工程