



本节主题:

栈的应用1-表达式求值

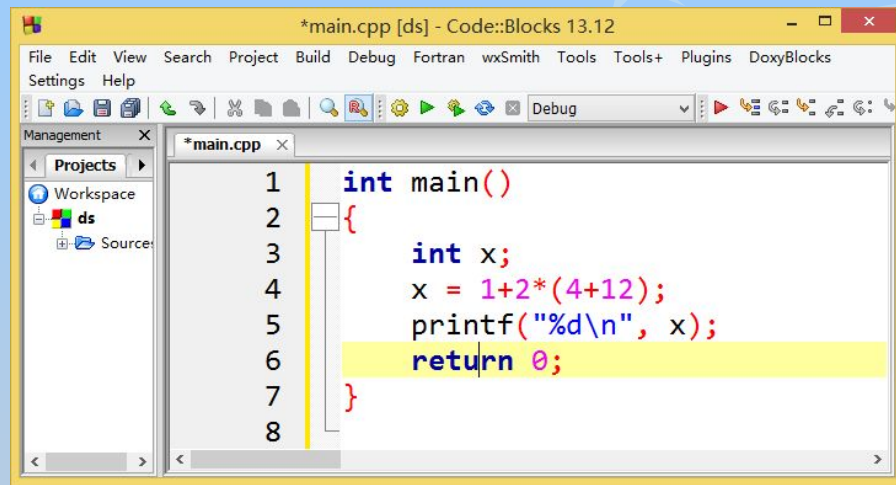
问题：表达式求值

需求

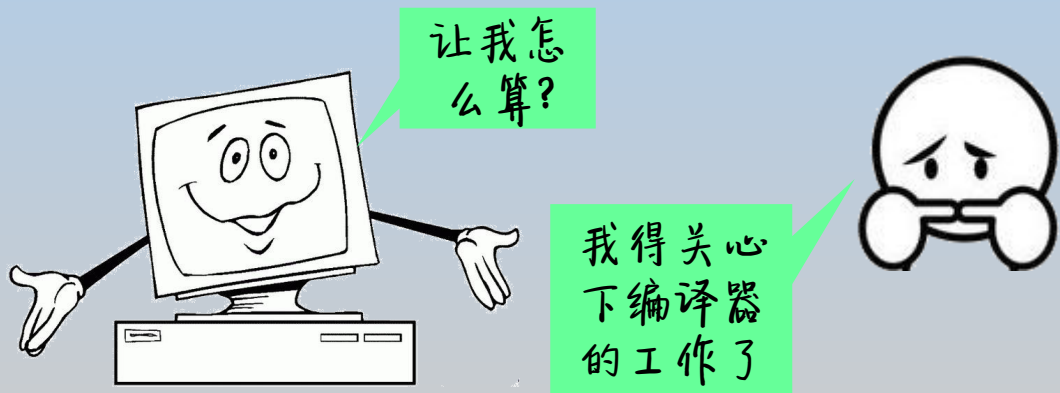
- 用户输入一个包含“+”、“-”、“*”、“/”、正整数和圆括号的合法数学表达式，计算该表达式的运算结果。

数据特点

- 算术表达式exp采用字符数组表示，其中只含“+”、“-”、“*”、“/”、正整数和圆括号
- 为方便，假设该表达式都是合法的数学表达式
- 例如，`exp="1+2*(4+12)"`



```
1  int main()
2  {
3      int x;
4      x = 1+2*(4+12);
5      printf("%d\\n", x);
6      return 0;
7  }
8
```



“表达式”的表达

例

1+2*3

1+2*(4+12)

运算规则

先乘除，后加减，从左到右计算，先括号内，后括号外

不仅要依赖运算符优先级，而且还要处理括号

表达式的三种标识方法

通式

Exp = S1 OP S2

运算规则

前缀表示法: OP S1 S2

中缀表示法: S1 OP S2

后缀表示法: S1 S2 OP

举例

100 + 300

+ 100 300

100 + 300

100 300 +

认识后缀表达式

再例

$$\text{Exp} = a \times b + (c - d / e) \times f$$

后缀式

$$a \ b \ \times \ c \ d \ e \ / \ - \ f \ \times \ +$$

☐ 让计算机求解表达式

☐ STEP 1：先将算术表达式转换成后缀表达式

☐ STEP 2：然后对该后缀表达式求值

☐ 表达式的三种表示法特点

☐ 操作数之间的相对次序**不变**；

☐ 运算符的相对次序**不同**；

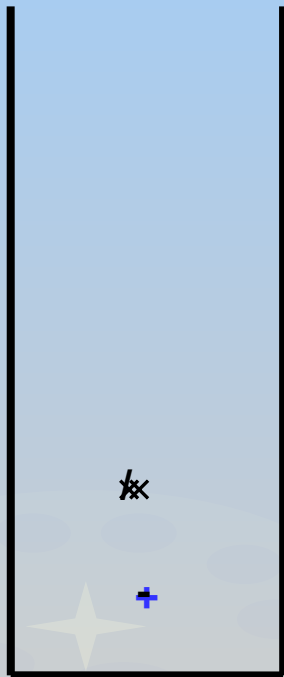
☐ 后缀表达式中已考虑了运算符的优先级，没有括号，只有操作数和运算符。



将算术表达式转换成后缀表达式

表达式 $\text{Exp} = a \times b + (c - d / e) \times f$
后缀式 $a b \times c d e / - f \times +$

运算符栈OP



- ☐ 用exp字符数组存储满足前面条件的算术表达式
- ☐ 对应后缀表达式存放在字符数组postexp中
- ☐ 用一个字符数组op作为栈

exp

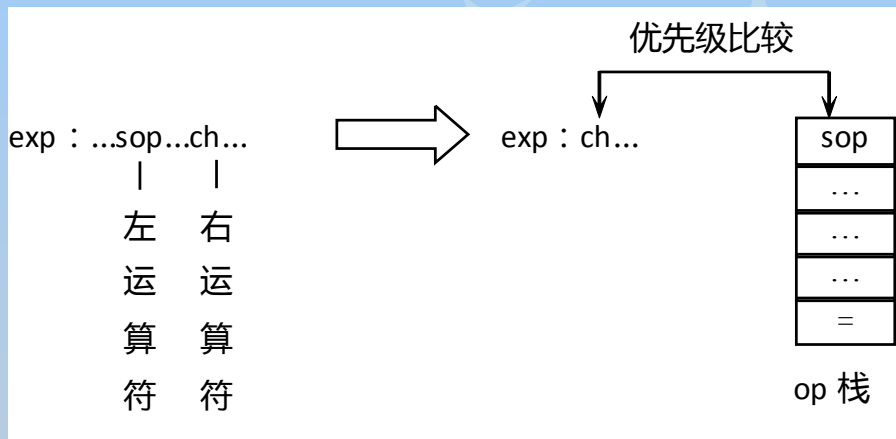
a	+	b	×	c	-	d	/	e	×	f	'\0'
---	---	---	---	---	---	---	---	---	---	---	------

ch	↑	ch	↑	ch	↑	ch	↑	ch	↑	ch	↑
----	---	----	---	----	---	----	---	----	---	----	---

postexp

a	b	c		d	e		f
---	---	---	--	---	---	--	---

运算符的优先级如何表示？



运算符	=	(+	-	*	/)
左 : lpri(ch)	0	1	3	3	5	5	6
右 : rpri(ch)	0	6	2	2	4	4	1

中缀表达式exp转后缀表达式postexp

初始化运算符栈op;

将 '=' 进栈;

从exp读取字符ch;

while (ch != '\0')

{

if (ch不为运算符)

将后续的所有数字均依次存放到postexp中,并以字符 '#' 标志数值串结束;

else

switch(Precede(op栈顶运算符, ch))

{

case '<': //栈顶运算符优先级低

将ch进栈; 从exp读取下字符ch; break;

case '=': //只有栈顶运算符为 '(', ch为 ')' 的情况

退栈; 从exp读取下字符ch; break;

case '>': //栈顶运算符应先执行,所以出栈并存放到postexp中

退栈运算符并将其存放到postexp中; break;

}

}

exp



postexp



运算符栈OP



“(56-20)/(4+2)”转换成后缀表达式的过程

exp	操作过程	op	postexp
	先将=进栈	=	
(56-20)/(4+2)	遇到ch为“(”,将此括号进栈op。	= (
56-20)/(4+2)	遇到ch为数字,将56存入postexp中,并插入一个字符“#”。	= (56#
-20)/(4+2)	遇到ch为“-”,由于op中“(”以前没有字符,则直接将ch进栈op中。	= (-	56#
20)/(4+2)	遇到ch为数字,将20#存入数组exp中。	= (-	56#20#
)/(4+2)	遇到ch为“)”,则将栈op中“(”以前的字符依次出栈并存入postexp中,然后将“(”删除。	=	56#20#-
/(4+2)	遇到ch为“/”,将ch进栈op中。	= /	56#20#-
(4+2)	遇到ch为“(”,将此括号进栈op中。	= /(56#20#-
4+2)	遇到ch为数字,将4#存入数组postexp中。	= /(56#20#-4#
+2)	遇到ch为“+”,由于op栈顶运算符为“(”,则直接将ch进栈op中。	= /(+	56#20#-4#
2)	遇到ch为数字,将2#存入postexp中。	= /(+	56#20#-4#2#
)	遇到ch为“)”,则将栈op中“(”以前的字符依次出栈并存放到postexp中,然后将“(”出栈。	= /	56#20#-4#2#+
	str扫描完毕,则将栈op中的所有运算符依次弹出并存放到postexp中,得到后缀表达式。	=	56#20#-4#2#+/

运算优先符处理模块

```
struct
{
    char ch; //运算符
    int pri; //优先级
}
lpri[] = {{ '=', 0 }, { '(', 1 }, { '*', 5 }, { '/', 5 }, { '+', 3 }, { '-', 3 }, { ')', 6 },
rpri[] = {{ '=', 0 }, { '(', 6 }, { '*', 4 }, { '/', 4 }, { '+', 2 }, { '-', 2 }, { ')', 1 };
```

//op1和op2运算符优先级的比较结果

```
int Precede(char op1, char op2)
{
    if (leftpri(op1) == rightpri(op2))
        return 0;
    else if (leftpri(op1) < rightpri(op2))
        return -1;
    else
        return 1;
}
```

```
int rightpri(char op) //求右运算符op的优先级
{
    int i;
    for (i = 0; i < MaxOp; i++)
        if (rpri[i].ch == op) return rpri[i].pri;
}
```

```
int leftpri(char op) //求左运算符op的优先级
{
    int i;
    for (i = 0; i < MaxOp; i++)
        if (lpri[i].ch == op) return lpri[i].pri;
}
```

```
bool InOp(char ch) //判断ch是否为运算符
{
    if (ch == '(' || ch == ')' || ch == '+' || ch == '-'
        || ch == '*' || ch == '/')
        return true;
    else
        return false;
}
```

将算术表达式exp转换成后缀表达式postexp

```
void trans(char *exp, char postexp[])
{
    struct    //定义运算符栈
    {
        char data[MaxSize];
        int top;
    } op;
    int i=0;
    op.top=-1;
    op.top++;
    op.data[op.top]='=';
    //处理exp中的每一个字符

    //此时exp扫描完毕,退栈到'='为止
    while (op.data[op.top]!='=')
    {
        postexp[i++]=op.data[op.top];
        op.top--;
    }
    postexp[i]='\0';
}
```

```
while (*exp!='\0')
{
    if (!InOp(*exp)) //判定为数字时
    {
        while (*exp>='0' && *exp<='9')
        {
            postexp[i++]=*exp;
            exp++;
        }
        postexp[i++]='#';
    }
    else //为运算符的情况
        switch(Precede(op.data[op.top], *exp))
        {
            case -1: //栈顶运算符的优先级低:进栈
                op.top++;
                op.data[op.top]=*exp;
                exp++; //继续扫描其他字符
                break;
            case 0: //只有括号满足这种情况
                op.top--; //将(退栈
                exp++; //继续扫描其他字符
                break;
            case 1: //退栈并输出到postexp中
                postexp[i++]=op.data[op.top];
                op.top--;
                break;
        }
}
```

后缀表达式求值

运算数栈

postexp

5 6 # 2 0 # - 4 # 2 # + /



+

2
~~20~~
~~56~~

- ❑ 从左到右读入后缀表达式
- ❑ 若读入的是一个操作数，就将它入数值栈
- ❑ 若读入的是一个运算符op，就从数值栈中连续出栈两个元素(两个操作数)，并将计算结果入数值栈；
- ❑ 对整个后缀表达式读入结束时，栈顶元素就是计算结果。

对后缀表达式postexp求值的算法

```
while (从postexp读取字符ch, ch!='\0')
```

```
{
```

若ch为数字，将后续的所有数字构成一个整数存放到数值栈st中。

若ch为“+”，则从数值栈st中退栈两个运算数，相加后进栈st中。

若ch为“-”，则从数值栈st中退栈两个运算数，相减后进栈st中。

若ch为“*”，则从数值栈st中退栈两个运算数，相乘后进栈st中。

若ch为“/”，则从数值栈st中退栈两个运算数，相除后进栈st中。

```
}
```

若字符串postexp扫描完毕，则数值栈op中的栈顶元素就是表达式的值。

若除数为零，则
提示相应的错误
信息。

后缀表达式“56#20#-4#2#+/”的求值过程

postexp	操作过程	st
56#20#-4#2#+/	遇到56#，将56进栈。	56
20#-4#2#+/	遇到20#，将20进栈。	56,20
-4#2#+/	遇到“-”，出栈两次，将 $56-20=36$ 进栈。	36
4#2#+/	遇到4#，将4进栈。	36,4
2#+/	遇到2#，将2进栈。	36,4,2
+/	遇到“+”，出栈两次，将 $4+2=6$ 进栈。	36,6
/	遇到“/”，出栈两次，将 $36/6=6$ 进栈。	6
	postexp扫描完毕，算法结束，栈顶的元素6即为所求。	

算法实现

```
float compvalue(char exp[])
{
    struct //定义数值栈
    {
        float data[MaxSize];
        int top;
    } st;
    float d;
    char ch;
    int t=0;
    st.top=-1;
    ch=exp[t];
    t++;
    //处理exp字符串

    return st.data[st.top];
}
```

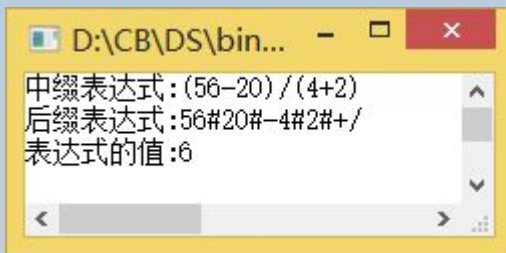
```
while (ch!='\0')          //exp字符串未扫描完时循环
{
    switch (ch)
    {
        case '+':
            st.data[st.top-1]=st.data[st.top-1]+st.data[st.top];
            st.top--; break;
        case '-':
            st.data[st.top-1]=st.data[st.top-1]-st.data[st.top];
            st.top--; break;
        case '*':
            st.data[st.top-1]=st.data[st.top-1]*st.data[st.top];
            st.top--; break;
        case '/':          //未表达st.data[st.top]==0的情形
            st.data[st.top-1]=st.data[st.top-1]/st.data[st.top];
            st.top--;
            break;
        default:
            //遇到数字字符时
            ch=exp[t];
            t++;
    }
}
```

```
d=0; //将数字字符转换成数值存放到d中
while (ch>='0' && ch<='9')
{
    d=10*d+ch-'0';
    ch=exp[t];
    t++;
}
st.top++;
st.data[st.top]=d;
```

设计求解程序

```
int main()
{
    char exp[]="(56-20)/(4+2)";
    char postexp[MaxSize];
    trans(exp,postexp);
    printf("中缀表达式:%s\n",exp);
    printf("后缀表达式:%s\n",postexp);
    printf("表达式的值:%g\n",compvalue(postexp));
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#define MaxOp 100
#define MaxSize 100
```



A screenshot of a Windows command prompt window. The title bar shows the path "D:\CB\DS\bin...". The window contains three lines of text: "中缀表达式:(56-20)/(4+2)", "后缀表达式:56#20#-4#2#+/", and "表达式的值:6".

```
D:\CB\DS\bin...
中缀表达式:(56-20)/(4+2)
后缀表达式:56#20#-4#2#+/
表达式的值:6
```