**实验3：利用栈将中缀表达式转换为后缀表达式并进行计算（4学时）**

**[问题描述]**

中缀表达式是最普通的一种书写表达式的方式，而后缀表达式不需要用括号来表示，计算机可简化对后缀表达式的计算过程，而该过程又是栈的一个典型应用。

**[实验目的]**

（1） 深入理解栈的特性。

（2） 掌握栈结构的构造方法。

**[实验内容及要求]**

1. 中缀表达式中只包含＋、－、×、/ 运算及（ 和 ）。
2. 可以输入任意中缀表达式,输入计算数据可为多位整数。
3. 显示中缀表达式及转换后的后缀表达式(为清楚起见，要求每输出一个数据用逗号隔开)。
4. 对转换后的后缀表达式进行计算。

**[测试数据]**

1. 6＋3\*(9-7)-8/2

转换后的后缀表达式为：

计算结果为：

1. (8-2)/(3-1)\*(9-6)

转换后的后缀表达式为：

**#include** <stdio.h>

**#include** <stdlib.h>

**#define** TRUE 1

**#define** FALSE 0

**#define** OK 1

**#define** ERROR 0

**#define** OVERFLOW **-**1

**#define** STACK\_INIT\_SIZE 100 // 初始大小

**#define** STACKINCREMENT 10   // 增量

**typedef** ***int*** *ElemType*;       // 数据类型

**typedef** ***struct***

{

*ElemType* **\***data; //堆栈数组,用top指向栈顶

***int*** top;        //栈顶指针

***int*** stacksize;

} *SqStack*;

*SqStack* s; // 全局堆栈

***int*** InitStack(*SqStack* &*s*) //堆栈初始化

{

*s*.data **=** **new** *ElemType*[STACK\_INIT\_SIZE];

**if** (**!***s*.data)

    exit(OVERFLOW); //存储分配失败

*s*.top **=** **-**1;       //栈空

*s*.stacksize **=** STACK\_INIT\_SIZE;

**return** OK;

}

***int*** Push(*SqStack* &*s*, *ElemType* *e*) //进栈操作

{

**if** (*s*.top **>=** *s*.stacksize **-** 1) // 堆栈满

  {

*ElemType* **\***p **=** (*ElemType* **\***)realloc(*s*.data, (*s*.stacksize **+** STACKINCREMENT) **\*** **sizeof**(*ElemType*)); //给堆栈分配增量内存空间

**if** (**!**p)

      exit(OVERFLOW); //存储分配失败

*s*.data **=** p;

*s*.stacksize **=** *s*.stacksize **+** STACKINCREMENT;

  }

*s*.data[**++***s*.top] **=** *e*;

**return** OK;

}

***int*** Pop(*SqStack* &*s*, *ElemType* &*e*) //出栈操作

{                                //若栈不空，则删除s的栈顶元素，用e返回其值，并返回OK；否则返回ERROR

**if** (*s*.top **==** **-**1)

**return** ERROR;

*e* **=** *s*.data[*s*.top**--**];

**return** *e*;

**return** OK;

}

*ElemType* GetTop(*SqStack* *s*) //取栈顶元素

{                          //栈已存在且不为空，返回栈顶元素

**if** (*s*.top **==** **-**1)

**return** ERROR;

**return** *s*.data[*s*.top];

}

***int*** StackEmpty(*SqStack* *s*) //判断栈是否为空

{

**if** (*s*.top **==** **-**1)

**return** OK;

**else**

**return** ERROR;

}

***int*** judge(***char*** *p1*, ***char*** *p2*) // 符号优先级判定

{

**if** (*p1* **==** '(' **||** *p2* **==** '(')

**return** 0;

**else** **if** ((*p2* **==** '-' **||** *p2* **==** '+') **&&** (*p1* **==** '\*' **||** *p1* **==** '/'))

**return** 0;

**else**

**return** 1;

}

***void*** trans(***char*** \**tmp*, ***char*** \**str*) //将中缀表达式转化为后缀表达式

{

***char*** **\***p **=** *tmp*, p1, p2;

*ElemType* e;

***int*** i **=** 0, j **=** 0, k **=** 0;

  InitStack(s);

**for** (i **=** 0; p[i] **!=** '\0'; i**++**)

  {

**if** (p[i] **<=** '9' **&&** p[i] **>=** '0')

    {

*str*[j**++**] **=** p[i];

*str*[j**++**] **=** ' ';

**continue**;

    }

**else**

    {

**if** (StackEmpty(s))

      {

        Push(s, p[i]);

**continue**;

      }

**else**

      {

**if** (p[i] **==** ')')

        {

**while** (GetTop(s) **!=** '(')

          {

*str*[j**++**] **=** Pop(s, e);

*str*[j**++**] **=** ' ';

          }

          Pop(s, e);

**continue**;

        }

      }

      p1 **=** p[i];

      p2 **=** GetTop(s);

**switch** (judge(p1, p2))

      {

**case** 1:

*str*[j**++**] **=** Pop(s, e);

*str*[j**++**] **=** ' ';

**for** (k **=** 0; k **<=** s.top;)

        {

**if** (**!**judge(p1, GetTop(s)))

**break**;

**else**

*str*[j**++**] **=** Pop(s, e);

*str*[j**++**] **=** ' ';

        }

        Push(s, p1);

**break**;

**case** 0:

        Push(s, p1);

**break**;

      }

    }

  }

**while** (**!**StackEmpty(s))

  {

*str*[j**++**] **=** Pop(s, e);

*str*[j**++**] **=** ' ';

  }

*str*[j] **=** '\0';

}

***int*** Calculate(***char*** \**p*) //计算后缀表达式

{

***int*** i, j;

***int*** c;

*ElemType* n, m, w;

**for** (i **=** 0; *p*[i]; i**++**)

  {

**if** (*p*[i] **<=** '9' **&&** *p*[i] **>=** '0')

    {

      Push(s, *p*[i]); //如果p为数则将其插入栈顶

**continue**;

    }

**if** (*p*[i] **==** '\*' **||** *p*[i] **==** '+' **||** *p*[i] **==** '/' **||** *p*[i] **==** '-')

    {

**switch** (*p*[i])

      {

**case** '+':

        c **=** Pop(s, n) **+** Pop(s, m) **-** '0';

        Push(s, c);

**break**;

**case** '-':

        c **=** Pop(s, n) **-** '0';

        j **=** Pop(s, m) **-** c;

        Push(s, j);

**break**;

**case** '\*':

        c **=** (Pop(s, n) **-** '0') **\*** (Pop(s, m) **-** '0') **+** '0';

        Push(s, c);

**break**;

**case** '/':

        c **=** Pop(s, n) **-** '0';

        j **=** (Pop(s, m) **-** '0') **/** c **+** '0';

        Push(s, j);

**break**;

      }

    }

  }

**return** Pop(s, w) **-** '0';

}

***int*** main()

{

***char*** s1[1024], s2[1024]; //中缀 && 后缀数组

  printf("请输入中缀表达式:");

  scanf("%s", **&**s1);

  trans(s1, s2); //转换

  printf("后缀表达式: ");

  puts(s2); //输出后缀表达式

  printf("结果为:%d", Calculate(s2)); //计算并输出结果

**return** 0;

} /\*

样例输入: 1+2\*3-4

后缀表达式:1 2 3 \* + 4 -

结果为:3

\*/