北京理工大學
Beijing Institute of Technology

本科实验报告

实验名称：简易计算器（栈）

课程名称：	数据结构与算法设计实验	实验时间：	2017/4/7
任课教师：	李岩	实验地点：	良乡机房 401
实验教师：	苏京霞	实验类型：	□ 原理验证 ■ 综合设计 □ 自主创新
学生姓名：	施念		
学号/班级：	1120161302/05011609	组　　号：	72
学　　院：	信息与电子学院	同组搭档：	
专　　业：	电子信息类	成　　绩：	

信息与电子学院
SCHOOL OF INFORMATION AND ELECTRONICS

1. 实验目的

通过模拟一个简单的计算器来进行+、-、*、/、%、^（乘方）等运算，从键盘上输入一算术表达式（一般为中缀表达式），计算出表达式的值。

2. 实验题目

编写程序，要求可对一实数算术表达式进行简单的数学运算。
可以识别带加减乘除等运算符及括号的中缀表达式。
a. 按照四则运算规则，求表达式的值。一般规则如下：
1）先括号内，再括号外。
2）先乘方，再乘除，后加减。
b. 同级运算从左到右顺序执行。
c．如表达式有误，应给出相应的提示信息。

3. 实验基础知识

首先是对栈进行建立、取顶元素、栈顶元素出栈以及元素入栈等操作。
然后是对后缀表达式的计算。

4. 概要分析

1) 构造字符型的栈结构体

代码如下：

```
typedef char Elemtype1;
typedef struct Stack1 {//定义一个字符型的栈指针
    Elemtype1 *base;
    Elemtype1 *top;
    int stacksize;
} Stack1;
```

2) 构造浮点型的栈结构体

```
typedef double Elemtype2;
typedef struct Stack2 {//定义一个浮点型的栈指针
   Elemtype2 *base;
   Elemtype2 *top;
   int stacksize;
} Stack2;
```

3) 函数模块

主函数：先调用 Change（char*M,char*B）,将中缀表达式转化为后缀表达式，然后调用 Calculate(char*B)函数，计算后缀表达式的值。

子函数：实现栈的各种操作以及对表达式的转化与计算

```
Elemtype1 Peak1(Stack1 &S)//取字符型栈的栈顶元素
int StackLength(Stack2 &S) //判断在栈的长度，栈空时栈长为 0
int InitStack1(Stack1 &S) //建立一个字符型的栈
int InitStack2(Stack2 &S) //建立一个浮点型的栈
```

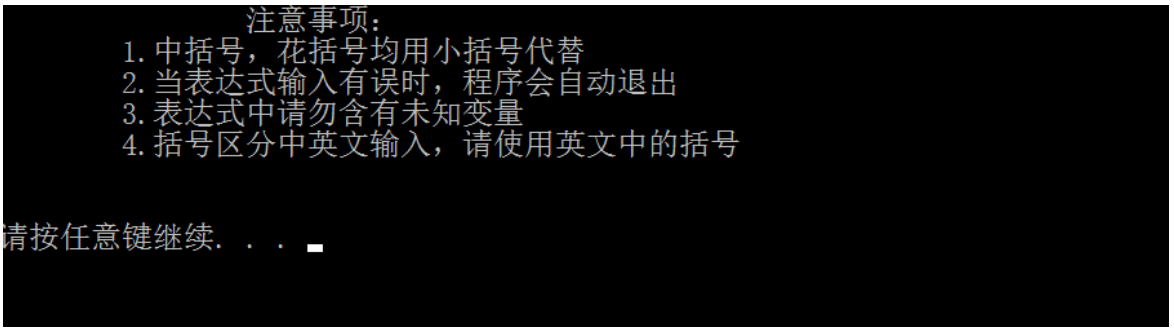
```
Elemtype1 Pop1(Stack1 &S) //字符型栈栈顶元素出栈并返回栈顶元素
Elemtype2 Pop2(Stack2 &S) //浮点型栈栈顶元素出栈并返回栈顶元素
int Push1(Stack1 &S, Elemtype1 e) //将字符型元素 e 入栈
int Push2(Stack2 &S, Elemtype2 e) //将浮点型元素 e 入栈
int Judge(char op)//两个作用：
    1.判断符号的优先级，根据优先级的高低，分别划分为0，1，2，3 ，
          数越大，优先级越高
    2.根据定义的数字判断字符是否为'+', '-', '*', '/', '%', '^' ，
          是则返回非零值
int Change(char *M, char *B)//将中缀表达式 M 转化为后缀表达式 B
double Calculate(char *B)//此函数用于计算后缀表达式，将结果返回
```

5. 调试分析

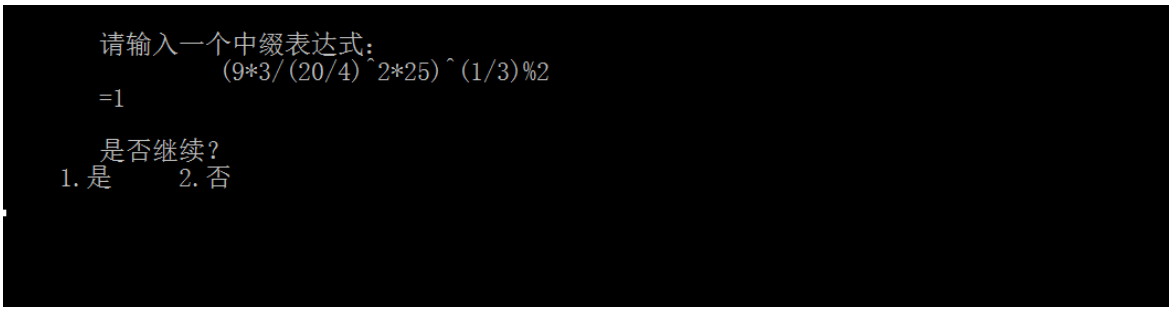
1) 因为此实验设计的栈类型不同，所以需要设计两个不同的栈，一个对字符进行操作，一个对浮点数进行操作。
2) 最一开始时要入栈一个元素'#',并把其的优先级调到最低，以便进行结束判断。
3) 要对不同的错误进行判断并显示出来。
4) 加减优先级为 1，乘除优先级为 2，乘方优先级为 3.
5) 将判断字符是否为可识别的运算符和计算可识别运算符的优先级两个函数结合，若运算符不可识别，返回 0，可识别就返回优先级。
6) 便于用户使用，在最一开始加入说明界面。
7) 采用%g 输出，去除无用的 0.

6. 测试结果

1) 说明界面



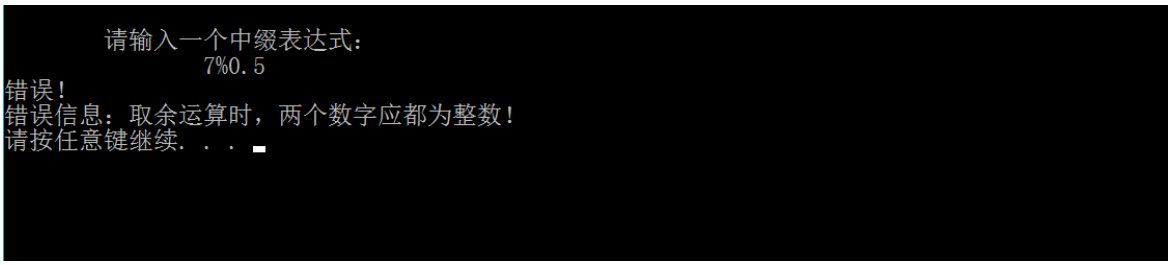
2) 计算结果



3) 错误提示 1

4) 错误提示 2

5) 错误提示 3



6) 错误提示 4

7. 附录（源代码）

```
//表达式的计算
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#define STACKINCREMENT 100
#define STACK_INIT_SIZE 20

typedef char Elemtype1;
typedef double Elemtype2;

//因为将中缀表达式转化为后缀表达式和计算后缀表达式所用的栈不一样（一个是字符型，一个是浮点型）
//所以栈的相关函数有两个，有后缀 1 的是字符型的，有后缀 2 的是浮点型
typedef struct Stack1 {//定义一个字符型的栈指针
    Elemtype1 *base;
```

```
    Elemtype1 *top;
    int stacksize;
} Stack1;


typedef struct Stack2 {//定义一个浮点型的栈指针
    Elemtype2 *base;
    Elemtype2 *top;
    int stacksize;
} Stack2;

Elemtype1 Peak1(Stack1 &S) {//取字符型栈的栈顶元素
    if (S.top == S.base) return 0;
    return *(S.top - 1);
}

int StackLength(Stack2 &S) {//判断在栈的长度，栈空时栈长为 0
    return S.top - S.base;
}

int InitStack1(Stack1 &S) {//建立一个字符型的栈
```

```
    S.base = (Elemtype1*)malloc(STACK_INIT_SIZE *
sizeof(Elemtype1));
    if (!S.base) {
        return 0;
    }
    S.stacksize = STACK_INIT_SIZE;
    S.top = S.base;
    return 1;
}

int InitStack2(Stack2 &S) {//建立一个浮点型的栈
    S.base = (Elemtype2*)malloc(STACK_INIT_SIZE *
sizeof(Elemtype2));
    if (!S.base) {
        return 0;
    }
    S.stacksize = STACK_INIT_SIZE;
    S.top = S.base;
    return 1;
}
```

```
Elemtype1 Pop1(Stack1 &S) {//字符型栈栈顶元素出栈并返回栈顶元素

    if (S.base == S.top) {
        return 0;
    }
    S.top--;
    return *S.top;
}

Elemtype2 Pop2(Stack2 &S) {//浮点型栈栈顶元素出栈并返回栈顶元素

    if (S.base == S.top) {
        return 0;
    }
    S.top--;
    return *S.top;
}

int Push1(Stack1 &S, Elemtype1 e) {//将字符型元素  e  入栈
```

```
    if (S.top - S.base >= S.stacksize) {
        S.base = (Elemtype1 *)realloc(S.base, (S.stacksize +
STACKINCREMENT) * sizeof(Elemtype1));
        if (!S.base)exit(OVERFLOW);
        S.top = S.base + S.stacksize;
        S.stacksize += STACKINCREMENT;
    }
    *S.top++ = e;
    return 1;
}

int Push2(Stack2 &S, Elemtype2 e) {//将浮点型元素 e 入栈
    if (S.top - S.base >= S.stacksize) {
        S.base = (Elemtype2 *)realloc(S.base, (S.stacksize +
STACKINCREMENT) * sizeof(Elemtype2));
        if (!S.base)exit(OVERFLOW);
        S.top = S.base + S.stacksize;
        S.stacksize += STACKINCREMENT;
    }
    *S.top++ = e;
    return 1;
```

```
}

int Judge(char op)
{    //两个作用：
    //1.判断符号的优先级，根据优先级的高低，分别划分为 0，1，2，3 ，数越大，优先级越高
    //2.根据定义的数字判断字符是否为'+', '-', '*', '/', '%'，'^' ,是则返回非零值

    switch (op)
    {
    case '+':
    case '-':
        return 1;
    case '*':
    case '/':
    case '%':
        return 2;
    case '^':
        return 3;
    default:
```

```
        return 0;
    }
}

int Change(char *M, char *B)
{
    //将中缀表达式 M 转化为后缀表达式 B
    Stack1 S;//S 为运算符
    InitStack1(S);//建立一个字符型栈
    Push1(S, '#');//栈底首先推入元素 #，作为优先级最低的一个运算符
    int i = 0, j = 0;//分别以 i ，j 来表达中缀和后缀表达式中字符的位置
    char ch;//ch 移动指示中缀表达式中的字符
    ch = M[i];
    while (ch != '\0')
    {
        if (ch == ' ')
            ch = M[++i];
        else if (ch == '(')
        {//左括号直接进栈，等待右括号
```

```
            Push1(S, ch);
            ch = M[++i];
        }
        else if (ch == ')')
        {//将与上一个左括号之间的运算符直接写入后缀表达式
            while (Peak1(S) != '(')
                B[j++] = Pop1(S);
            Pop1(S);//删除此时栈顶的  '('
            ch = M[++i];
        }
        else if (Judge(ch))
        {
            char temp;//定义中间变量
            temp = Peak1(S);
            while (Judge(ch) <= Judge(temp))
            {//将所有优先级大于等于  ch  的运算符写入后缀表达式并出栈
                B[j++] = temp;
                Pop1(S);
                temp = Peak1(S);
            }
```

```
            Push1(S, ch);
            ch = M[++i];
        }
        else if ((ch >= '0'&& ch <= '9') || ch == '.')
        {
            while ((ch >= '0'&& ch <= '9') || ch == '.')
            {
                B[j++] = ch;
                ch = M[++i];
            }
            B[j++] = ' ';
        }
        else
        {
            printf("错误！\n 错误信息：表达式中有未能识别的字符！
\n");
            system("pause");
            exit(1);
        }
    }
    ch = Pop1(S);//利用  ch  将栈中剩余字符全部写入后缀表达式中
```

```
    while (ch != '#')
    {
        if (ch == '(')
        {//当中缀表达式处理完毕，但栈中仍然含有未配对的 ( 时，说明输入的中缀表达式括号不匹配
            printf("错误！\n 错误信息：表达式括号不匹配！\n");
            system("pause");
            exit(1);
        }
        else {
            B[j++] = ch;
            ch = Pop1(S);
        }

    }
    B[j++] = '\0';//给后缀表达式加上一个结束标志
                  //函数结束
}

double Calculate(char *B)
{//此函数用于计算后缀表达式，返回值为计算结果
```

```
Stack2 S;
InitStack2(S);//将字符转化为数字并储存在浮点型栈中 ，按照规则进行运算
char ch;//依次指向后缀表达式的字符
int i = 0;//用来储存后缀表达式的下标
double integer = 0.0, decimal = 0.0;//分别用来存储表达式中的整数部分和小数部分以及结果
double temp, temp1;//temp 用来处理除法，乘方和减法时先弹出除数（指数，减数） ,temp1 和 c 特用于取余运算
int c;
while (B[i])
{
    if (B[i] == ' ')
    {//对空格不进行处理
        i++;
        continue;
    }
    else
    {
        switch (B[i])
        {
```

```
case '+':integer = Pop2(S) + Pop2(S);//加法
    i++;
    break;
case '-':temp = Pop2(S);//减法
    integer = Pop2(S) - temp;
    i++;
    break;
case '*':integer = Pop2(S)*Pop2(S);//乘法
    i++;
    break;
case '/':temp = Pop2(S);//除法
    if (temp == 0.0)
    {//除数为 0
        printf("错误！\n 错误信息：除数不可以为零！\n");
        system("pause");
        exit(1);
    }
    else
    {
        integer = Pop2(S) / temp;
    }
```

```
            i++;
            break;
        case '%':temp = Pop2(S);//取余
            temp1 = Pop2(S);
            if ((int)temp == temp && (int)temp1 == temp1) {
                integer = (int)temp1 % (int)temp;
                i++;

            }
            else {
                printf("错误！\n 错误信息：取余运算时，两个数字
应都为整数！\n");
                system("pause");
                exit(1);
            }
            break;
        case '^':temp = Pop2(S);//乘方
            integer = pow(Pop2(S), temp);
            i++;
            break;
        default:
```

```
{//此时一定是数，不是运算符
    integer = 0.0;
    decimal = 0.0;
    while (B[i] >= '0'&&B[i] <= '9')
    {//如果 B[i] 为整数部分，则一直进行循环累加
        integer = integer*10.0 + B[i] - '0';//储存整数部分
        i++;
    }
    if (B[i] == '.') {
        i++;
        double n = 0.1;//n 为小数的位数
        while (B[i] >= '0'&&B[i] <= '9')
        {//如果存在小数点且小数点后面有小数部分，则一直循环累加
            decimal = decimal + (B[i] - '0')*n;
            n = n*0.1;
            i++;
        }
    }
    integer = integer + decimal;//将整数与小数部分相加到入栈
```

```
            }
            }
            Push2(S, integer);//将数或者运算结果入栈
        }

    }
    if (!StackLength(S))
    {//判断栈是否空，正确的话栈里应该只有结果，空则说明表达式错误
        printf("错误！\n 错误信息：表达式有误！\n");
        system("pause");
        exit(1);
    }
    integer = Pop2(S);//结果出栈

    if (!StackLength(S))
    {//结果出栈后栈应为空
        return integer;
    }
    else
```

```
    {
        printf("错误！\n 错误信息：表达式有误！\n");
        system("pause");
        exit(1);
    }
}

int main(void)
{
    printf("\t\t 注意事项：\n");
    printf("\t1.中括号，花括号均用小括号代替\n");
    printf("\t2.当表达式输入有误时，程序会自动退出\n");
    printf("\t3.表达式中请勿含有未知变量\n");
    printf("\t4.括号区分中英文输入，请使用英文中的括号\n\n\n");
    system("pause");
    system("cls");
    int flag = 1;
    int choice;
    char a[100];
    char b[100];
    double result;
```

```
while (flag) {
    printf("\n\t 请输入一个中缀表达式：\n\t\t");
    fflush(stdin);
    gets(a);
    Change(a, b);
    result = Calculate(b);
    printf("\t=%g\n\n", result);
    printf("\t 是否继续？\n");
    printf("          1.是          2.否\n");
    scanf("%d", &choice);
    while (choice<1 || choice>2) {
        printf("输入错误，请重新输入（1-2）：");
        fflush(stdin);
        scanf("%d", &choice);
    }
    switch (choice) {
    case 1:system("cls");break;
    case 2:flag = 0;
        break;
    }
```

}

}