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：	2017/4/14
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

由键盘输入一算术表达式，以中缀形式输入，试编写程序将中缀表达式转换成一棵二叉表达式树，通过对该的后序遍历求出计算表达式的值。

2. 实验题目

a．要求对输入的表达式能判断出是否合法。不合法要有错误提示信息。
b．将中缀表达式转换成二叉表达式树。
c．后序遍历求出表达式的值

3. 实验基础知识

一棵表达式树，它的树叶是操作数，如常量或变量名字，而其他的结点为操作符。

a．建立表达式树。二叉树的存储可以用顺序存储也可用链式存储。当要创建二叉树时，先从表达式尾部向前搜索，找到第一个优先级最低的运算符，建立以这个运算符为数据元素的根结点。注意到表达式中此运算符的左边部分对应的二叉绔为根结点的左子树，右边部分对应的是二叉绔为根结点的右子树，根据地这一点，可用递归调用自己来完成对左右子树的构造。

b．求表达式的值。求值时同样可以采用递归的思想，对表达式进行后序遍历。先递归调用自己计算左子树所代表的表达式的值，再递归调用自己计算右子树代表的表达式的值，最后读取根结点中的运算符，以刚才得到的左右子树的结果作为操作数加以计算，得到最终结果。

4. 概要分析

1) 二叉树的结构体

```
struct node {//定义一个节点
    char data;//储存数据
    float opnd;
    struct node *lchid;//左孩子
    struct node *rchild;//右孩子
};
typedef struct node TREE
```

2) 构造含有浮点型和字符型的结构数组

```
struct OperType {
  char optr;
  float opnd;
}array[20];
```

3) 函数模块

主函数：当输入退出选项时才结束程序。

子函数：判断、运算、遍历以及创建等操作

```
int jud(char stack[], int n);//对左括号进行扫描
TREE *create(TREE *T, char d[]);//根据 T，d[]建立二叉链表
```

```
char *Change(char str1[]);//将 str 转化为逆后缀表达式并返回地址
void PosterOrderTraverse(TREE *e);//进行后序遍历二叉树
int IsOperand(char ch);//判断是否是数字
int IsOperator(char op);//判断是否是运算符
int judge(char S[100]);//对输入的表达式进行错误判定
float Calculate(float a, float b, char c);//对 a 和 b 进行 c 种运算
int Without(char c);//判断没有括号的运算符
```

4) 全局变量说明：

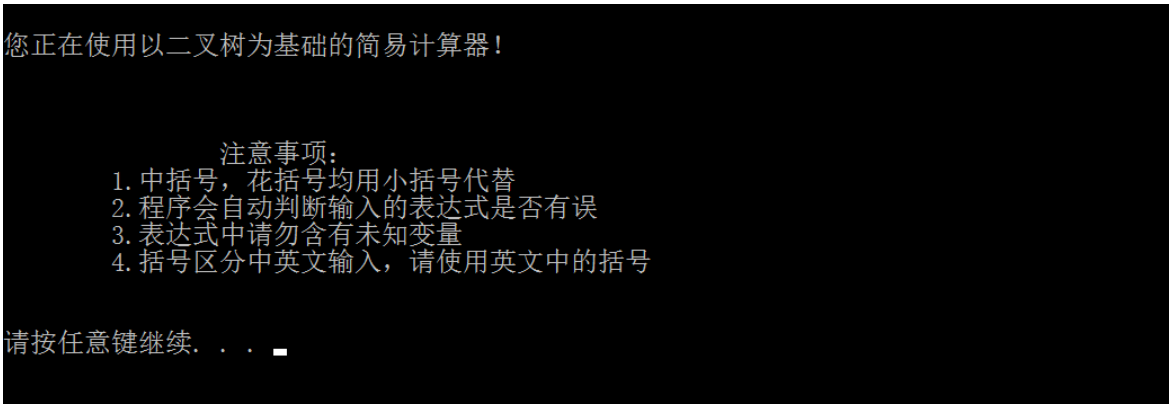
```
int w, flag;//全局变量
float result = 0;//计算结果，便于访问
```

5. 调试分析

1) 要对不同的错误进行判断并显示出来。
2) 判断函数有两个，一个是判断表达式是否正确，一个是进行运算时所需要的判断
3) 便于用户使用，在最一开始加入说明界面。
4) 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

7. 附录（源代码）

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>

struct node {//定义一个节点
 char data;//储存数据
 float opnd;
 struct node *lchid;//左孩子
 struct node *rchild;//右孩子
};

typedef struct node TREE;

struct OperType {
 char optr;
 float opnd;
}array[20];
```

```
int w, flag;//全局变量
float result = 0;//计算结果，便于访问

int jud(char stack[], int n);//对左括号进行扫描
TREE *create(TREE *T, char d[]);//根据 T，d[]建立二叉链表
char *Change(char str1[]);//将 str 转化为逆后缀表达式并返回地址
void PosterOrderTraverse(TREE *e);//进行后序遍历二叉树
int IsOperand(char ch);//判断是否是数字
int IsOperator(char op);//判断是否是运算符
int judge(char S[100]);//对输入的表达式进行错误判定
float Calculate(float a, float b, char c);//对 a 和 b 进行 c 种运算
int Without(char c);//判断没有括号的运算符

int main()
{
 printf("\n\n 您正在使用以二叉树为基础的简易计算器！ \n\n");
 printf("\n\n\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.程序会自动判断输入的表达式是否有误\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");

 int choice = 1;
 char str[100];
 TREE *T;
 char *b, a[100], *t;
 int i;
 while (choice) {
        system("cls");
        flag = 0;
        T = (TREE *)malloc(sizeof(TREE));
        printf(" \n\n 请输入您要计算的表达式:\n\t\t");
        scanf("%s", str);
        b = Change(str);
        if (b != 0)
        {
                for (t = b;*t != '\0';t++);
                i = 0;
                do
                {
                        t--;
```

```
            a[i] = *t;
            i++;
        } while (t != b);
        a[i] = '\0';//完成反序后缀式的逆向
        T = create(T, a); //create()函数创建二叉链表
        PosterOrderTraverse(T);//后序遍历进行计算
        printf("\t\t 计算结果为\n");
        printf("\t\t%g", result); //输出结果
        free(T);
        printf("\n\n\t\t 是否继续？\n");
        printf("\t\t1.是\n");
        printf("\t\t2.否\n");
        scanf("%d", &choice);
        if (choice != 1 && choice != 2) {
            printf("\n\t 输入错误，请重新输入（1-2）：");
            fflush(stdin);
            scanf("%d", choice);
        }
        if (choice == 2) return 0;
    }

 }

 return 0;
}

int Without(char c)//判断是否为运算符，是运算符返回 1
{
 if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^' || c == '%')
    return 1;
 else
    return 0;
}

float Calculate(float a, float b, char c)//进行各种运算函数
{
 int d, e, f;
 switch (c)
 {
 case '+':return a + b;
    break;
 case '-':return a - b;
```

```
        break;
 case '*':return a*b;
        break;
 case '/':
        if (b == 0) {
            printf("\n\n 除数不能为 0\n\n");
            system("pause");
            exit(0);
        }
        return a / b;
        break;
 case '^':d = (int)a;
        e = (int)b;
        f = pow(a, b);
        return (float)f;
        break;
 case '%':

        d = (int)a;
        e = (int)b;
        if (d != a || e != b) {
            printf("\n 取余运算时数字不可以为小数\n");
            system("pause");
            exit(0);
        }
        f = d%e;
        return (float)f;
 default:printf("运算符错误!\n");
        system("pause");
        exit(0);
        break;
 }
}

int judge(char S[100])     //判断输入的表达式是否正确
{
 char check;
 int error = 0, lb = 0, rb = 0, numofoperand = 0, numofoperator = 0;
 for (int m = 0; m < strlen(S); m++)
 {
        check = S[m];
        if (IsOperand(check)) //判断是否是数字
```

```
        {
            if (check == '.')//判断浮点型数据是否正确
            {
                if (!(S[m - 1] >= '0'&&S[m - 1] <= '9') && (S[m + 1] >= '0'&&S[m + 1] <=
'9'))
                {
                    error++;
                    printf("表达式错误！\n");
                    system("pause");
                    exit(0);
                }
            }
            numofoperand++;
        }
        else if (IsOperator(check)) //判断是否是运算符
        {
            if (check == ')')
            {
                rb++;
                if (rb > lb)
                {
                    error++;
                    printf("表达式错误！\n");
                    system("pause");
                    exit(0);
                }
                if (IsOperator(S[m + 1]) && (S[m + 1] == '+' || S[m + 1] == '-' || S[m + 1] ==
'*' || S[m + 1] == '/' || S[m + 1] == ')' || S[m + 1] == '^' || S[m + 1] == '%'))
                {
                    numofoperator++;
                    m++;
                    if (S[m] == ')')
                        rb++;
                }
                else if (IsOperator(S[m + 1]) || IsOperand(S[m + 1]))
                {
                    error++;
                    printf("表达式错误！\n");
                    system("pause");
                    exit(0);
                }
            }
```

```
            else if (check == '(')
            {
                lb++;
                if (IsOperator(S[m + 1]) && S[m + 1] == '(' || S[m + 1] == '-')//左括号右边只能是数字或者"-"号
                {
                    m++;
                    m++;
                    lb++;
                }
                else if (IsOperator(S[m + 1]))
                {
                    error++;
                    printf("表达式错误！\n");
                    system("pause");
                    exit(0);
                }
            }
            else
            {
                numofoperator++;
                if (IsOperator(S[m + 1]) && S[m + 1] == '(')
                {
                    m++;
                    lb++;

                }
                else if (IsOperator(S[m + 1]))
                {
                    error++;
                    printf("表达式错误！\n");
                    system("pause");
                    exit(0);
                }
            }
        }
        else
        {
            error++;
            printf("表达式错误！\n");
            system("pause");
            exit(0);
```

```
        }
    }
    if ((error == 0) && (lb == rb) && (numofoperand != 0) && (numofoperator != 0))
        return 1;
    else
        return 0;
}

int IsOperator(char op)    //判断一个字符是否是运算符
{
    if (op == '+' || op == '-' || op == '*' || op == '/' || op == '(' || op == ')' || op == '^' || op == '%')
        return 1;
    else
        return 0;
}

int IsOperand(char ch)     //判断是否是数字
{
    if (((ch >= '0') && (ch <= '9')) || (ch == '.'))
        return 1;
    else
        return 0;
}

void PosterOrderTraverse(TREE *e)//进行后序遍历计算
{
    if (e->lchid != NULL&&e->rchild != NULL)
        if ((e->lchid->data == '+' || e->lchid->data == '-' || e->lchid->data == '*' ||
e->lchid->data == '/' || e->lchid->data == '^' || e->lchid->data == '%' || e->rchild->data == '+'
|| e->rchild->data == '-' ||
            e->rchild->data == '*' || e->rchild->data == '/' || e->rchild->data == '^' ||
e->rchild->data == '%') ||
            (e->lchid->data >= 'A'&&e->lchid->data <= 'Z'&&e->rchild->data >=
'A'&&e->rchild->data <= 'Z'))
        {
            PosterOrderTraverse(e->lchid);
            PosterOrderTraverse(e->rchild);
            e->opnd = Calculate(e->lchid->opnd, e->rchild->opnd, e->data);
            e->data = 'a';
            result = e->opnd;
        }
}
```

```
char *Change(char str1[])//将运算式转换成逆后缀序列
{
 w = 0;
 char *a, *b;
 bool jud(char stack[], int n);
 char str[100];
 char S[100];
 char stack[100];
 char ch;
 int flag = 1;
 int zs;
 int i = 0, j = 0, t = 0, top = 0, k = 0, l = 0;
 if (judge(str1) == 1)
 {
        for (i = 0;str1[i] != '\0';i++)
        {
                b = &str1[i];
                if (Without(str1[i]) == 0 && (i == 0 && (Without(str1[0]) == 0)) || (i != 0 &&
Without(str1[i - 1]) == 1 && Without(str1[i]) == 0))
                {
                        array[k].opnd = atof(b);
                        array[k].optr = k + 65;
                        str[l] = array[k].optr;
                        k++;l++;
                }
                else if (Without(str1[i]) == 1)
                {
                        str[l] = str1[i];l++;
                }
        }
        array[k].optr = '\0';
        str[l] = '\0';
        i = 0;k = 0;l = 0;
        zs = strlen(str);
        str[zs] = '#';
        ch = str[i];
        while (ch != '#')
        {
                if (ch >= 'A'&&ch <= 'Z')
                {
                        S[t] = ch;
```

```
        t++;
    }
    else if (ch == '(')
    {
        top++;
        stack[top] = ch;
        k++;
    }
    else if (ch == ')')
    {
        if (top != 0)
        {
            if (jud(stack, top))
            {
                while (stack[top] != '(')
                {
                    S[t] = stack[top];
                    top--;
                    t++;
                }
                top--;
                l++;
            }
        }
    }
    else if (ch == '+' || ch == '-')
    {
        while (top != 0 && stack[top] != '(')
        {
            S[t] = stack[top];
            top--;
            t++;
        }
        top++;
        stack[top] = ch;
    }
    else if (ch == '*' || ch == '/' || ch == '%')
    {
        while (stack[top] == '*' || stack[top] == '/' || stack[top] == '%')
        {
            S[t] = stack[top];
            top--;
```

```
                t++;
            }
            top++;
            stack[top] = ch;
        }
        else if (ch == '^')
        {
            while (stack[top] == '^')
            {
                S[t] = stack[top];
                top--;
                t++;
            }
            top++;
            stack[top] = ch;
        }
        i++;
        ch = str[i];
    }
    if (flag != 0)
    {
        while (top != 0)
        {
            S[t] = stack[top];
            t++;
            top--;
        }
    }
    S[t] = '\0';
    a = &S[0];
    return a;
 }
 else
    return 0;
}

TREE *create(TREE *T, char d[])//建立二叉链表
{
 int i;
 if (d[w] >= 'A'&&d[w] <= 'Z')
 {
    T->data = d[w];
```

```
        for (i = 0;array[i].optr != '\0';i++)
        {
            if (array[i].optr == d[w])
            {
                T->opnd = array[i].opnd;
                break;
            }
        }
        T->lchid = NULL;
        T->rchild = NULL;
        w++;
    }
    else
    {
        if (flag == 0)flag = 1;
        else
            T = (TREE *)malloc(sizeof(TREE));
        (T)->data = d[w];
        w++;
        T->lchid = (TREE *)malloc(sizeof(TREE));
        T->rchild = (TREE *)malloc(sizeof(TREE));
        T->rchild = create((T)->rchild, d);
        T->lchid = create((T)->lchid, d);
    }
    return T;
}

int jud(char stack[], int n)//扫描左括号
{
    int i;
    for (i = 0;i < n;i++)
    {
        if (stack[i] == '(')
        {
            return 1;
            break;
        }
    }
}
```