

**课程设计报告**

**课程设计题目：** 表达式求值

**姓 名** 刘凌杰

**学 号 201720180411**

**班 级** 1721804

**指导教师**

**2018年12月19日**

一．问题分析

要求能输入一个带“（ ）”的任意多位实数的表达式。

要求：

1、有合理的提示,可进行加减乘除，求余，求次方计算。

2、有非法数据的判断；输入中只允许出现“1，2,3,4,5,6,7,8,9,0，(,),+,-,\*,/，.，%，^”

1. 能够进行多次计算；先将中缀转化成后缀，在利用栈实现逐步运算
2. 加减优先级 < 乘除求余优先级 < 求次方优先级

二．结构分析

辅助容器

1. 存放输入的中缀表达式的字符串string1
2. 存放输出的后缀表达式的字符串string2
3. 存放运算字符的栈stack1
4. 存放代数的栈stack2

功能函数

1. 判断运算符优先级int priority(char e)
2. 中缀转后缀表达式char\* change(char\* string)

3．二目计算函数float caculate(float a,float b,char ch)

4.字符数转化成float数型float transformation(char \*ch,int n)

5.多次计算实现四则运算float All\_caculate(char \*string)

6.输入表达式语句检查bool check(char \*ch)

大致关系

首先bool check(char \*ch)函数检查输入语句string1格式是否正确，再用char\* change(char\* string)将前缀表达式转换成后缀表达式string2，最后用float All\_caculate(char \*string)实现四则多次运算

其中int priority(char e)函数用于将前缀表达式转换成后缀表达式过程中判断入栈stack1字符的优先级，从而达到顺序入栈; float transformation(char \*ch,int n)函数将字符数字换成数型数字；float caculate(float a,float b,char ch)函数用于多次计算时逐个将遇到的两个数和相匹配的一个运算符进行计算返回结果；

1. 实现流程分析

1.工程建立

1）头文件stack.h

◆包含栈类的代码：

typedef float Status;

template <typename DataType>

**class Stack**

{

private:

int top;

DataType \*data;

public:

Stack()

{

this->data=(DataType\*)malloc(SIZE\*sizeof(DataType));

if(!this->data)

{

cout<<"内存申请失败！"<<endl;

return ;

}

this->top=-1;

}

Stack(int n)

{

this->data=(DataType\*)malloc(n\*sizeof(DataType));

if(!this->data)

{

cout<<"内存申请失败！"<<endl;

return ;

}

this->top=-1;

}

~Stack()

{

free(this->data);

this->data=NULL;

this->top=-1;

}

bool emptyStack(); //判断栈是否为空

bool fullStack(); //判断栈是否满

DataType topStack(); //获取栈顶元素

Status inputStack(DataType e); //入栈

Status popStack(DataType &e); //出栈

};

//判断栈是否为空

template <typename DataType>

bool Stack<DataType>::emptyStack()

{

return this->top==-1;

}

//判断栈是否满

template <typename DataType>

bool Stack<DataType>::fullStack()

{

return this->top==SIZE-1;

}

//获取栈顶元素

template <typename DataType>

DataType Stack<DataType>::topStack()

{

if(this->emptyStack())

return ERROR;

return this->data[this->top];

}

//入栈

template <typename DataType>

Status Stack<DataType>::inputStack(DataType e)

{

if(this->fullStack())

data=(DataType\*)realloc(data,(top+SIZE)\*sizeof(DataType));

this->data[++this->top]=e;

return OK;

}

//出栈

template <typename DataType>

Status Stack<DataType>::popStack(DataType &e)

{

if(this->emptyStack())

}

◆头文件及宏定义：

#include<iostream>

#include<cmath>

#define SIZE 100

#define MAX 100

#define OK 0;

#define ERROR -1;

#define C '|'

typedef float Status;

2）源文件arimethtic.cpp

◆代码流程

1)输入表达式string1

2)检查表达式语句

//检查语句

[**bool check(char \*ch)**](#检查语句)**；**

{

int i=0,j,k=0;

int n=0; //记录括号数

char number[10]; //存放数字检验

int len=strlen(ch);

while(ch[i]!='\0')

{

// cout<<"ch["<<i<<"]"<<ch[i];

switch(ch[i])

{

case'(':

case')':

if(n>=0&&ch[i]=='(')

n++;

else

n--;

case'+':

case'-':

case'\*':

case'/':

case'%':

case'^':

case'.':

case'0':

case'1':

case'2':

case'3':

case'4':

case'5':

case'6':

case'7':

case'8':

case'9':i++;

break;

default:

//judge=9;

cout<<"第"<<i+1<<"个字符为非法字符！"<<endl;

return false;

/\* //去除无关字符

for(j=i;j<len-1;j++)

{

ch[j]=ch[j+1];

}

ch[len-1]='\0';

len--;

\*/

}

}

//检验括号数是否为偶数

if(n!=0)

{

judge=0;

cout<<"括号不匹配！"<<endl;

return false;

}

//检验字符串首字符和尾字符

if(!( (ch[0]>='0'&&ch[0]<='9'|| ch[0]=='(') && (ch[len-1]>='0'&&ch[len-1]<='9' || ch[len-1]==')' ) ) )

{

//judge=1;

if(!(ch[0]>='0'&&ch[0]<='9'|| ch[0]=='('))

cout<<"首字符应为0-9或 “（”！"<<endl;

else

cout<<"尾字符应为0-9或 “）”！"<<endl;

return false;

}

//增加该代码修复bug

else

{

if(ch[0]=='(')

if(ch[1]=='+' || ch[1]=='-' || ch[1]=='\*' || ch[1]== '/' || ch[1]=='%'|| ch[1]=='^' )

{

cout<<"出错位置 2"<<endl;

cout<<"左括号后不能存在运算符"<<endl;

return false;

}

if(ch[len-1]==')')

if(ch[len-2]=='+' || ch[len-2]=='-' || ch[len-2]=='\*' || ch[len-2]== '/' || ch[len-2]=='%'|| ch[len-2]=='^' )

{

cout<<"出错位置 "<<len-1<<endl;

cout<<"右括号前不能存在运算符"<<endl;

return false;

}

}

for(i=1;i<len-1;i++)

{

bool whether=(ch[i]=='+' || ch[i]=='-' || ch[i]=='\*' || ch[i]== '/' || ch[i]=='%'|| ch[i]=='^');

bool whether\_follow=( ch[i+1]=='+' || ch[i+1]=='-' || ch[i+1]=='\*' || ch[i+1]== '/' || ch[i+1]=='%'|| ch[i+1]=='^');

bool whether\_pioneer=( ch[i-1]=='+' || ch[i-1]=='-' || ch[i-1]=='\*' || ch[i-1]== '/' || ch[i-1]=='%'|| ch[i-1]=='^' );

if(whether)

{

//检验两个运算符是否相邻

if(whether\_follow)

{

//judge=2;

cout<<"出错位置 "<<i+2<<endl;

cout<<"两个运算符不能相邻！"<<endl;

return false;

}

}

if(ch[i]=='(')

{

//检验左括号前是否为数字

if(ch[i-1]>='0' && ch[i-1]<='9')

{

//judge=3;

cout<<"出错位置 "<<i<<endl;

cout<<"左括号前不能存在数字！"<<endl;

return false;

}

//检验左括号后是否存在运算符

if(whether\_follow)

{

//judge=4;

cout<<"出错位置 "<<i+2<<endl;

cout<<"左括号后不能存在运算符"<<endl;

return false;

}

}

if(ch[i]==')')

{

//检验右括号后是否为数字

if(ch[i+1]>='0'&&ch[i+1]<='9')

{

//judge=5;

cout<<"出错位置 "<<i+2<<endl;

cout<<"右括号后不能存在数字！"<<endl;

return false;

}

//检验右括号前是否存在运算符

if(whether\_pioneer)

{

//judge=6;

cout<<"出错位置 "<<i<<endl;

cout<<"右括号前不能存在运算符！"<<endl;

return false;

}

}

//检验小数点周围是否存在数字字符

if(ch[i]=='.')

{

if((ch[i-1]<'0'||ch[i-1]>'9') && (ch[i+1]<'0'||ch[i+1]>'9'))

{

//judge=7;

cout<<"出错位置 "<<i+1<<endl;

cout<<"不能单独存在一个小数点！"<<endl;

return false;

}

}

//检验数中是否存在一个以上的小数点

bool whether\_number=(ch[i]<='9'&&ch[i]>='0'||ch[i]=='.');

if(whether\_number)

number[k++]=ch[i];

else

{

int num=0;

for(j=0;j<k;j++)

{

if(number[j]=='.')

num++;

}

if(num>1)

{

//judge=8;

cout<<"一个数存在多个小数点！"<<endl;

return false;

}

k=0;

}

}

return true;

}

3)将前缀表达式转化成后缀

//中缀表达式转后缀表达式

[**char\* change(char\* string)**](#中缀转后)

{

int i=0; //表达式下标

int j=0; //后缀下标

char ch;

Stack<char> S; //栈

char\* suffix=new char[MAX];

while(string[i]!='\0')

{

if(string[i]<='9' && string[i]>='0'||string[i]=='.')

{

suffix[j++]=string[i];

//将一组数字分隔标识

if((string[i+1]>'9' || string[i+1]<'0')&&string[i+1]!='.')

suffix[j++]=C;

}

else

{

if(S.emptyStack() || string[i]=='(')

{

S.inputStack(string[i]);

}

else

{

if(string[i]==')')

{

while(S.topStack()!='(')

{

S.popStack(suffix[j++]);

}

S.popStack(ch);

}

else

{

while(priority( S.topStack())>=priority( string[i]) )

{

S.popStack(suffix[j++]);

}

S.inputStack (string[i]);

}

}

}

i++;

}

//全部出栈

while(!S.emptyStack())

{

S.popStack(suffix[j++]);

}

suffix[j]='\0';

return suffix;

}

//判断优先级（辅助函数）

[**int priority(char e)**](#优先级)

{

int p=0;

switch(e)

{

case '-':

case '+':p=1;break;

case'%':

case'/':

case'\*':p=2;break;

case'^':p=3;

}

return p;

}

4)多次计算实现四则运算

//四则运算

[**float All\_caculate(char \*string)**](#四则运算)

{

Stack<float> S\_num(20);

float num,n,m; //存放数字字符转化的数

int i=0;

int j; //存放数字字符的下标

char ch[10];//存放数字字符

while(string[i]!='\0')

{

//找出数字入栈

while(string[i]!='+' && string[i]!='-' && string[i]!='\*' && string[i]!='/' && string[i]!='%' &&string[i]!='\0'&&string[i]!='^' )

{

j=0;

while((string[i]>='0'&&string[i]<='9')|| string[i]=='.')

{

ch[j++]=string[i++];

}

ch[j]='\0';

num=transformation(ch,j);

S\_num.inputStack(num);

i++; //跳过分隔符C

}

//计算结果入栈

while(string[i]=='+' || string[i]=='-' || string[i]=='\*' || string[i]=='/' || string[i]=='%'||string[i]=='^')

{

S\_num.popStack(n);

S\_num.popStack(m);

S\_num.inputStack(caculate(m,n,string[i++]));

}

}

return S\_num.topStack();

}

//二目计算（辅助函数）

[**float caculate(float a,float b,char ch)**](#二目运算)

{

switch(ch)

{

case'+':return a+b;

case'-':return a-b;

case'\*':return a\*b;

case'/':return a/b;

case'%':return int(a)%int(b);

case'^':return pow(a,b);

default:

return -1;

}

}

//字符数转化成float（辅助函数）

[**float transformation(char \*ch,int n)**](#转化数型)

{

int i=0;

float sum=0;

int flag=n; //默认小数点在整数后面

//求小数点的下标

while(ch[i]!='\0')

{

if(ch[i]=='.')

{

flag=i;

break;

}

i++;

}

i=0;

while(ch[i]!='\0')

{

if(ch[i]!='.')

{

if(i<flag) //整数部分

sum=sum+(ch[i]-48)\*pow(10,flag-i-1);

else //小数部分

{

sum=sum+(ch[i]-48)\*pow(10,flag-i);

// cout<<ch[i]-48<<" "<<flag-i<<" "<<pow(10,flag-i)<<endl;

}

}

i++;

}

return sum;

}

5)将输出结果

◆main函数

[**int main()**](#main函数)

{

char \*ch,string[MAX];

while(1)

{

cout<<"请输入表达式（退出按e）："<<endl;

cin>>string;

if(string[0]=='e'||string[0]=='E')

break;

if(check(string))

{

ch=change(string);

cout<<"="<<All\_caculate(ch)<<endl;

}

system("pause");

system("cls");

}

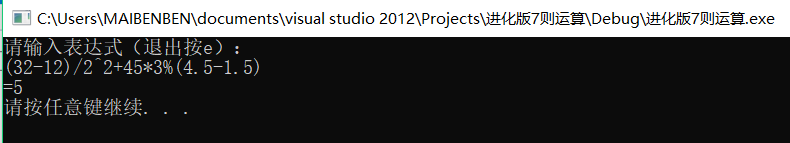
// cout<<string<<endl;

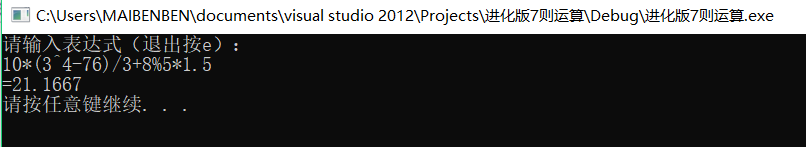
// cout<<transformation(string,strlen(string))<<endl;

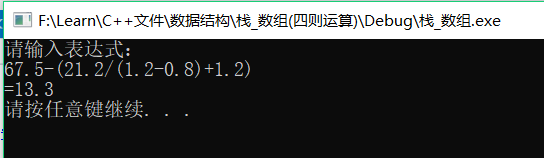
return 0;

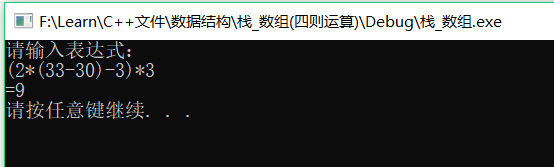
}

2．过程演示









四．个性功能介绍

★可实现float类型小数计算，并且加入了求余”%”的计算和求次方”^”的计算，例：34%5=4,2^5=32.

1.如果小数的整数部分为“0”时，可忽略输入“0”

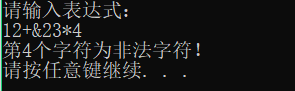
例“0.245”可输入为”.245”

2.如果输入小数点后无数字字符，也可看成整数计算

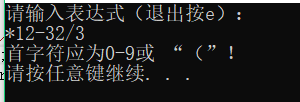
例“245.”将会化为“245”

★检验输入语句，如输入错误会提示错在何处

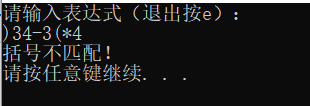
1.出现无关字符



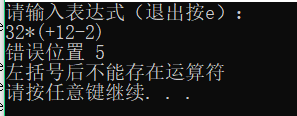
2.首尾字符不能出现运算符



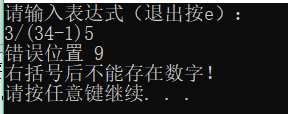
3.括号必须匹配 且右括号不能出现在其匹配左括号左边。



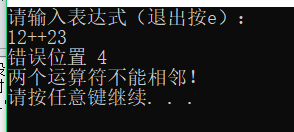
1. 左括号右边 和 右括号左边不能出现运算符



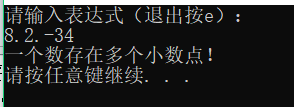
1. 左括号左边 和 右括号右边不能出现数字字符



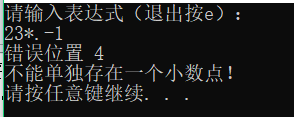
1. 两运算符不能相邻



1. 一个数中不能出现一个以上的小数点



1. 小数点周围至少出现一个数字字符



五.课程设计小结

本次课程实验再次锻炼了我对写程序的严谨性，同时让我认识了做程序不仅仅是序写出来，更要考虑用户的使用体验。

一方面，我反复调试修改程序，只为让程序更人性化和代码健壮性达到最强。另一方面，我在不断修改错误和思考中对代码的熟悉程度再次加深，也对写代码产生了浓厚的有趣。就拿这次而言，我遇到将数字字符转化成数型，中缀如何转化成后缀，如何考虑语句准确性的周全，如何输入小数也能计算等一系列问题，并且我一一都解决了。并且在本次实验中，我在四则运算的基础上加入了求余“%”和求次方“^”的功能，发现方法相似，“^”的优先级最高，“%”的优先级和乘除相同，这样使功能更全，这使我非常有成就感。我希望以后能够不断学精，自己带领团队做出一项伟大的项目。