项目说明文档

数据结构课程设计

——算术表达式

作 者 姓 名： 谢宇翔

学 号： 1951708

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc28252)

[1.1 背景分析 1](#_Toc29855)

[1.2 功能分析 1](#_Toc6177)

[2 设计 1](#_Toc23720)

[2.1 数据结构设计 1](#_Toc19484)

[2.2 类结构设计 2](#_Toc10259)

[2.3 成员与操作设计 2](#_Toc12518)

[2.4 系统设计 3](#_Toc11577)

[3 实现 3](#_Toc27915)

[3.1 中序表达式转后序表达式功能的实现 3](#_Toc25508)

[3.1.1 功能流程图 3](#_Toc8278)

[3.1.2 功能核心代码 5](#_Toc185)

[3.2 计算功能的实现 9](#_Toc29784)

[3.2.1 功能流程图 9](#_Toc4759)

[3.2.2计算功能核心代码 9](#_Toc26168)

[4 测试 15](#_Toc21286)

[4.1 功能测试 15](#_Toc15388)

[4.2 边界测试 15](#_Toc31850)

[4.2.1 无输入数据 15](#_Toc27296)

[4.2.2 单单一个数 16](#_Toc28816)

[4.2.3 除数为0 16](#_Toc4917)

[4.2.4 大于10的整数 16](#_Toc23385)

[4.3 出错测试 17](#_Toc19046)

[4.3.1 缺少右括号 17](#_Toc24515)

[4.3.2 缺少左括号 17](#_Toc3409)

[4.3.3 缺少末尾等号 17](#_Toc15399)

# 1 分析

## 背景分析

算术表达式是我们日常生活中经常遇到的，依靠计算机的快速计算能力，我们可以快速地判断一个表达式是否正确，如果正确也可以快速地计算出表达式的值，可以大大方便了人们的计算。

## 1.2 功能分析

从键盘上输入中缀算数表达式，包括括号，计算出表达式的值

程序对所有输入的表达式作简单的判断，如表达式有错，能给出适当的提示。支持包括加减，乘除取余，乘方和括号等操作符，其中优先级是等于<括号<加减<乘除取余<乘方

能处理单目运算符：+或-。

# 2 设计

## 2.1 数据结构设计

由于计算机并不会像人们一样自动区分算术的优先级，当我们输入一个中序表达式之后，为了让计算机能够“读懂”式子的意思，我们需要通过栈将中序表达式修改为后序表达式，再对后序表达式进行运算，就可以简单方便许多。再设计cal类进行算术表达式的运算并输出。

## 2.2 类结构设计

栈类Stack，存储中序表达式，后进先出的结构，普通类cal，存储后序表达式，计算表达式的值。

## 2.3 成员与操作设计

**栈类(Stack)**

**私有成员：**

int top;//栈顶指针

int maxsize;

**公有操作：**

Stack(int sz);//构造函数

void push(double& x);//压入

bool pop(double& x);//弹出

bool gettop(double& x);//取顶值

bool isempty();//判断栈空

bool isfull();//判断栈满

int getsize();//获取元素个数

void overflow();//栈溢出操作

void makeempty()//清空栈

{

top = -1;

}

double\* elements;//储存数据

**普通类（cal）**

**私有成员：**

Stack s;

**公有操作：**

void run(char array[]);//运行表达式求解操作

void clear();//清空

**私有操作：**

void addoperand(double value);//把数据压入栈中

bool get2operand(double& left, double& right);//从栈中取出2个数

void dooperator(char op);//对2个数进行算数操作

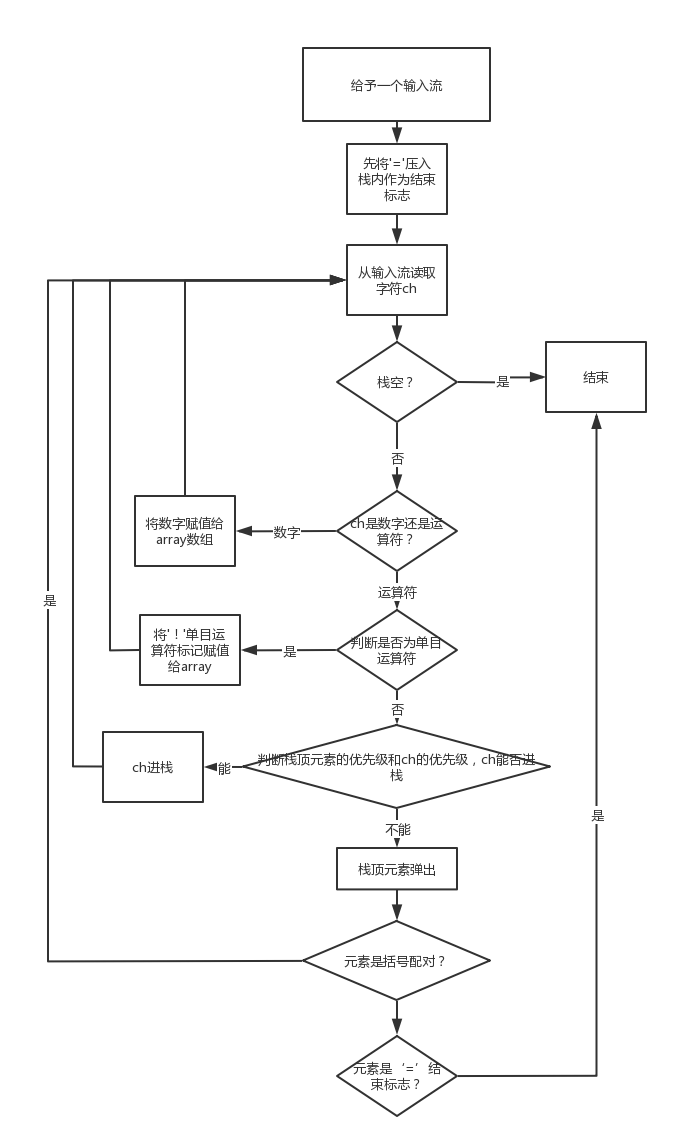
## 2.4 系统设计

首先键盘输入一串中序算术表达式，在main函数里转换成后续表达式并储存在array数组中，cal类内对array数组中的后序表达式进行求解，cal类内也包含一个栈s，对后序表达式里的数字和符号也是后进先出原则，重复读取array里的元素并计算，得出最后的答案结束

# 3 实现

## 3.1 中序表达式转后序表达式功能的实现

### 3.1.1 功能流程图



### 3.1.2 功能核心代码

必要函数:

bool isdight(char ch)//判断是否是数字

{

if (ch >= '0' && ch <= '9')

return true;

return false;

}

int isp(char ch)//栈内的各个字符的权值

{

switch (ch)

{

case'=':

return 0;

case'(':

return 1;

case'\*':

case'/':

case'%':

return 5;

case'+':

case'-':

return 3;

case')':

return 9;

case'^':

return 7;

default:

cout << "错误" << endl;

return-1;

}

}

int icp(char ch)//栈外的各个字符的权值

{

switch (ch)

{

case'=':

return 0;

case'(':

return 9;

case '^':

return 6;

case'\*':

case'/':

case'%':

return 4;

case'+':

case'-':

return 2;

case')':

return 1;

default:

cout << "错误" << endl;

return-1;

}

}

核心代码:

cout << "请输入中序算术表达式，以=结尾！" << endl;

stack <char>s;

char array[50];//存储后序表达式

int i = 0;

char ch = '=', ch1, op;

bool isfirst = 1;//单目运算符的判断

s.push(ch); cin.get(ch);//先把‘#’进栈，然后cin读数据

while (s.empty() == false)

{

if (isdight(ch))

{

isfirst = 0;

array[i] = ch;

i++;

while (isdight(cin.peek()))

{

cin.get(ch);

array[i] = ch;

i++;

}

array[i] = ' ';

i++;

cin.get(ch);

}

else if(ch!='+'&&ch!='-'&&ch!='\*'&&ch!='/'&&ch!='('&&ch!=')'&&ch!='^'&&ch!='%'&&ch!='=')

{

cout << "错误的符号！" << endl;

if (ch == '\n')

isworry = 2;

else

isworry = 1;

break;

}

else

{

ch1 = s.top();//取栈顶元素

if ((ch == '-' || ch == '+') && (isfirst == 1))//是单目运算符之后的数字

{

array[i] = '!';//单目运算符标志进array数组

i++;

array[i] = ch;//加号减号进数组

i++;

array[i] = ' ';

i++;

cin.get(ch);

}

else

{

if (ch == '(')//重置单目运算符的判定

isfirst = 1;

if (isp(ch1) < icp(ch))//新输入的运算符ch优先级高

{

s.push(ch);//进栈

cin.get(ch);//读入下一个

}

else if (isp(ch1) > icp(ch))//优先级低

{

op = s.top();//读栈顶元素

s.pop();//出栈

array[i] = op;//写入array里

i++;

array[i] = ' ';

i++;

//cout << op;

}

else

{

op = s.top();//优先级相等要么是‘=’，要么括号配对

s.pop();

if (op == '(')cin.get(ch);

}

}

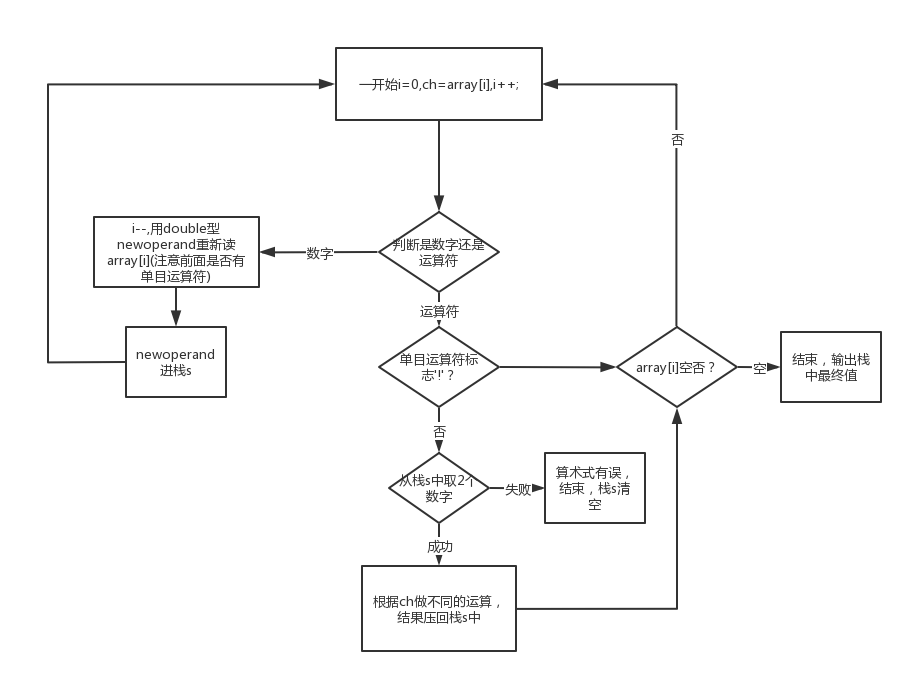
}

}

array[i] = '=';

## 3.2 计算功能的实现

### 3.2.1 功能流程图



### 3.2.2计算功能核心代码

**必要函数:**

bool cal::dooperator(char op)

{

double left, right, value;

bool result;

result = get2operand(left, right);//取操作数

if (result == true)//取成功

{

switch (op)

{

case'+':

value = left + right;

s.push(value);

break;

case'-':

value = left - right;

s.push(value);

break;

case'\*':

value = left \* right;

s.push(value);

break;

case'/':

if (fabs(right - 0.0) < 0.0001)

{

cout << "除数为0，计算错误！" << endl;

clear();

return 0;

}

else

{

value = left / right;

s.push(value);

break;

}

case'%':

value = (int)left % (int)right;

s.push(value);

break;

case '^':

value = pow(left, right);

s.push(value);

break;

}

}

else

{

clear();//不成功报错退出

return 0;

}

return 1;

}

bool cal::get2operand(double& left, double& right)

{

if (s.isempty() == true)

{

cout << "缺少右操作数!" << endl;

return false;

}

s.pop(right);

if (s.isempty() == true)

{

cout << "缺少左操作数!" << endl;

return false;

}

s.pop(left);

return true;

}

void cal::addoperand(double value)

{

s.push(value);

}

**核心函数：**

void cal::run(char array[])

{

int i = 0;//i循环array读数（array是存储着后序表达式的算式）

bool minus = 0;//判断是不是﹣单目运算符，然后对数据取相反数

bool isfirst = 0;//判断是不是单目运算符

bool well = 1;//判断除数为0的情况

char ch; double newoperand;

while (ch = array[i], i++, ch != '=')//每次从array里读出一个ch，i++

{

switch (ch)

{

case '(':

cout << "缺少右括号！" << endl;

return;

case ')':

cout << "缺少左括号！" << endl;

return;

case ' ':

break;

case'!'://ch=！是单目运算符的标志

isfirst = 1;

break;

case'\*':

case'/':

case'^':

case'%':

well=dooperator(ch);//做运算

if (well == 0)

return;

break;

case'+':

if (i == 1 || isfirst == 1)//单目运算符不做运算

{

isfirst = 0;

break;

}

else//不是单目就运算

{

dooperator(ch);

break;

}

case'-':

if (i == 1 || isfirst == 1)//原理同+，

{

minus = 1;//多了个minus，让读进来的数字取相反数

isfirst = 0;

break;

}

else

{

dooperator(ch);

break;

}

default:

i--;//现在数字是char型，i--，让array[i]重新指向数字，因为后面要读double型

if (minus == 1)//取相反数

{

double t=0;

while (array[i] >= '0' && array[i] <= '9')

{

t \*= 10;

t+=((double)array[i] - '0');

i++;

}

newoperand = -t;

minus = 0;

}

else

{

double t = 0;

while (array[i] >= '0' && array[i] <= '9')

{

t \*= 10;

t+=((double)array[i] - '0');

i++;

}

newoperand = t;

}

addoperand(newoperand);//数字进栈

}

}

cout << (double)s.elements[0];//运算完毕，输出计算结果。

}

# 4 测试

## 4.1 功能测试

**测试用例：**

-2×（3+5）+2^3/4=

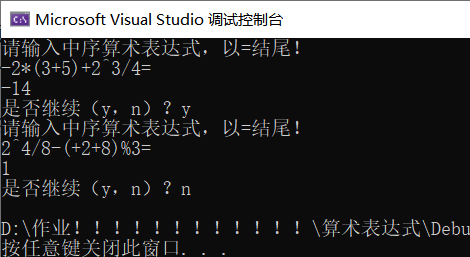
2^4/8-(+2+8)%3=

**预期结果：**

-14

1

**实验结果：**



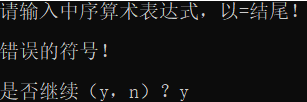
## 4.2 边界测试

### 4.2.1 无输入数据

**测试用例：**无输入数据(直接输入回车)

**预期结果：**给出错误提示，程序运行正常不崩溃。

**实验结果：**

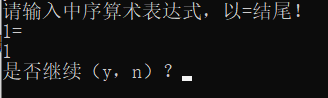


### 4.2.2 单单一个数

**测试用例：1=**

**预期结果：**程序正常运行，不崩溃。

**实验结果：**

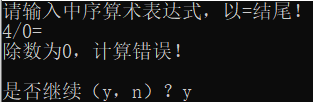


### 4.2.3 除数为0

**测试用例：**4/0=

**预期结果：**程序正常运行，不崩溃。

**实验结果：**

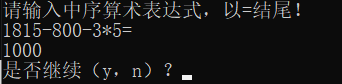


### 4.2.4 大于10的整数

**测试用例：**1815-800-3\*5=

**预期结果：**程序正常运行，不崩溃。

**实验结果：**



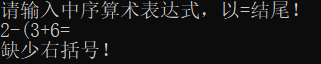
## 4.3 出错测试

### 4.3.1 缺少右括号

**测试用例：2-(3+6=**

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

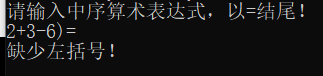


### 4.3.2 缺少左括号

**测试用例：2+3-6)=**

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**



### 4.3.3 缺少末尾等号

**测试用例：2+3**

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**