项目说明文档

数据结构课程设计

——中缀算术表达式求解

作 者 姓 名： 罗佳瑞

学 号： 1952528

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](#_Toc25484)

[1.1 背景分析 1](#_Toc22947)

[1.2 功能分析 1](#_Toc8063)

[2 设计 2](#_Toc5877)

[2.1 数据结构设计 2](#_Toc20771)

[2.2 类结构设计 2](#_Toc3549)

[2.3 成员与操作设计 2](#_Toc28432)

[2.4 系统设计 2](#_Toc14783)

[3 实现 3](#_Toc12679)

[3.1中缀表达式转换为后缀表达式的实现 3](#_Toc6816)

[3.1.1 中缀表达式转换为后缀表达式流程图 3](#_Toc9781)

[3.1.2 中缀表达式转换为后缀表达式核心代码 3](#_Toc17992)

[3.2 取左右操作数的实现 5](#_Toc5960)

[3.2.1 取左右操作数流程图 5](#_Toc25487)

[3.2.2 取左右操作数核心代码 6](#_Toc2065)

[3.3 运算功能的实现 7](#_Toc26868)

[3.3.1 运算功能流程图 7](#_Toc23490)

[3.3.2 运算功能核心代码 7](#_Toc229)

[3.4 总体系统的实现 9](#_Toc7944)

[3.4.1 总体系统流程图 9](#_Toc18794)

[3.4.3 总体系统截屏示例 10](#_Toc29588)

[4 测试 11](#_Toc4628)

[4.1 功能测试 11](#_Toc21712)

[4.1.1 运算符测试 11](#_Toc4691)

[4.1.2 括号测试 11](#_Toc26451)

[4.1.3 单目运算符测试 11](#_Toc9132)

[4.2 出错测试 12](#_Toc20683)

[4.2.1 缺操作数错误 12](#_Toc9822)

[4.2.2 除数为0错误 12](#_Toc17535)

# 1 分析

## 1.1 背景分析

中缀表达式，是一个通用的算术或逻辑公式表示方法， 操作符是以中缀形式处于[操作数](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E6%95%B0/7658270" \t "https://baike.baidu.com/item/%E4%B8%AD%E7%BC%80%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)的中间，中缀表达式是人们常用的算术表示方法。与[前缀表达式](https://baike.baidu.com/item/%E5%89%8D%E7%BC%80%E8%A1%A8%E8%BE%BE%E5%BC%8F" \t "https://baike.baidu.com/item/%E4%B8%AD%E7%BC%80%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)或[后缀表达式](https://baike.baidu.com/item/%E5%90%8E%E7%BC%80%E8%A1%A8%E8%BE%BE%E5%BC%8F" \t "https://baike.baidu.com/item/%E4%B8%AD%E7%BC%80%E8%A1%A8%E8%BE%BE%E5%BC%8F/_blank)相比，中缀表达式不容易被计算机解析，但因为它符合人们的普遍用法，开发求解中缀算数表达式的计算机程序具有十分重要的意义。

## 1.2 功能分析

作为一个简易的中缀算数表达式计算器，首先应该有的功能就是输入要求解的表达式。其次，计算器还应支持包括加减、乘除、取余、乘方、括号和等号等操作符的识别运算，其中优先级是等于<括号<加减<乘除取余<乘方；应该能处理单目运算符：+或-。

并且，程序对所有输入的表达式应该能够作简单的判断，如表达式有错能给出适当的提示。最后，计算器还应该确保可以正确输出计算结果。

综上所述，一个中缀算术表达式至少应该具有输入、判断、处理、输出的功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该计算器要根据操作码优先级以“后进先出”的方式处理字符串，因而选用栈的的数据结构对表达式进行处理。

## 2.2 类结构设计

栈类（Stack）应含有成员：栈顶指针（t），储存空间（elements），应含有基本操作：判空（empty）、判满（full）、返回大小（size）、入栈（push）、出栈（pop）、取栈顶元素（top）、等。

## 2.3 成员与操作设计

**队列类（Stack）**

**私有成员：**

int t;//栈顶指针

T\* elements;//储存空间

**公有操作：**

bool Stack::empty() ;//判断是否为空

bool Stack::full() ;//判断是否已满

int Stack::size() ;//返回栈当前大小

void Stack::push(T element) ;//入栈

void Stack::pop();//出栈

T Stack::top() ;//取栈顶元素

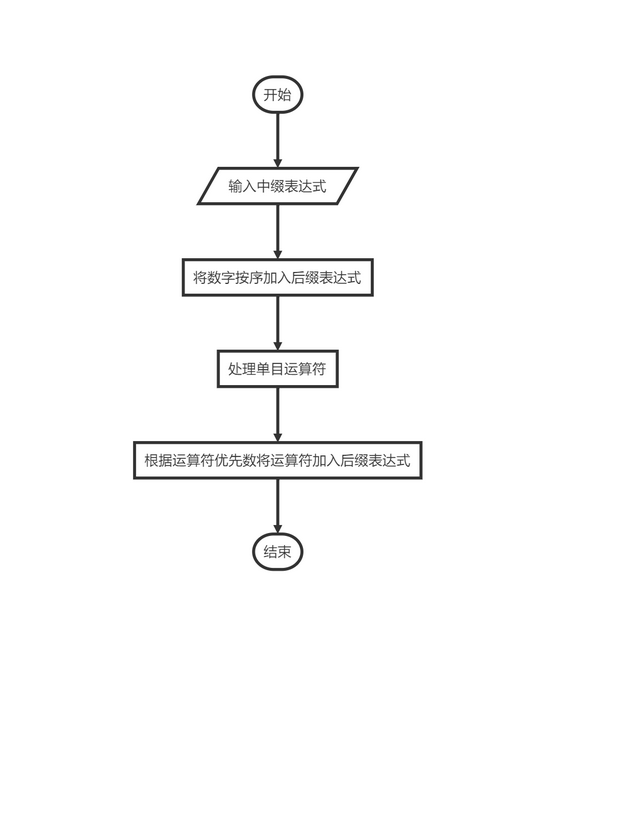
## 2.4 系统设计

计算器首先提示用户输入中缀算术表达式字符串（ch），利用预设优先数以及栈操作将中缀表达式转换为后缀表达式字符串（ans），然后利用操作数栈（operand）取出的值根据运算符进行运算操作，最后输出运算结果。

# 3 实现

## 3.1中缀表达式转换为后缀表达式的实现

### 3.1.1 中缀表达式转换为后缀表达式流程图



### 3.1.2 中缀表达式转换为后缀表达式核心代码

//栈内优先数

int isp(char op) {

switch (op)

{

case '+':return 3;break;

case '-':return 3;break;

case '\*':return 5;break;

case '/':return 5;break;

case '%':return 5;break;

case '^':return 7;break;

case '(':return 1;break;

case ')':return 8;break;

case '=':return 0;break;

}

}

//栈外优先数

int icp(char op) {

switch (op)

{

case '+':return 2;break;

case '-':return 2;break;

case '\*':return 4;break;

case '/':return 4;break;

case '%':return 4;break;

case '^':return 6;break;

case '(':return 8;break;

case ')':return 1;break;

case '=':return 0;break;

}

}

//将中缀表达式转为后缀表达式

string postfix() {

Stack<char> s;

string ans;

string temp,ch;

char ch1,op;

s.push('=');//在栈底放'='

cin >> temp;

temp= '!' + temp;

ch = '!'+temp;

//留两位作为单目运算符缓冲区

for (int i = 2;s.empty() == false;) {

if (isdigit(ch[i])) {

ans += ch[i];

i++;

}

else if ((ch[i] == '-' || ch[i] == '+' )&& isdigit(ch[i - 1]) == false&&ch[i-1]!=')') {//处理单目运算符

string temp2(ch,i+2);//拷贝正/负数后面的所有式子

ch[i - 2] = '(';

ch[i - 1] = '0';

ch[i + 2] = ')';

ch.resize(i + 3);//将单目运算符改为“(0+\_?)"的形式

ch = ch + temp2;

i = i - 2;

}

else {

ch1 = s.top();

if (isp(ch1) < icp(ch[i]))//若后进入的运算符优先级大，则将其入栈

{

s.push(ch[i]);

i++;

}

else if (isp(ch1) > icp(ch[i]))//若后进小，则将栈顶运算符输出

{

ans += s.top();

s.pop();

}

else//若相等，则退栈不输出

{

op = s.top();

s.pop();

if (op == '(')//若为'('，则读下一字符

i++;

}

}

}

ans += '=';

while (!s.empty())

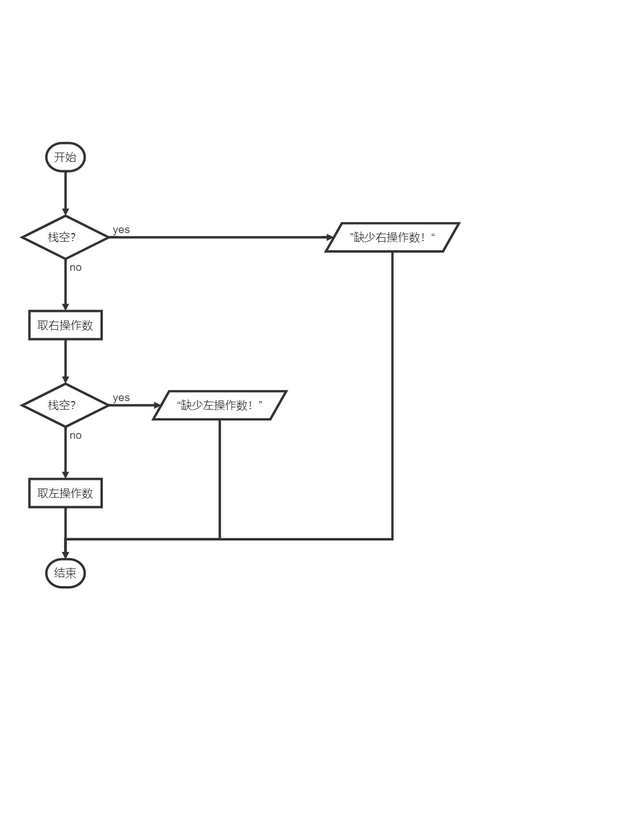
s.pop();

return ans;

}

## 3.2 取左右操作数的实现

### 3.2.1 取左右操作数流程图



### 3.2.2 取左右操作数核心代码

bool get(Stack<double>&num,double& left, double& right) {//从操作数栈中取出两个操作数

if (num.empty() == true) {

cerr << "缺少右操作数！" << endl;

return false;

}

right = num.top();

num.pop();

if (num.empty() == true) {

cerr << "缺少左操作数！" << endl;

return false;

}

left = num.top();

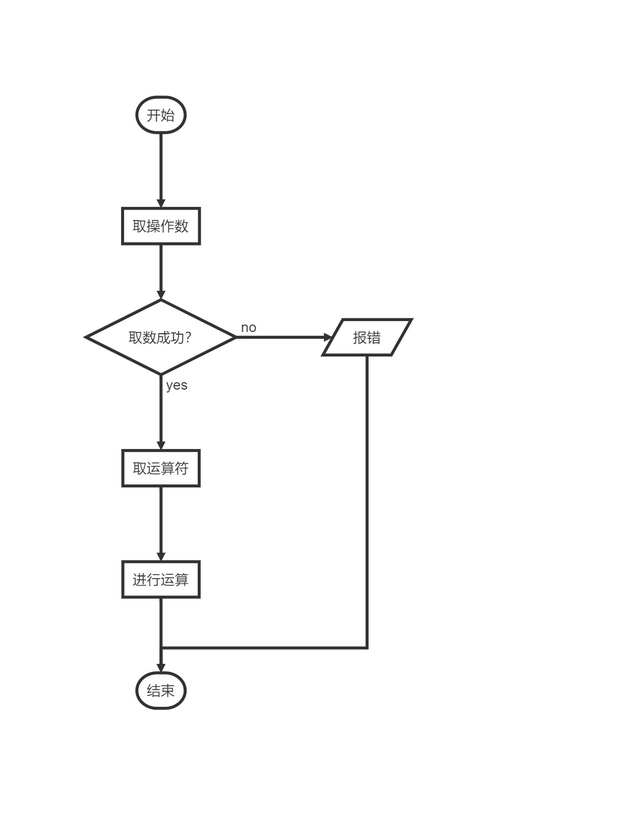
num.pop();

return true;

}

## 3.3 运算功能的实现

### 3.3.1 运算功能流程图



### 3.3.2 运算功能核心代码

void Doperator(Stack<double>&num,char op)//运算操作

{

double left, right, value;

bool wrong;

wrong = get(num,left, right);

if (wrong == true)

switch (op)

{

case '+':value = left + right;num.push(value);break;

case '-':value = left - right;num.push(value);break;

case '\*':value = left \* right;num.push(value);break;

case '/':

if (right == 0.0) {

cerr << "除数为0！" << endl;

while (!num.empty())

num.pop();

exit(1);

}

else

{

value = left / right;num.push(value);

}

;break;

case '%':value = (int)left % (int)right;num.push(value);break;

case '^':value = pow(left, right);num.push(value);break;

}

else

{

while (!num.empty())

num.pop();

exit(1);

}

}

## 3.4 总体系统的实现

### 3.4.1 总体系统流程图总体

3.4.2 总体系统核心代码

int main() {

cout << "欢迎使用运算器！\n请输入您要计算的表达式（以=结尾）：" << endl;

string s;

Stack<double>operand;

s=postfix();

for (int i = 0; s[i] != '='; i++)

if (isdigit(s[i]))

operand.push((double)s[i] - (double)'0');

else

Doperator(operand, s[i]);

cout << "运算结果为：" << operand.top();

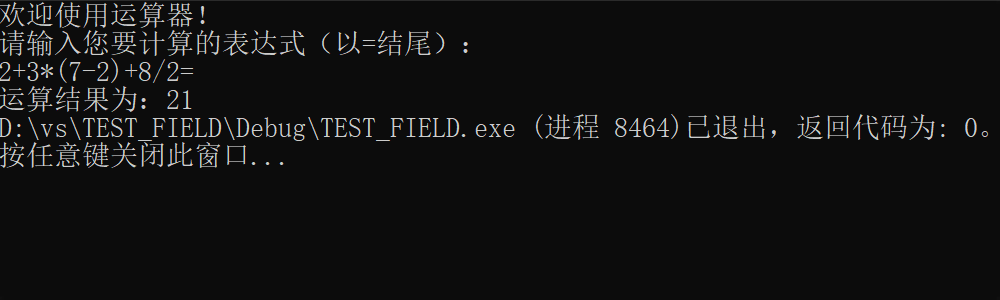
while (!operand.empty())

operand.pop();

return 0;

}

### 3.4.3 总体系统截屏示例



# 4 测试

## 4.1 功能测试

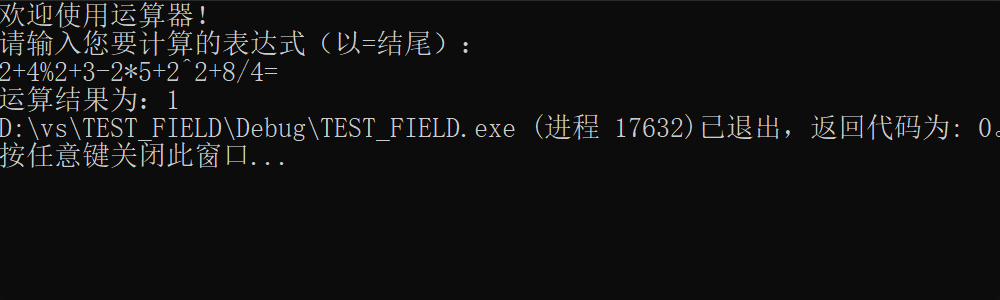
**见各功能截屏示例**

### 4.1.1 运算符测试

**测试用例：**2+4%2+3-2\*5+2^2+8/4=

**预期结果：**1

**实验结果：**

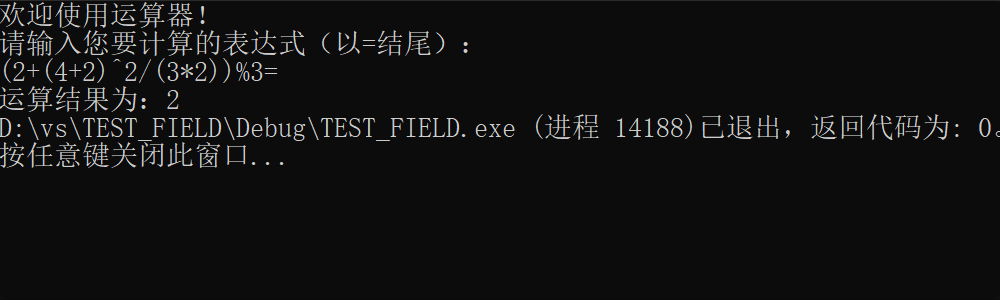


### 4.1.2 括号测试

**测试用例：(**2+(4+2)^2/(3\*2))%3=

**预期结果：**2

**实验结果：**

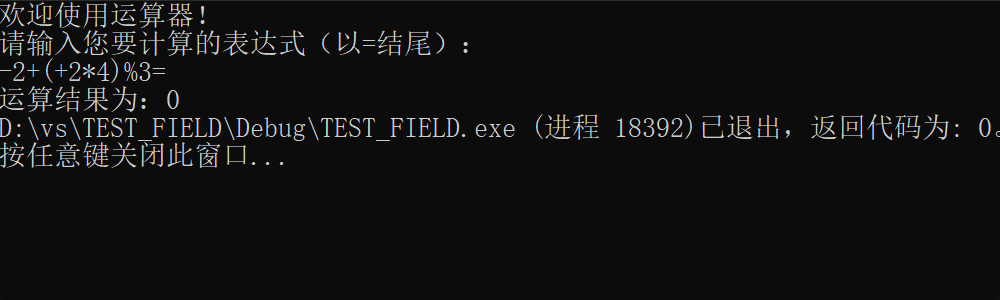


### 4.1.3 单目运算符测试

**测试用例：**-2+(+2\*4)%3=

**预期结果：**0

**实验结果：**



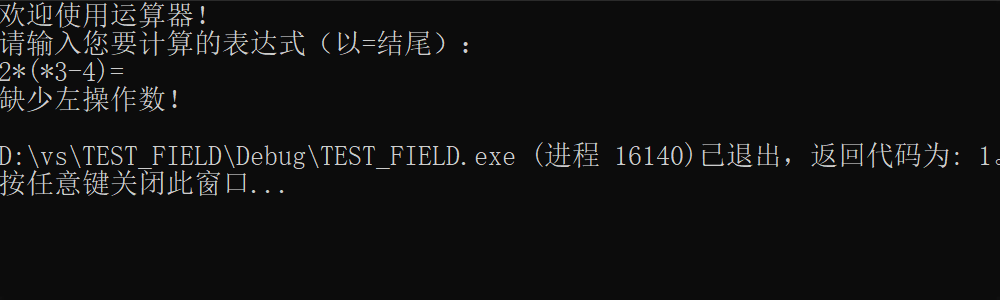
## 4.2 出错测试

### 4.2.1 缺操作数错误

**测试用例：**2\*(\*2-4)=

**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：**

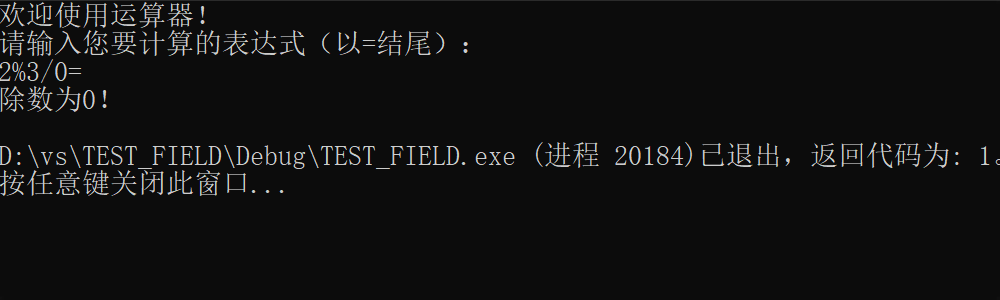
****

### 4.2.2 除数为0错误

**测试用例：**2%3/0=

**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：**

****