项目说明文档

数据结构课程设计

——算数表达式求解

作 者 姓 名： 伊啸

学 号： 1951220

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](#_Toc60004412)

[1.1 功能分析 1](#_Toc60004413)

[2 设计 1](#_Toc60004414)

[2.1 数据结构设计 1](#_Toc60004415)

[2.2 类结构设计 1](#_Toc60004416)

[2.3 成员与操作设计 1](#_Toc60004417)

[2.4 系统设计 3](#_Toc60004418)

[3 实现 4](#_Toc60004419)

[3.1 判断中缀表达式正确与否功能的实现 4](#_Toc60004420)

[3.1.1 判断中缀表达式正确与否功能流程图 4](#_Toc60004421)

[3.1.2 判断中缀表达式正确与否功能核心代码 6](#_Toc60004422)

[3.1.3 判断表达式正确与否功能截屏示例 7](#_Toc60004423)

[3.2 单目运算符功能的实现 7](#_Toc60004424)

[3.2.1 单目运算符功能流程图 7](#_Toc60004425)

[3.2.2 单目运算符功能核心代码 9](#_Toc60004426)

[3.2.3 单目运算符功能截屏示例 9](#_Toc60004427)

[3.3 转化成后缀表达式功能的实现 10](#_Toc60004428)

[3.3.1 转化成后缀表达式功能流程图 10](#_Toc60004429)

[3.3.2 转化成后缀表达式功能核心代码 10](#_Toc60004430)

[3.4 总体系统的实现 12](#_Toc60004431)

[3.4.1 总体系统流程图 12](#_Toc60004432)

[3.4.2 总体系统核心代码 14](#_Toc60004433)

[3.4.3 总体系统截屏示例 14](#_Toc60004435)

[4 测试 14](#_Toc60004436)

[4.1 功能测试 14](#_Toc60004437)

[4.2 出错测试 16](#_Toc60004441)

[4.2.1 判断中缀表达式正确与否功能测试 16](#_Toc60004442)

[4.2.2 除号或求余号后有0 16](#_Toc60004451)

[4.2.3 输入不合法的字符 17](#_Toc60004452)

# 1 分析

## 1.1 功能分析

作为一个最简易的计算器，能够判断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。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

对操作符的优先级进行操作，容易想到通过栈实现，我写的栈是基于数组实现的。通过辅助队列来存储中缀表达式转化成后缀表达式的结果，便于处理一元运算符和出错测试等等。此外，该项目也设计了一个Calculator类，对表达式进行操作，如取操作数，做各种运算等等。

## 2.2 类结构设计

数组栈：栈顶指针、栈的大小、存储栈的数组为私有成员，对栈进行的操作为公有函数。

数组队列：头、尾指针、队列大小和存储队列的数组为私有成员，对队列进行的操作为公有函数。

Calculator类：存储表达式的栈和取操作数的函数为私有成员，其他操作如做运算等为公有函数。

## 2.3 成员与操作设计

**栈（class stack）**

**由于用到了不止一次栈，所以设计了栈的模板。**

**私有成员：**

T\* elements;//栈数组

int top;//指向栈顶的指针

int maxsize;//栈的最大容量

**公有操作：**

stack(int sz=50);//构造函数

~stack() { delete[]elements; }//析构函数

bool Push(const T& x);//将元素x入栈

bool Pop(T& x);// 出栈，并且弹出的元素由x返回

bool getTop(T& x);// 用x返回栈顶元素

bool IsEmpty()const { return top == -1 ? true : false; }

//判断栈是否为空

bool IsFull()const { return top == maxsize-1 ? true : false; }

//确定堆栈是否已满

int getSize()const { return top + 1; }//返回栈的大小

void MakeEmpty() { top = -1; }//将栈清空

**队列（class queue）**

**私有成员：**

int rear, front;//队列头指针和尾指针

char\* element;//队列数组

int maxsize;// 队列可容纳的最大元素数

**公有操作：**

queue(int sz = 50);//构造函数

~queue() {};//析构函数

bool Enqueue(const char& x);

//如果队列未满，请入队x并返回true，否则返回false

bool Dequeue(char& x);

//如果队列不为空，则使head元素出队并返回true，否则返回false

bool getFront(char& x);// 通过x返回队列的head元素

void makeEmpty() { front = rear = 0; }//使队列为空

bool IsEmpty()const { return (rear == front) ? true : false; }

//确定队列是否为空

bool IsFull()const { return rear == maxsize ? true : false; }

//确定队列是否已满

int getSize()const { return rear - front; }//返回堆栈的大小

bool postfix();//将中缀表达式转换为后缀表达式

bool judge\_infix(char ch[]);//对中缀表达式进行判断

**计算器类（class Calculator）**

**私有成员：**

stack<int> st;// 栈对象定义

bool get2Operands(int& left, int& right);

//弹出两个操作数，成功则返回true，否则返回false

**公有操作：**

Calculator(int sz) :st(sz) {}//构造函数

~Calculator(){}//析构函数

void DoOperator(char op);//对两个操作数计算

void AddOperand(int value);//将元素value入栈

void Clear(); //置空

void show(); //输出结果

## 2.4 系统设计

由用户输入中缀表达式，系统先对表达式进行判断，如果表达式不合理，则输出相应错误提示，直到输入正确的表达式，然后进行运算，运算后输出结果再提示用户是否继续进行表达式计算。

# 3 实现

## 3.1 判断中缀表达式正确与否功能的实现

2

2

2

2

2

2

### 3.1.1 判断中缀表达式正确与否功能流程图



### 3.1.2 判断中缀表达式正确与否功能核心代码

int temp = 0; //用于判断括号是否匹配的标志

if (ch[0] == '\*' || ch[0] == '/' || ch[0] == '%' || ch[0] == '^')

//确定第一个字符是否正确

return false;

for (int i = 0; i < strlen(ch); i++) {

if (ch[i] == '(') { //确定左括号之前和之后的字符是否正确

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

return false;

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

return false;

temp++;

}

else if (ch[i] == ')') { //确定右括号前后的字符是否正确

if (i == 0)

return false;

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

return false;

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

return false;

temp--;

}

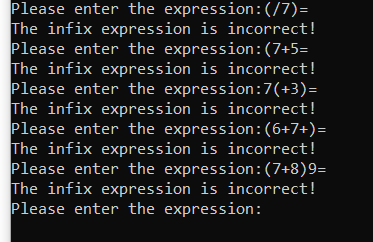
}

if (temp == 0) //括号数量匹配

return true;

return false;

### 3.1.3 判断表达式正确与否功能截屏示例



## 3.2 单目运算符功能的实现

### 3.2.1 单目运算符功能流程图

输入的中缀表达式先用队列存储，有单目运算符时在其前面加上‘0’，使之成为双目运算符。



### 3.2.2 单目运算符功能核心代码

queue unary\_operator(queue& Queue) {

char elem, next\_elem;

queue unary;

Queue.getFront(elem);

if (elem == '+' || elem == '-') { //如果第一个字符是“ +”或“-”

Queue.Dequeue(elem);

Queue.getFront(next\_elem);

if (next\_elem >= '0' && next\_elem <= '9') {

unary.Enqueue('0');

//将字符“ 0”添加到队列中，以便一元运算符成为二元运算符 unary.Enqueue(elem);

}

}

while (!Queue.IsEmpty()) {

Queue.Dequeue(elem);

if (Queue.getFront(next\_elem)) {

if ((next\_elem == '+' || next\_elem == '-') && (elem == '(')) {

//一元运算符后加括号

unary.Enqueue('0');

}

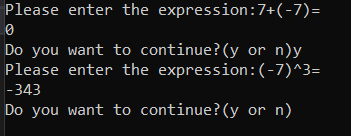
}

unary.Enqueue(elem);

}

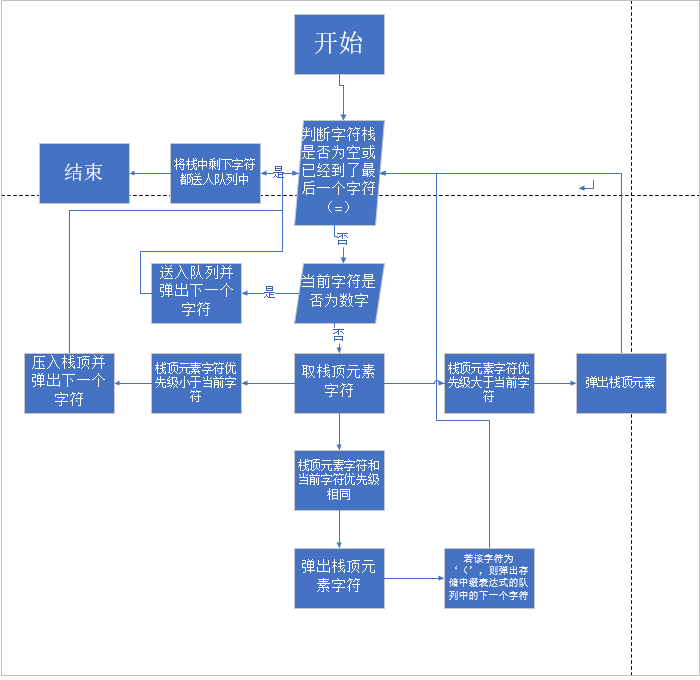
return unary;

### }3.2.3 单目运算符功能截屏示例



## 3.3 转化成后缀表达式功能的实现

### 3.3.1 转化成后缀表达式功能流程图



### 3.3.2 转化成后缀表达式功能核心代码

while (st\_ch.IsEmpty() == false && ch != '=') {

if (isdigit(ch)) {

if (Enqueue(ch))

infix.Dequeue(ch);

}

else {

if (st\_ch.getTop(ch1)) {

if (isp(ch1) < icp(ch)) { //新输入运算符具有较高的优先级 st\_ch.Push(ch);

infix.Dequeue(ch);

}

else if (isp(ch1) > icp(ch)) { //新输入运算符的优先级较低 st\_ch.Pop(op);

Enqueue(op);

}

else {

st\_ch.Pop(op);

if (op == '(')

//新输入运算符的优先级与堆栈顶部的运算符的优先级相同 infix.Dequeue(ch);

}

}

}

}

while (st\_ch.IsEmpty() == false) {

st\_ch.Pop(ch1);

Enqueue(ch1);

}

## 3.4 总体系统的实现

### 3.4.1 总体系统流程图



### 3.4.2 总体系统核心代码

while (queue\_postfix.IsEmpty() == false) {

queue\_postfix.Dequeue(element);

if (isdigit(element))

Calc.AddOperand(element - 48);

//将char类型数据转换为int类型

else {

if (element != '=')

Calc.DoOperator(element); //对操作符进行操作

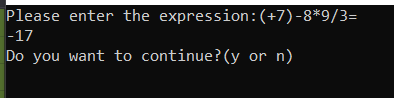
else

break;

}

### }

### 3.4.3 总体系统截屏示例



# 4 测试

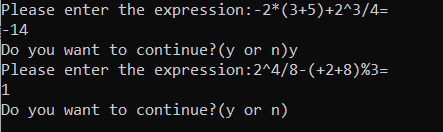
## 4.1 功能测试

### 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 出错测试

### 4.2.1 判断中缀表达式正确与否功能测试

### 测试用例：（/7）=

### （7+5=

### 7（+3）=

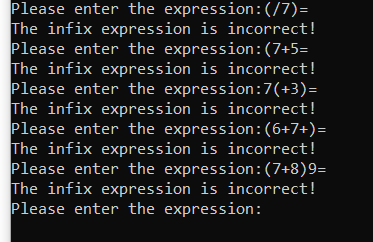
### （6+7+）=

### （7+8）9=

### 预期结果：

### 错误提示，程序不崩溃

### 实验结果

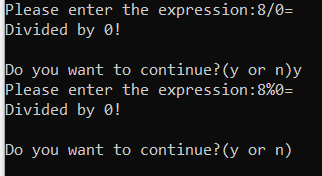


### 4.2.2 除号或求余号后有0

**测试用例：**8/0= 8%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：**



### 4.2.3 输入不合法的字符

**测试用例：**abc, $@符号等

**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：**

