

项目说明文档

数据结构课程设计

——算术表达式求解

作者姓名：_____翟晨昊_____

学 号：_____1952216_____

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.....	- 4 -
1.1	背景分析	- 4 -
1.2	功能分析	- 4 -
2	设计.....	- 5 -
2.1	数据结构设计.....	- 5 -
2.2	类结构设计	- 5 -
2.3	成员与操作设计	- 5 -
2.4	系统设计	- 9 -
3	实现.....	- 10 -
3.1	分析单目运算符功能的实现	- 10 -
3.1.1	分析单目运算符功能流程图.....	- 10 -
3.1.2	分析单目运算符功能核心代码	- 11 -
3.1.3	分析单目运算符功能描述.....	- 12 -
3.2	分析优先级功能的实现	- 13 -
3.2.1	分析优先级功能流程图.....	- 13 -
3.2.2	分析优先级功能核心代码.....	- 14 -
3.2.3	分析优先级功能描述	- 16 -
3.3	表达式计算功能的实现	- 17 -
3.3.1	表达式计算功能流程图.....	- 17 -
3.3.2	表达式计算功能核心代码.....	- 18 -
3.3.3	表达式计算功能描述	- 24 -
3.4	总体功能的实现.....	- 25 -
3.4.1	总体功能流程图	- 25 -
3.4.2	总体功能核心代码	- 26 -
3.4.3	总体功能截屏示例	- 27 -
4	测试.....	- 28 -
4.1	功能测试	- 28 -
4.1.1	加法功能测试.....	- 28 -
4.1.2	减法功能测试.....	- 28 -
4.1.3	乘法功能测试.....	- 28 -
4.1.4	除法功能测试.....	- 29 -
4.1.5	取余功能测试.....	- 29 -
4.1.6	乘方功能测试.....	- 29 -
4.1.7	带括号功能测试	- 29 -

4.1.8 一目运算符测试	30 -
4.1.9 多一目运算符测试	30 -
4.1.10 综合测试.....	30 -
4.1.11 运算数带小数的综合测试.....	30 -
4.2 边界测试	31 -
4.2.1 表达式中无运算符	31 -
4.2.2 取余运算时运算数是小数.....	31 -
4.3 出错测试	32 -
4.3.1 表达式最后没有等号	32 -
4.3.2 表达式左右括号不匹配.....	32 -
4.3.3 除法运算时除数为零	32 -
4.3.4 取余运算时除数为零	33 -
4.3.5 表达式中出现未定义符号.....	33 -
4.3.6 操作数和操作符不匹配.....	33 -
4.3.7 小数点位置不正确或多余.....	34 -
4.3.8 表达式为空.....	34 -
4.3.9 表达式中只有括号	35 -
4.3.10 表达式最后有多个等号.....	35 -
4.3.11 表达式中含有空括号	35 -
4.3.12 小数部分存在单目运算符.....	36 -

1 分析

1.1 背景分析

在日常生活中，我们经常会遇到需要计算算术表达式的情况，如果是遇到包括数字很小、长度较短、计算不复杂的算术表达式还好，但如果遇到的是比较复杂的表达式，那么我们必须借助纸张来进行计算，并且还很有可能算错，这给日常生活带来了不便。因此，编写一个可以用来计算算术表达式的程序十分重要。让计算机来计算，不仅正确率大大提高，并且也会省去很多时间。

1.2 功能分析

作为一个算数表达式求解程序，要将输入中缀表达式转化为后缀表达式。我们日常遇到与使用的表达式都为中缀表达式，即二元运算符处于两个运算数中间。这种表达式人读起来非常容易理解，但是用计算机计算中缀表达式的时候会出现符号优先级的問題，还会出现括号带来的问题，因此我们要将输入的中缀表达式转化为计算机容易理解的后缀表达式，再进行计算。例如， $8+4-6*2$ 用后缀表达式表示即为 $8\ 4+6\ 2\ *-$ ； $2*(3+5)+7/1-4$ 用后缀表达式表示即为 $2\ 3\ 5+*7\ 1/+4-$ 。

所以，程序首先判断输入表达式的合法性，随后将中缀表达式转为后缀表达式，再通过计算得到结果后输出，等待用户决定是否再进行一次新的计算操作或是退出。

在本程序中，支持加减乘除，取余，乘方以及一元运算符 $+$ ， $-$ 运算。还支持带有括号的表达式，以及运算数或运算结果中含有小数的表达式。表达式须以“=”结尾。

2 设计

2.1 数据结构设计

如上功能分析所述，该考试报名系统要求大量增加、删除、修改操作，使用数组，链表等数据结构都可以完成这些操作。但数组在进行这些操作的时候可能会移动整个数组，太过浪费时间，相比较之下，链表在进行增加，删除等操作时非常简便，因此最后选择使用双向链表来维护信息，同时在链表前附加了一个头结点，这样使得增加和删除头结点时与处理其它结点方法相同，简化了代码。

2.2 类结构设计

首先，为了能够将中缀表达式转化为后缀表达式，我们必须要有个栈类（Stack 类），本程序中是用 Vector 实现的栈类。其次，由于表达式是用字符串输入的，我们必须保存每一个字符是否为数字，如果为运算符，还要保存它的优先级，因此，设计了一个字符数据类（CharData 类）来保存每一个字符的信息，CharData 类中的成员全部为公有类型，方便读取。最后，还设计了一个运算系统类（CalculateSystem 类），将表达式字符串的解析，转化，计算，展示结果等功能整合在一起，更加方便管理。为了使向量类和栈类更具有泛用性，本程序将 Vector 类与 Stack 类都设计为了模板类。

2.3 成员与操作设计

向量类（Vector）：

```
template <typename ElementType> class Vector {
public:
    ~Vector();
    Vector();
    ElementType& operator[](const int x);
    const ElementType& operator[](const int x)const;
    Vector<ElementType>(const Vector<ElementType>& rhs);
    Vector<ElementType>& operator=(const Vector<ElementType>& rhs);
    bool isFull();
    bool isEmpty();
    void pushBack(const ElementType& temp);
    void popBack();
    void clear();
    int getSize();
    void reSize(int newSize);
private:
    void extendSize();
    int size;
    int maxSize;
    ElementType* myVector;
```

```
};
```

私有成员:

```
int size;//Vector 中已经存储的元素数量
int maxSize;//Vector 中已经申请的空间大小, 元素数量如果超过要再次申请
ElementType* myVector;//存储的元素序列的起始地址
```

私有操作:

```
void extendSize();
//在 Vector 申请的空间已经被占满时再次申请空间, 每次扩充至 maxSize*2+1
是因为刚开始的 maxSize 为 0
```

公有操作:

```
Vector();
//构造函数, 初始化指针并将 size 与 maxSize 置为 0
```

```
~Vector();
//析构函数, 调用 clear() 函数来删除元素, 释放内存
```

```
ElementType& operator[](const int x);
//重载[]运算符, 返回 ElementType&类型
```

```
const ElementType& operator[](const int x)const;
//重载[]运算符, 返回 const ElementType&类型
```

```
Vector<ElementType>(const Vector<ElementType>& rhs);
//复制构造函数, 将一个 Vector 复制给另一个 Vector
```

```
Vector<ElementType>& operator=(const Vector<ElementType>& rhs);
//重载=运算符, 可以将一个 Vector 赋给另一个 Vector
```

```
bool isFull();
//判断是否 Vector 中申请的内存已经被占满
```

```
bool isEmpty();
//判断 Vector 是否为空
```

```
void pushBack(const ElementType& temp);
//在 Vector 末尾添加一个元素
```

```
void popBack();
//删除 Vector 最末尾的元素
```

```
void clear();
```

//删除 Vector 中的所有元素并释放内存

int getSize();

//返回 Vector 已经存储的元素数量

void reSize(int newSize);

//重设 Vector 的大小，若比之前小，则会抛弃多余的元素

栈类 (Stack):

template <typename ElementType> class Stack

{

public:

void push(ElementType input);

void pop();

ElementType top();

bool isEmpty();

int getSize();

void clear();

private:

Vector<ElementType> stack;

};

私有成员:

Vector<ElementType> stack; //用 Vector 储存栈中的元素

公有操作:

void push(ElementType input);

//向栈顶增加元素

void pop();

//删除栈顶元素

ElementType top();

//返回栈顶元素

bool isEmpty();

//判断栈中是否为空

int getSize();

//返回栈中元素个数

void clear();

//清空栈中元素

字符数据类 (CharData):

```
class CharData {
public:
    CharData() = default;
    CharData(char inputChar):character(inputChar){}
    int priority = 0;
    char character = 0;
    bool isNum = false;
};
```

公有成员:

```
int priority;//该字符的运算优先级
char character;//该字符
bool isNum;//该字符是否为数字
```

公有操作:

```
CharData() = default;
//默认构造函数
```

```
CharData(char inputChar)
//含参构造函数
```

运算系统类 (CalculateSystem):

```
class CalculateSystem {
public:
    void clear();
    void unaryAnalysis(string& s);
    bool analysis(const string& s);
    bool calculate();
    int calculateNum();
    bool operation();
    double getAnswer();
private:
    Vector<CharData> inputExpression;
    Stack<CharData> operatorStack;
    Stack<double> numStack;
    string eachNum = "";
    int current = 0;
    int left = 0;
};
```


私有成员：

```
Vector<CharData> inputExpression;  
//保存输入后经过分析的字符串表达式  
Stack<CharData> operatorStack;  
//运算符要入的栈  
Stack<double> numStack;  
//数字要入的栈  
string eachNum;//每次计算出的数字  
int current;//当前运算到的位置  
int left;//多余左括号个数
```

公有操作：

```
void clear();  
//清空并初始化运算系统  
  
void unaryAnalysis(string& s);  
//处理单目运算符  
  
bool analysis(const string& s);  
//判断并添加每一个运算符的优先级，同时判断是否有等号，括号是否匹配  
  
bool calculate();  
//计算表达式的结果  
  
int calculateNum();  
//计算字符串中某一个数字的值  
  
bool operation();  
//进行运算符的运算  
  
double getAnswer();  
//返回运算结果
```

2.4 系统设计

程序在使用时，首先输入表达式字符串，然后对字符串进行解析，之后在进行后缀表达式计算，如果表达式合法，计算成功，则会返回并展示运算结果，如果计算失败则会显示错误提示。再计算完一个表达式后，用户可以选择重新计算一个新的表达式，或者退出系统。

3.1.2 分析单目运算符功能核心代码

CalculateSystem 类中:

```
void CalculateSystem::unaryAnalysis(string& s)
{
    for (size_t i = 0; i < s.size(); i++)
    {
        if (i == 0)
        {
            if (s[i] == '+')
            {
                s[i] = 1;
            }
            else if (s[i] == '-')
            {
                s[i] = 2;
            }
        }
        else
        {
            if ((s[i - 1] != ')') && s[i - 1] != '.') && (s[i - 1] < '0'
|| s[i - 1] > '9'))
            {
                if (s[i] == '+')
                {
                    s[i] = 1;
                }
                else if (s[i] == '-')
                {
                    s[i] = 2;
                }
            }
        }
    }
}
```

3.1.3 分析单目运算符功能描述

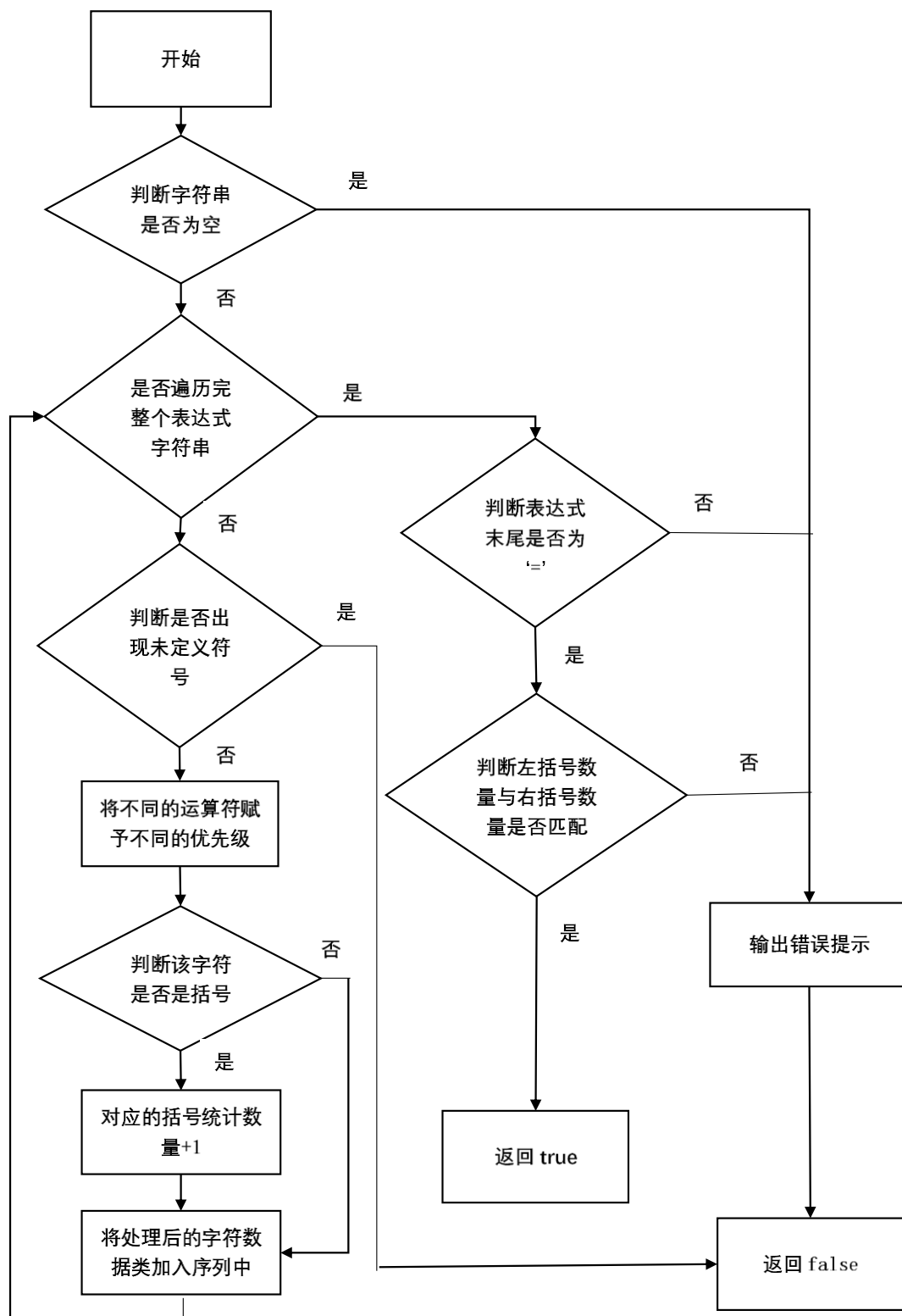
在刚开始，为了将一元运算符的 ‘+’ ‘-’ 与二元运算符的 ‘+’ ‘-’ 区分开，首先要把一元运算符用其他的符号来表示，但是选用其他符号后，如果表达式输入的就是该符号的话，不会报错，反而会将其当作一元运算符，综上思考，最后选用了 ASCLL 码中的 1 和 2 来代替表示。函数中的 if 语句：

```
if ((s[i - 1] != ')') && s[i - 1] != '.') && (s[i - 1] < '0' || s[i - 1] > '9'))
```

是用来判断该符号应该被看作一元运算符还是二元运算符。

3.2 分析优先级功能的实现

3.2.1 分析优先级功能流程图



3.2.2 分析优先级功能核心代码

CalculateSystem 类中:

```
bool CalculateSystem::analysis(const string& s)
{
    if (s.size() == 0)
    {
        cout << "未输入表达式! " << endl;
        return false;
    }
    size_t i = 0;
    for (i = 0; i < s.size() - 1; i++)
    {
        CharData temp(s[i]);
        if (s[i] < '0' || s[i] > '9')
        {
            switch (s[i])
            {
                case '.':
                    temp.priority = 8;
                    break;
                case '1':
                    temp.priority = 7;
                    break;
                case '2':
                    temp.priority = 7;
                    break;
                case '(':
                    temp.priority = 6;
                    left++;
                    break;
                case '^':
                    temp.priority = 5;
                    break;
                case '%':
                    temp.priority = 4;
                    break;
                case '*':
                    temp.priority = 4;
                    break;
                case '/':
                    temp.priority = 4;
                    break;
                case '+':
```

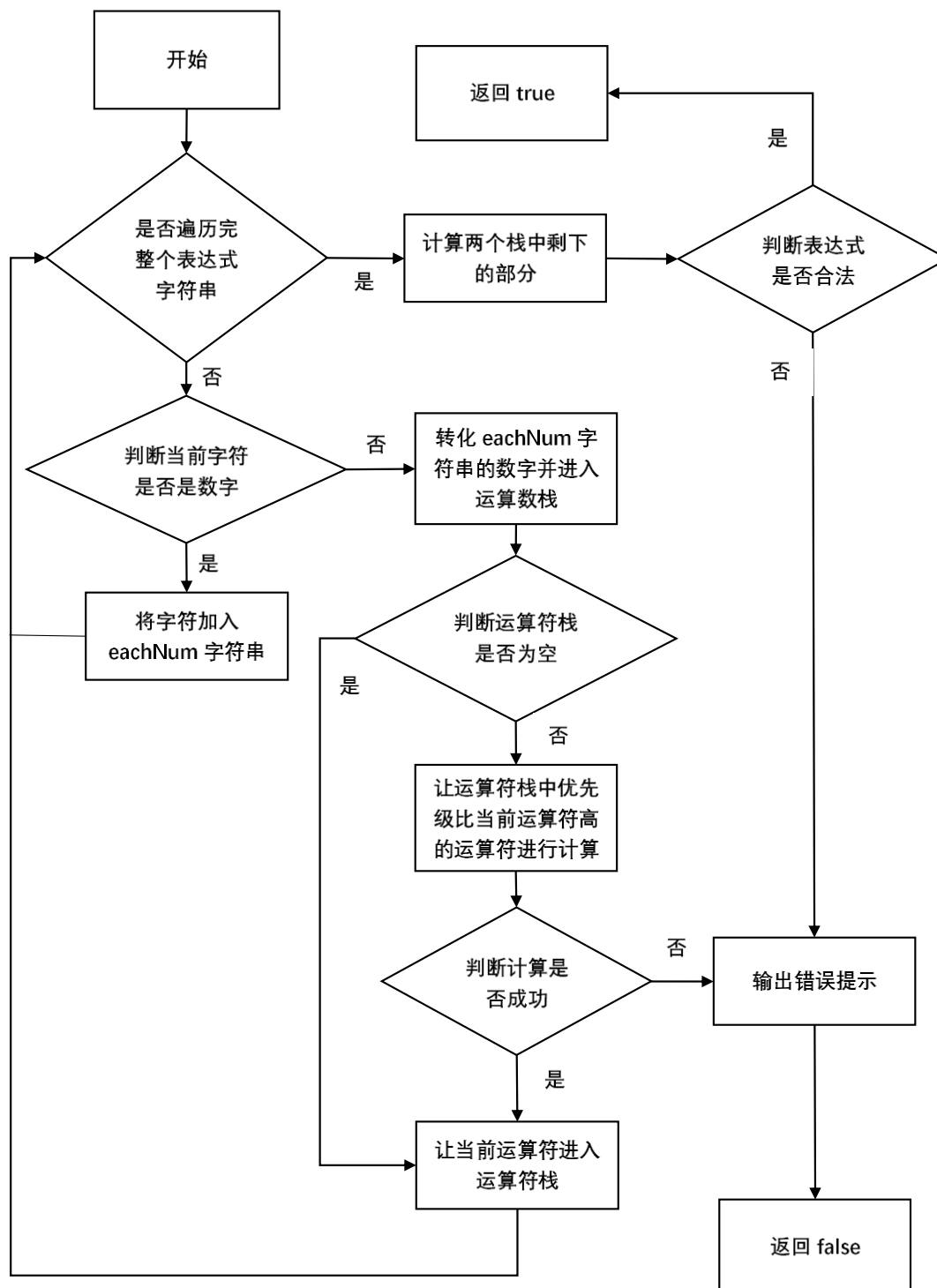
```
        temp.priority = 3;
        break;
    case '-':
        temp.priority = 3;
        break;
    case ')':
        temp.priority = 1;
        if (s[i - 1] == '(')
        {
            cout << "出现空括号!" << endl;
            return false;
        }
        left--;
        break;
    default:
        cout << "出现未定义的符号! " << endl;
        return false;
    }
}
else
{
    temp.isNum = true;
}
if (left < 0)
{
    cout << "左括号和右括号未成功匹配! " << endl;
    return false;
}
inputExpression.pushBack(temp);
}
if (s[i] != '=')
{
    cout << "表达式的结尾不为等号! " << endl;
    return false;
}
if (left != 0)
{
    cout << "左括号和右括号数量未成功匹配! " << endl;
    return false;
}
return true;
}
```

3.2.3 分析优先级功能描述

这是将字符串中的运算符全部赋予优先级，优先级的顺序是小数点>正负号>左括号>乘方>取余>乘除号>加减号>右括号。其中程序将小数点也认为是一种运算符，在之后的计算中会为其安排对应的运算规则；左括号给的初始优先级很高，但在其入栈后会调低其优先级；同时该函数还可以通过判断表达式是否为空，最后是否为等号，左右括号是否匹配及是否存在未定义符号这四个方面来对表达式的合法性有一个初步的判断。

3.3 表达式计算功能的实现

3.3.1 表达式计算功能流程图



3.3.2 表达式计算功能核心代码

CalculateSystem 类中:

```
bool CalculateSystem::calculate()
{
    for (current = 0; current < inputExpression.getSize(); current++)
    {
        if (inputExpression[current].isNum)
        {
            eachNum += inputExpression[current].character;
        }
        else
        {
            if (eachNum != "")
            {
                numStack.push(calculateNum());
            }
            while (!operatorStack.isEmpty())
            {
                if (inputExpression[current].priority <= operatorStack.
top().priority)
                {
                    if (inputExpression[current].priority == 7 && opera
torStack.top().priority == 7)
                    {
                        break;
                    }
                    if (!operation())
                    {
                        cout << "在解析第" << current + 1 << "位字符时出
现问题!" << endl;
                        cout << "输入的表达式不合法或遇到未知错误!"
<< endl;
                        return false;
                    }
                }
            }
            else
            {
                break;
            }
        }
        if (inputExpression[current].character == '(')
        {
            inputExpression[current].priority = 2;
        }
    }
}
```

```
        }
        operatorStack.push(inputExpression[current]);
        eachNum = "";
    }
}
if (eachNum != "")
{
    numStack.push(calculateNum());
}
while (!operatorStack.isEmpty() && !numStack.isEmpty())
{
    if (!operation())
    {
        cout << "输入的表达式不合法或遇到未知错误!" << endl;
        return false;
    }
}
if (operatorStack.getSize() != 0 || numStack.getSize() != 1)
{
    cout << "输入的表达式不合法!" << endl;
    return false;
}
return true;
}

int CalculateSystem::calculateNum()
{
    int multiple = 1;
    int sum = 0;
    for (size_t i = eachNum.size(); i > 0; i--)
    {
        sum += ((eachNum[i - 1] - '0') * multiple);
        multiple *= 10;
    }
    return sum;
}

bool CalculateSystem::operation()
{
    if (numStack.isEmpty())
    {
        return false;
    }
    double temp1 = numStack.top();
```

```
double temp2 = 0.0;
switch (operatorStack.top().character)
{
case 1:
    if (numStack.getSize() >= 1)
    {
        temp1 = temp1;
    }
    else
    {
        return false;
    }
    break;
case 2:
    if (numStack.getSize() >= 1)
    {
        temp1 = -temp1;
        numStack.pop();
        numStack.push(temp1);
    }
    else
    {
        return false;
    }
    break;
case '.':
    if (numStack.getSize() >= 2)
    {
        if (current != inputExpression.getSize())
        {
            if (inputExpression[current].character == '.')
            {
                return false;
            }
        }
        if (eachNum.size() == 0)
        {
            return false;
        }
        for (size_t i = 0; i < eachNum.size(); i++)
        {
            temp1 = temp1 / 10;
        }
        numStack.pop();
    }
}
```

```
        temp2 = numStack.top();
        temp2 += temp1;
        numStack.pop();
        numStack.push(temp2);
    }
    else
    {
        return false;
    }
    break;
case '^': //x 不能为负数且 y 为小数, 或者 x 为 0 且 y 小于等于 0
    if (numStack.getSize() >= 2)
    {
        numStack.pop();
        temp2 = numStack.top();
        if ((temp2 < 0) && (static_cast<int>(temp1) != temp1))
        {
            return false;
        }
        else if ((temp2 == 0) && (temp1 <= 0))
        {
            return false;
        }
        temp2 = pow(temp2, temp1);
        numStack.pop();
        numStack.push(temp2);
    }
    else
    {
        return false;
    }
    break;
case '%':
    if (abs(temp1)<ERROR)
    {
        cout << "除数不能是 0!" << endl;
        return false;
    }
    else if (numStack.getSize() >= 2)
    {
        numStack.pop();
        temp2 = numStack.top();
        temp2 = static_cast<int>(temp2) % static_cast<int>(temp1);
        numStack.pop();
```

```
        numStack.push(temp2);
    }
    else
    {
        return false;
    }
    break;
case '*':
    if (numStack.getSize() >= 2)
    {
        numStack.pop();
        temp2 = numStack.top();
        temp2 = temp2 * temp1;
        numStack.pop();
        numStack.push(temp2);
    }
    else
    {
        return false;
    }
    break;
case '/':
    if (abs(temp1)<ERROR)
    {
        cout << "除数不能是 0!" << endl;
        return false;
    }
    else if (numStack.getSize() >= 2)
    {
        numStack.pop();
        temp2 = numStack.top();
        temp2 = temp2 / temp1;
        numStack.pop();
        numStack.push(temp2);
    }
    else
    {
        return false;
    }
    break;
case '+':
    if (numStack.getSize() >= 2)
    {
        numStack.pop();
```

```
        temp2 = numStack.top();
        temp2 = temp2 + temp1;
        numStack.pop();
        numStack.push(temp2);
    }
    else
    {
        return false;
    }
    break;
case '-':
    if (numStack.getSize() >= 2)
    {
        numStack.pop();
        temp2 = numStack.top();
        temp2 = temp2 - temp1;
        numStack.pop();
        numStack.push(temp2);
    }
    else
    {
        return false;
    }
    break;
case '(':
    inputExpression[current].priority = 10;
    break;
case ')':
    break;
}
operatorStack.pop();
return true;
}
```

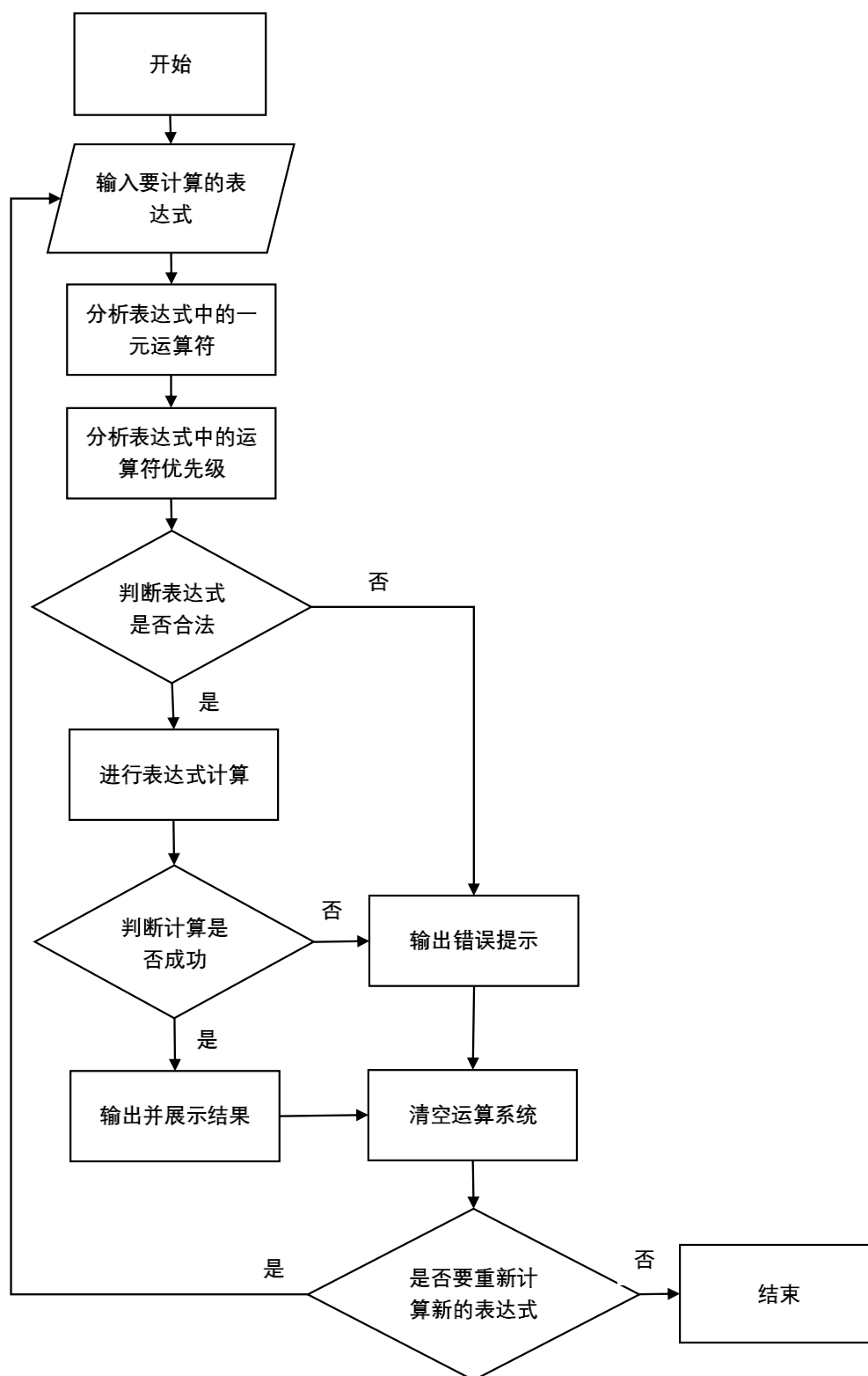
3.3.3 表达式计算功能描述

对于已经处理好的字符数据序列进行遍历，首先要判断每一个字符是不是数字，如果是的话就把它添加进 `eachNum` 字符串中，直到遇到第一个不是数字的字符 `op`。随后将 `eachNum` 中的这个数转化为可以进行计算的 `int` 类型，进入运算数栈（注意转化后先不要清空 `eachNum`）。之后，再把运算符栈中优先级比 `op` 高的运算符拿出来进行运算后将 `op` 入栈。如果 `op` 是 ‘(’，需要在入栈后将 ‘(’ 的优先级调低至只比 ‘)’ 高的水平。最后把 `eachNum` 清 0。最后再把 `eachNum` 清 0 是因为如果运算的时候遇到小数点，需要知道小数部分的数字位数。

在表达式计算结束后进行检查，如果运算符栈中大小不为 0 或数字栈中大小不为 1，就说明输入的表达式不合法。在运算过程中也会进行检查，一旦出现例如除数为 0，或者运算数栈中没有两个数支持二元运算等错误，也说明输入的表达式不合法。

3.4 总体功能的实现

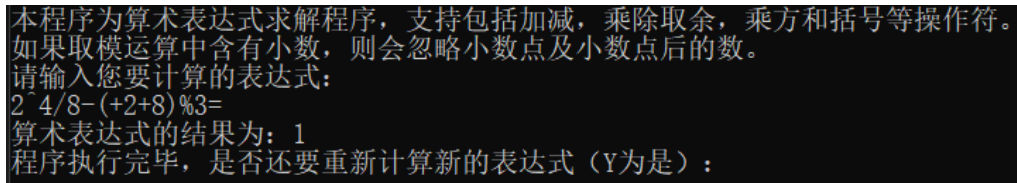
3.4.1 总体功能流程图



3.4.2 总体功能核心代码

```
int main()
{
    CalculateSystem arithmetic;
    string expression = "";
    string judge = "Y";
    cout << "本程序为算术表达式求解程序，支持包括加减，乘除取余，乘方和括号等  
操作符。" << endl;
    cout << "如果取模运算中含有小数，则会忽略小数点及小数点后的数。  
" << endl;
    while (judge == "Y")
    {
        cout<<"请输入您要计算的表达式： " << endl;
        cin >> expression;
        arithmetic.unaryAnalysis(expression);
        if (arithmetic.analysis(expression))
        {
            if (arithmetic.calculate())
            {
                cout << "算术表达式的结果为： ";
                cout << arithmetic.getAnswer() << endl;
            }
        }
        cout << "程序执行完毕，是否还要重新计算新的表达式（Y 为是）： ";
        cin >> judge;
        arithmetic.clear();
    }
    cout << "程序即将退出，欢迎下次使用！ " << endl;
    return 0;
}
```

3.4.3 总体功能截屏示例



本程序为算术表达式求解程序，支持包括加减，乘除取余，乘方和括号等操作符。
如果取模运算中含有小数，则会忽略小数点及小数点后的数。
请输入您要计算的表达式：
2^4/8-(+2+8)%3=
算术表达式的结果为：1
程序执行完毕，是否还要重新计算新的表达式（Y为是）：

4 测试

4.1 功能测试

4.1.1 加法功能测试

测试用例: $13546+78345=$

预期结果: 91891

实验结果:

```
请输入您要计算的表达式:  
13546+78345=  
算术表达式的结果为: 91891
```

4.1.2 减法功能测试

测试用例: $45878-57646=$

预期结果: -11768

实验结果:

```
请输入您要计算的表达式:  
45878-57646=  
算术表达式的结果为: -11768
```

4.1.3 乘法功能测试

测试用例: $999*666=$

预期结果: 665334

实验结果:

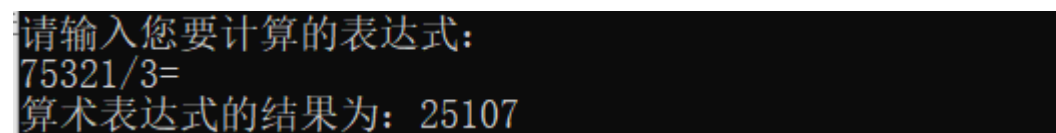
```
请输入您要计算的表达式:  
999*666=  
算术表达式的结果为: 665334
```

4.1.4 除法功能测试

测试用例: $75321/3=$

预期结果: 25107

实验结果:



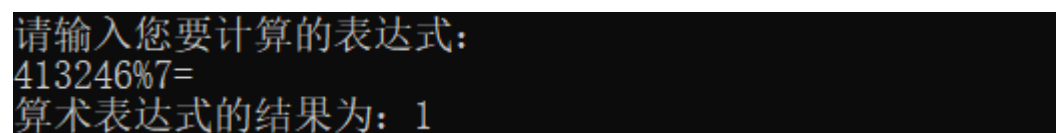
请输入您要计算的表达式:
 $75321/3=$
算术表达式的结果为: 25107

4.1.5 取余功能测试

测试用例: $413246\%7=$

预期结果: 1

实验结果:



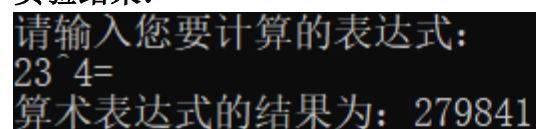
请输入您要计算的表达式:
 $413246\%7=$
算术表达式的结果为: 1

4.1.6 乘方功能测试

测试用例: $23^4=$

预期结果: 279841

实验结果:



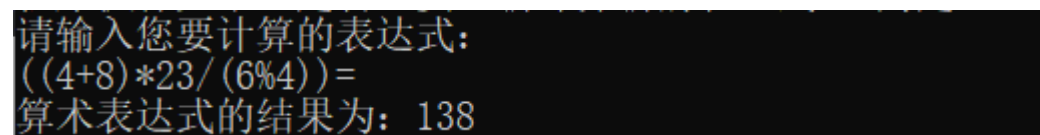
请输入您要计算的表达式:
 $23^4=$
算术表达式的结果为: 279841

4.1.7 带括号功能测试

测试用例: $((4+8)*23/(6\%4))=$

预期结果: 138

实验结果:



请输入您要计算的表达式:
 $((4+8)*23/(6\%4))=$
算术表达式的结果为: 138

4.1.8 一目运算符测试

测试用例: $-45 + (+58/2 - 8*7) =$

预期结果: 138

实验结果:

```
请输入您要计算的表达式:
-45+(+58/2-8*7)=
算术表达式的结果为: -72
```

4.1.9 多一目运算符测试

测试用例: $1+++23-----48=$

预期结果: 72

实验结果:

```
请输入您要计算的表达式:
1+++23-----48=
算术表达式的结果为: 72
```

4.1.10 综合测试

测试用例: $---78 + (7*8^3 - ((9\%4++89/2^2)*7/-9) + 8\%2) =$

预期结果: 3524.08

实验结果:

```
请输入您要计算的表达式:
---78+(7*8^3-((9%4++89/2^2)*7/-9)+8%2)=
算术表达式的结果为: 3524.08
```

4.1.11 运算数带小数的综合测试

测试用例: $(-8)/7+3.25^1.6*(8\%7+9/2.6--7.45/(-4.01*1.2)-6.3^(-3.9))=$

预期结果: 18.0566

实验结果:

```
请输入您要计算的表达式:
(-8)/7+3.25^1.6*(8%7+9/2.6--7.45/(-4.01*1.2)-6.3^(-3.9))=
算术表达式的结果为: 18.0566
```

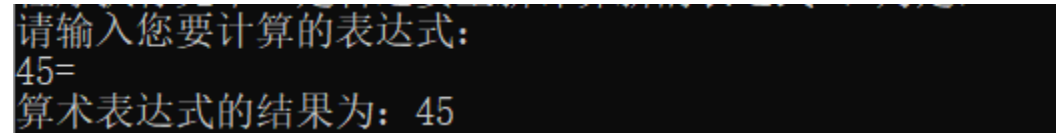
4.2 边界测试

4.2.1 表达式中无运算符

测试用例：45=

预期结果：45

实验结果：



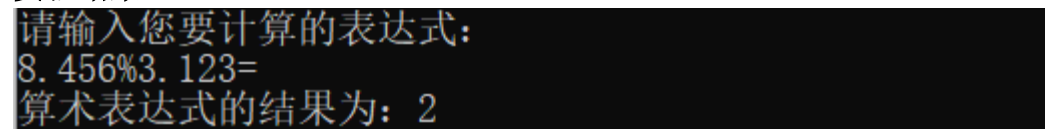
```
请输入您要计算的表达式：
45=
算术表达式的结果为： 45
```

4.2.2 取余运算时运算数是小数

测试用例：8.456%3.123=

预期结果：忽略小数进行运算，结果为 2

实验结果：



```
请输入您要计算的表达式：
8.456%3.123=
算术表达式的结果为： 2
```

4.3 出错测试

4.3.1 表达式最后没有等号

测试用例：78+87

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：

```
请输入您要计算的表达式：
78+87
表达式的结尾不为等号！
程序执行完毕，是否还要重新计算新的表达式（Y为是）：
```

4.3.2 表达式左右括号不匹配

测试用例：(45+(89*9)/7=

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：

```
请输入您要计算的表达式：
(45+(89*9)/7=
左括号和右括号数量未成功匹配！
程序执行完毕，是否还要重新计算新的表达式（Y为是）：
```

4.3.3 除法运算时除数为零

测试用例：5/0=

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：

```
请输入您要计算的表达式：
5/0=
除数不能是0！
输入的表达式不合法或遇到未知错误！
程序执行完毕，是否还要重新计算新的表达式（Y为是）：
```


4.3.4 取余运算时除数为零

测试用例：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：

```
请输入您要计算的表达式：
8%0=
除数不能是0!
输入的表达式不合法或遇到未知错误!
程序执行完毕，是否还要重新计算新的表达式（Y为是）：
```

4.3.5 表达式中出现未定义符号

测试用例：45+8/7&9=

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：

```
请输入您要计算的表达式：
45+8/7&9=
出现未定义的符号!
程序执行完毕，是否还要重新计算新的表达式（Y为是）：
```

4.3.6 操作数和操作符不匹配

测试用例：1-8/*9=

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：

```
请输入您要计算的表达式：
1-8/*9=
输入的表达式不合法或遇到未知错误!
程序执行完毕，是否还要重新计算新的表达式（Y为是）：
```

4.3.7 小数点位置不正确或多余

测试用例: $1+8/.4-7=$

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:

```
请输入您要计算的表达式:
1+8/.4-7=
在解析第7位字符时出现问题!
输入的表达式不合法或遇到未知错误!
程序执行完毕, 是否还要重新计算新的表达式(Y为是):
```

测试用例: $8+4.5.6=$

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:

```
请输入您要计算的表达式:
8+4.5.6=
在解析第6位字符时出现问题!
输入的表达式不合法或遇到未知错误!
程序执行完毕, 是否还要重新计算新的表达式(Y为是):
```

4.3.8 表达式为空

测试用例: $=$

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:

```
请输入您要计算的表达式:
=
输入的表达式不合法!
程序执行完毕, 是否还要重新计算新的表达式(Y为是):
```

4.3.9 表达式中只有括号

测试用例: $((()())())=$

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:

```
请输入您要计算的表达式:
((()())())=
出现空括号!
程序执行完毕, 是否还要重新计算新的表达式 (Y为是):
```

4.3.10 表达式最后有多个等号

测试用例: $1+2+3===$

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:

```
请输入您要计算的表达式:
1+2+3===
出现未定义的符号!
程序执行完毕, 是否还要重新计算新的表达式 (Y为是):
```

4.3.11 表达式中含有空括号

测试用例: $1+()+3=$

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:

```
请输入您要计算的表达式:
1+()+3=
出现空括号!
程序执行完毕, 是否还要重新计算新的表达式 (Y为是):
```

4.3.12 小数部分存在单目运算符

测试用例: $1+2.(-3)=$

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:

```
请输入您要计算的表达式:
1+2.(-3)=
在解析第5位字符时出现问题!
输入的表达式不合法或遇到未知错误!
程序执行完毕, 是否还要重新计算新的表达式(Y为是):
```

测试用例: $1+2.-3=$

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:

```
请输入您要计算的表达式:
1+2.-3=
在解析第5位字符时出现问题!
输入的表达式不合法或遇到未知错误!
程序执行完毕, 是否还要重新计算新的表达式(Y为是):
```