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: 沈星宇

学　　号: 1951576

指 导 教 师: 张颖

学院、 专业: 软件学院 软件工程

同济大学

Tongji University

一、 分析

（1） 应用背景

计算器作为一个很基本的功能有很大的应用范围，我们更习惯使用中缀表达式来进行求解，但计算机在处理的时候需要将其先转化为后缀表达式，然后进行计算。

要求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。

（2） 项目功能要求

1、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。

2、能处理单目运算符：+或-。

3、不考虑键入小数的情况。

二、 设计

（1） 数据结构设计

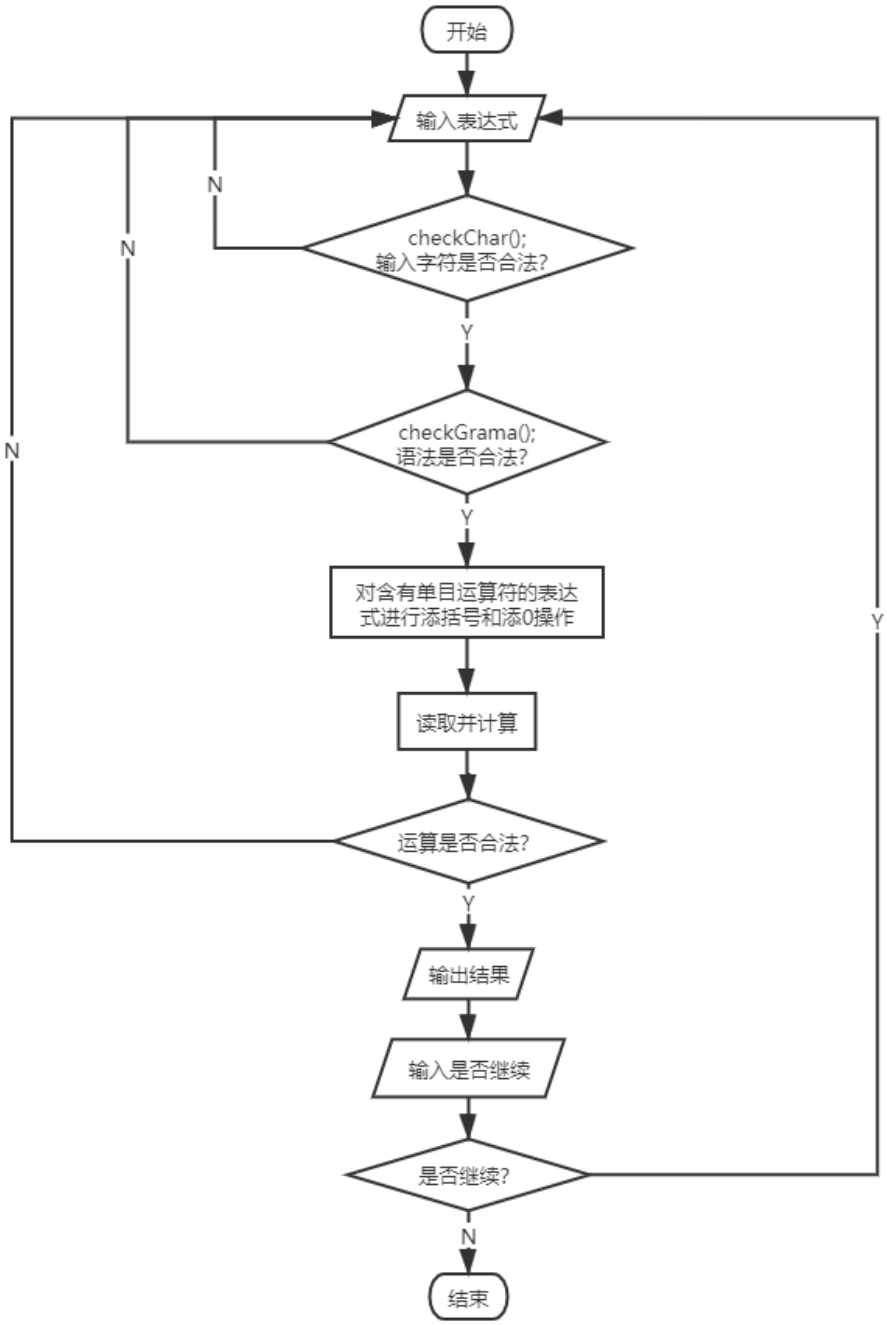
```
template<class T>
class Stack {
private:
    T* elements;
    int top;
    int maxSize;
    void overflowProcess();
public:
    Stack();
    ~Stack();
    bool push(T& value);
    bool pop(T& value);
    bool getTop(T& topNum);
    bool isEmpty();
    bool isFull();
    int getSize();
    void makeEmpty();
};
```

```
class Calculator {
private:
    Stack<double> numStack;
    Stack<char> operationStack;
    void addOperand(double value);
    bool get2Operand(double& left, double& right);
    bool doOperator(char op);
public:
    Calculator();
    //bool run(std::string str);
    void clear();

    bool checkChar(const std::string& str);
    void changeStr(std::string& str, int& position, int& length);
    bool checkGramma(std::string& str);

    double adjustNum(const std::string& str, int& position);
    bool calculate(const std::string& str);
    //std::string postfix(const std::string& str);
};
```

1、Stack 是栈，本项目利用栈后进先出的特性进行中缀转后缀同时计算。本项目使用的是线性栈，用一个 top 指向栈顶，以此来对栈顶元素进行操作。其中最主要的函数 push()和 pop()是将元素压入和弹出。

2、Calculator 是用于运算的计算器，其主要的成员变量是两个栈，分别用于存储运算符和运算数，其中 addOperand()用于往站内压入元素，get2Operand()用于从栈中弹出两个运算数，doOperator()用于将两个操作数和一个操作符进行运算。checkChar()、changeStr()、checkGrama()和 adjustNum()都是用于合法性检测的，最后将得到一个调整过的中缀表达式。calculate()对这个调整过的中准表达式进行计算。

（2） 程序流程设计



1、输入一个中缀表达式。
2、调用 checkChar();函数来判断该表达式是否合法，若非法则要求重新输入表达式。
3、调用 checkGrama();函数来判断表达式是否合法，若非法则要求重新输入表达式。同时对单目运算符进行判断，对表达式进行规范。
4、调用 calculate();函数根据中缀转后缀的规则进行计算，如果出现不合法运算会要求重新输入表达式，运算正确则输出结果。
5、询问是否继续

三、 实现

1、 main()函数的内部逻辑

（1） 首先输入一个中缀表达式。
（2） 调用 checkChar();函数来判断该表达式中是否含有除了数字和运算符（包括加减乘除、取余、乘方、等号和括号）之外的非法字符，若查到有非法字符则要求重新输入表达式。
（3） 调用 checkGrama();函数来判断表达式的语法是否合法，包括“是否以等号结尾”“括号是否匹配”“是否有连续的运算符出现”等基本的语法规则，如果查到非法的语法则要求重新输入表达式。同时在函数内部会对单目运算符进行判断，对表达式进行填括号和添 0 的操作。
（4） 调用 calculate();函数进行计算，根据中缀转后缀的规则进行计算，如果计算的过程中发现有“除数为 0”这样的不合法运算也会要求重新输入表达式，运算正确则输出结果。
（5） 询问是否继续，用户输入 y/Y 则继续请求输入表达式，若用户输入 n/N 则结束程序，若输入其他输入则请求输入 y/n

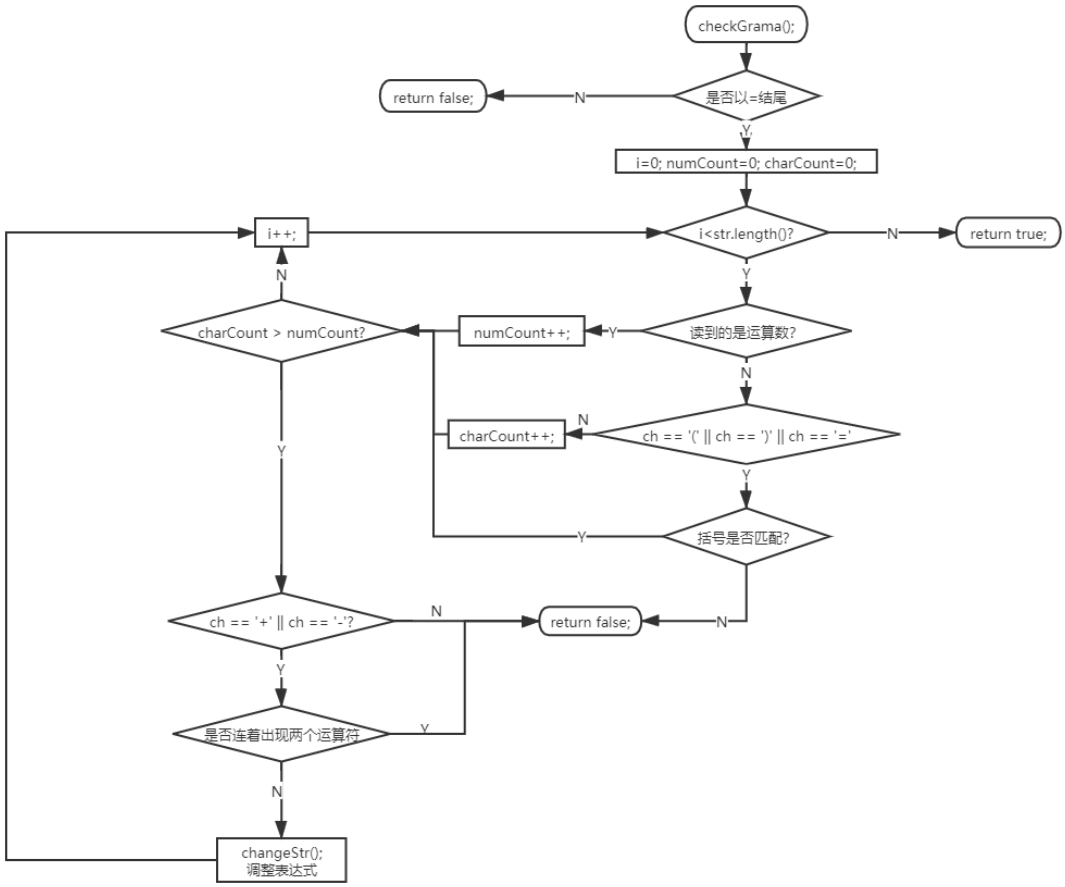
```
while (1) {
    std::cout << "----------------------------------------\n";
    std::string str;
    std::cout << "请输入表达式\n";
    std::cin >> str;
    Calculator cal;
    if (!cal.checkChar(str)) {
        continue;
    }
    if (!cal.checkGramma(str)) {
        continue;
    }
    //---------------------------------
    if (!cal.calculate(str)) {
        std::cout << "请重新输入\n";
        continue;
    }
    //---------------------------------
    if (continueOrNot() == CONTINUE) {
        continue;
    }
    else {
        break;
    }
}
```

2、 Calculator::checkChar()的实现

```cpp
bool Calculator::checkChar(const std::string& str) {
    int len = str.length();
    for (int i = 0; i < len; i++) {
        if (!((str[i] >= '0' && str[i] <= '9') ||
            str[i] == '+' || str[i] == '-' ||
            str[i] == '*' || str[i] == '/' ||
            str[i] == '%' || str[i] == '^' ||
            str[i] == '(' || str[i] == ')' || str[i] == '=')) {
            std::cout << "输入不合法，请重新输入\n";
            return false;
        }
    }
    return true;
}
```

(1) 对于输入的字符串进行逐位的检查，若含有除数字和运算符（包括加减乘除、取余、乘方、等号和括号）之外的非法字符则 return false,反之 return true

3、 Calculator::checkGrama()的实现



用于检测输入表达式的语法是否正确。

（1） 首先检测是否以等号结尾，如果不是则 return false

（2） 循环遍历字符串的元素，判断读到的字符是数字还是运算符。如果读到的是数字，则直到读到下一个运算符为止，将 numCount++；如果读到的是符号，则还要对运算符和括号进行分类，如果是运算符号则将 charCount++,如果是括号则利用栈进行括号匹配的判断。括号匹配失败

则 return false

（3） 一旦出现 charCount > numCount 的情况，则必定是包含了单目运算符，对其前后的语法进行检测（即不能有连续两个符号出现），一旦不合法则 return false。在确定了是正确语法的单目运算符之后，调用 changeStr() 函数对表达式进行调整。目标是将形如(-2+1)这样的字串补充成((0-2)+1)的形式。

（4） 所有的操作都执行成功则 return true，同时因为传入的参数是引用，所以 str 也已经是经过调整后的字符串了。

```
bool Calculator::checkGramma(std::string& str) {
    int len = str.length();
    int numCount = 0, charCount = 0;
    Stack<char>charStack;
    //----------------------------------------
    if (str[len - 1] != '=') {
        std::cout << "输入不合法! 请以=结尾! \n";
        return false;
    }
    //----------------------------------------
    for (int i = 0; i < len; i++) {
        char ch = str[i];
        if (ch >= '0' && ch <= '9') {
            for (; i < len; i++) {
                if (!(str[i+1] >= '0' && str[i+1] <= '9')) {
                    numCount++;
                    break;
                }
            }
        }
        else {
            if (!(ch == '(' || ch == ')' || ch == '=')) {
                charCount++;
            }
            //---------------------------------------
```

```
            else if (ch == '(') {
                if (!charStack.isFull()) {
                    charStack.push(ch);
                }
            }
            else if (ch == ')') {
                if (!charStack.isEmpty()) {
                    charStack.pop(ch);
                }
                else {
                    std::cout << "括号不匹配! \n";
                    return false;
                }
            }
            else {
                if (!charStack.isEmpty()) {
                    std::cout << "括号不匹配! \n";
                    return false;
                }
            }
        }
        //----------------------------------------
```

```
    //说明包含单目运算符
    if (charCount > numCount) {
        if (ch == '+' || ch == '-') {//只有+和-是单目运算符
            if (i != 0 && (!(str[i - 1] >= '0' && str[i - 1] <= '9'))
                && (str[i - 1] != '(')) {
                return false;
            }
            //因为最后肯定是以等号结尾，所以不用怕str[i+1]溢出
            if (str[i + 1] >= '0' && str[i + 1] <= '9') {
                changeStr(str, i, len);
                numCount += 2;
            }
            else {
                return false;
            }
        }
        else {
            return false;
        }
    }
}
return true;
```

4、 Calculator::changeStr()的实现

利用 changeStr()能将单目运算符转化成双目运算符，即目标是将形如(-2+1)这样的字串补充成((0-2)+1)的形式。

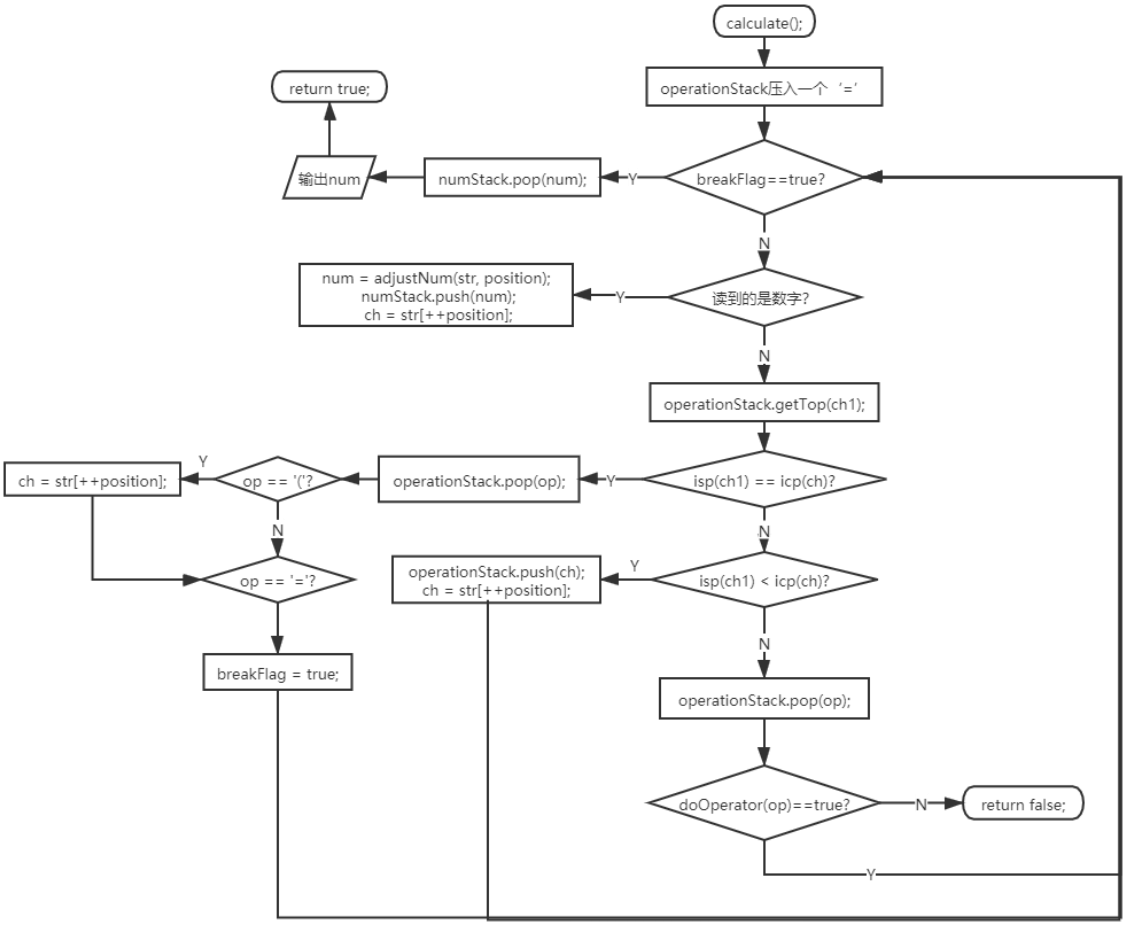
position 代表的是单目运算符的位置，在运算符前面插入字符‘0’，在‘0’的前面插入字符‘(’。

在处理右括号的时候要注意数字可能不止一位，所以要用循环查找到不为数字的部分，在它的前面插入字符‘)’。

```
void Calculator::changeStr(std::string& str, int& position, int& length) {
    //插入0
    str = str.insert(position, 1, '0');
    length++; position++;
    //下面进行补括号
    str = str.insert(position - 1, 1, '(');
    length++; position++;
    //因为position+1的位置肯定是数字，所以不用怕i = position + 1溢出
    for (int i = position + 1; i <= length - 1; i++) {
        if (!(str[i] >= '0' && str[i] <= '9')) {
            str = str.insert(i, 1, ')');
            length++;
            position = i;
            break;
        }
    }
}
```

5、 Calculator::calculate()的实现

calculate();函数进行计算，根据中缀转后缀的规则进行计算，如果计算的过程中发现有“除数为 0”这样的不合法运算也会要求重新输入表达式，运算正确则输出结果。



(1) 首先初始化两个栈 operationStack 和 numStack，分别用于存储操作数和运算符。往 operationStack 当中压入一个‘=’作为判断结束循环的依据。

(2) 从左到右处理各个元素直到末尾，可能遇到三种情况：

1、 遇到操作数，在经过 adjustNum()的转化之后压入 numStack

2、 遇到界限符。遇到“(”直接入 operationStack；遇到“)”则依次弹出 operationStack 内的运算符，直到弹出“(”为止，同时每弹出一个运算符，就需要调用 doOperator()再弹出两个 numStack 并执行相应运算，运算结果再压回 numStack。注意弹出“(”的时候不执行操作，直接后移。

3、遇到运算符，依次弹出 operationStack 中有限集高于或等于当前运算符的所有运算符，并调用 doOperator()再弹出两个 numStack 并执行相应运算，运算结果再压回 numStack。

操作符 ch	=	(	+,-	*,/,%	^	)
Isp	0	1	3	5	7	8
icp	0	8	2	4	6	1

```
operationStack.push(ch);
ch = str[position];
while (1) {
    if (ch >= '0' && ch <= '9') {
        num = adjustNum(str, position);
        numStack.push(num);
        ch = str[++position];
    }
    else {
        operationStack.getTop(ch1);
        if (isp(ch1) < icp(ch)) {
            operationStack.push(ch);
            ch = str[++position];
        }
        else if (isp(ch1) > icp(ch)) {
            operationStack.pop(op);
            if (!doOperator(op)) {
                return false;
            }
        }
        else {
            operationStack.pop(op);
            if (op == '(') {
                ch = str[++position];
            }
            if (op == '=') {
                breakFlag = true;
            }
        }
    }
    if (breakFlag) {
        break;
    }
}
numStack.pop(num);
std::cout << num << '\n';
```

```
bool Calculator::doOperator(char op) {
    double left, right, value;
    if (get2Operand(left, right)) {
        switch (op) {
        case '+':
            value = left + right;
            numStack.push(value);
            break;
        case '-':
            value = left - right;
            numStack.push(value);
            break;
        case'*':
            value = left * right;
            numStack.push(value);
            break;
        case'/':
            if (right == 0.0) {
                std::cout << "语法错误! 除数不能为0! \n";
                return false;
            }
            value = left / right;
            numStack.push(value);
            break;
        case '%':
            if ((int)(left * 100000) % 100000 != 0 ||
                (int)(right * 100000) % 100000 != 0) {
                std::cout << "语法错误! 小数无法进行取余! \n";
                return false;
            }
            value = (int)left % (int)right;
            numStack.push(value);
            break;
        case'^':
            value = 1;
            for (int i = 0; i < right; i++) {
                value *= left;
            }
            numStack.push(value);
            break;
        }
    }
    return true;
}
```

6、 Calculator::doOperator()的实现

（1） 首先调用 get2Operand()函数，从 numStack()当中弹出两个操作数

（2） 然后根据 op 的类型执行相应的操作

（3） 其中对除数为零等运算不合法的情况进行判定

7、 Calculator::get2Operand()的实现

```
bool Calculator::get2Operand(double& left, double& right) {
    if (numStack.isEmpty()) {
        std::cerr << "缺少右操作数! \n" << std::endl;
        return false;
    }
    numStack.pop(right);
    if (numStack.isEmpty()) {
        std::cerr << "缺少左操作数! \n" << std::endl;
        return false;
    }
    numStack.pop(left);
    return true;
}
```

8、 continueOrNot()的实现

控制程序的循环和调度

```
int continueOrNot() {
    while (1) {
        std::cout << "--------------------------------";
        std::cout << "是否继续 (y, n) ? ";
        char command;
        std::cin >> command;
        if (command == 'y' || command == 'Y') {
            return CONTINUE;
        }
        else if (command == 'n' || command == 'N') {
            return STOP;
        }
        else {
            std::cout << "请输入合法输入，y或n\n";
            continue;
        }
    }
}
```

四、 测试

1、合法性检测

不以等号结尾、输入非法字符、不符合中缀表达式语法、除数为 0 都会被要求重新输入

```
D:\VS文件\数据结构课程设计\Project4\Release\Project4
----------------------------------------
请输入表达式
a+1=
输入不合法，请重新输入
----------------------------------------
请输入表达式
++2-1=
----------------------------------------
请输入表达式
)1+2=
括号不匹配！
----------------------------------------
请输入表达式
2+2=1
输入不合法！请以=结尾！
----------------------------------------
请输入表达式
999*-999=
----------------------------------------
请输入表达式
1+1
输入不合法！请以=结尾！
----------------------------------------
请输入表达式
12/0=
语法错误！除数不能为0！
请重新输入
```

2、一般情况

```
请输入表达式
-2*(3+5)+2^3/4=
-14
--------------------------------是否继续（y，n）？y
----------------------------------------
请输入表达式
2^4/8-(+2+8)%3=
1
--------------------------------是否继续（y，n）？n
请按任意键继续. . .
```

3、 内嵌多余的括号的情况

内嵌的多余括号将会被忽视

```
----------------------------------------
请输入表达式
(()1+1)=
2
--------------------------------是否继续（y，n）？
```

4、 产生小数的情况

（1）能够正常输出小数的结果

（2）如果出现对小数取余会产生报错

```
----------------------------------------
请输入表达式
2^2/3=
1.33333
--------------------------------是否继续（y，n）？y
----------------------------------------
请输入表达式
2^2/3%1=
语法错误！小数无法进行取余！
请重新输入
----------------------------------------
```

5、对于输入的本身是数字的情况

能够直接输出

```
----------------------------------------
请输入表达式
-1=
-1
--------------------------------是否继续（y，n）？
```