1.题目名称

算术表达式求值

2.代码行数

261行

3.算法思想

先判断表达式是否合理，不合理直接退出。

利用栈这一数据结构来实现表达式求值。如果遇到括号，则把括号里的表达式拿出来单独求值，再回到原表达式。

4.主要/核心函数分析

isExpressionValid

```
//判断表达式是否合理
bool isExpressionValid(const std::string& expression) {
    std::stack<char> parenthesesStack;

    for (char c : expression) {
        if (c == '(') {      //判断括号
            parenthesesStack.push(c);
        } else if (c == ')') {
            if (parenthesesStack.empty() || parenthesesStack.top() != '(')
{
                std::cout << "Invalid expression: Unmatched closing
parenthesis ')'." << std::endl;
                return false;
            }
            parenthesesStack.pop();
        }
    }

    if (!parenthesesStack.empty()) {
        std::cout << "Invalid expression: Unmatched opening parenthesis
'('." << std::endl;
        return false;
    }

    std::string operators = "+-*/%";
    for (size_t i = 0; i < expression.length(); i++) {
        if (operators.find(expression[i]) != std::string::npos) {
            if (i == 0 || i == expression.length() - 1 ||
operators.find(expression[i + 1]) != std::string::npos) {
                std::cout << "Invalid expression: Invalid operator usage."
<< std::endl;
                return false;
            }
        }else if(expression[i]=='.'||isdigit(expression[i])){
            continue;
```

```
        }else{
            cout<<"非法字符"<<endl;
            return false;
        }
    }

    return true;
}
```

1. 括号必须匹配：左括号 `(` 必须有一个相应的右括号 `)` 来匹配。
2. 运算符的使用必须正确：运算符不能出现在表达式的开始、结束或连续出现。
3. 不存在非法字符。

Load_Stack

```
// 加载栈并计算结果
double Load_Stack(string std, stack<double>& Num, stack<char>& Symbol) {
    int i = 0;
    char k;
    int len = std.length();
    while (i < len) {
        k = std[i++];
        if (('0' <= k && k <= '9') || k == '.') {    // 如果是数字字符或小数点
            string numStr;
            if(i==len){
                numStr+=k;
            }
            while ((('0' <= k && k <= '9') || k == '.') && i < len) {
 // 处理数字和小数点
                numStr += k;
                k = std[i++];
                if(i==len)
                    numStr+=k;
            }
            if(i!=len){
                i--;
            }
            double num = stod(numStr);    // 将字符串转换为浮点数
            Num.push(num);    // 将数字压入操作数栈
        } else {    // 如果是运算符字符
            switch (k) {
                case '+':
                case '-':
                    Symbol.push(k);    // 将运算符压入运算符栈
                    break;
                case '*':
                    k = std[i++];
                    if (('0' <= k && k <= '9') || k == '.') {
                        string numStr;
                        if(i==len){
                            numStr+=k;
                        }
                        while ((('0' <= k && k <= '9') || k == '.') && i <
len) {
                            numStr += k;
                            k = std[i++];
                            if(i==len)
```

```
                                numStr+=k;
                        }
                        if(i!=len){
                            i--;
                        }
                        double num = stod(numStr);    // 将字符串转换为浮点数
                        double temp = Num.top() * num;    // 计算乘法结果
                        Num.pop();
                        Num.push(temp);
                    } else if (k == '(') {
                        string op;
                        while (1) {
                            char stTemp = std[i++];
                            if (stTemp == ')') {
                                break;
                            }
                            op += stTemp;
                        }
                        double temp = Load_Stack(op, TempNum, TempSymbol);
    // 递归计算括号内的表达式结果
                        temp *= Num.top();    // 计算乘法结果
                        stack<double>().swap(TempNum);    // 清空临时操作数栈
                        stack<char>().swap(TempSymbol);    // 清空临时运算符
栈
                        Num.pop();
                        Num.push(temp);
                    }
                    break;
                case '/':
                    k = std[i++];
                    if (('0' <= k && k <= '9') || k == '.') {
                        string numStr;
                        if(i==len){
                            numStr+=k;
                        }
                        while ((('0' <= k && k <= '9') || k == '.') && i <
len) {
                            numStr += k;
                            k = std[i++];
                            if(i==len)
                                numStr+=k;
                        }
                        if(i!=len){
                            i--;
                        }
                        double num = stod(numStr);    // 将字符串转换为浮点数
                        double temp = Num.top() / num;    // 计算除法结果
                        Num.pop();
                        Num.push(temp);
                    } else if (k == '(') {
                        string op;
                        while (1) {
                            char stTemp = std[i++];
                            if (stTemp == ')') {
                                break;
                            }
                            op += stTemp;
                        }
```

```
                        double temp = Load_Stack(op, TempNum, TempSymbol);
    // 递归计算括号内的表达式结果
                        temp = Num.top() / temp;     // 计算除法结果
                        stack<double>().swap(TempNum);     // 清空临时操作数栈
                        stack<char>().swap(TempSymbol);     // 清空临时运算符
栈
                        Num.pop();
                        Num.push(temp);
                    }
                    break;
                case '%':
                    k = std[i++];
                    if (('0' <= k && k <= '9') || k == '.') {
                        string numStr;
                        if(i==len){
                            numStr+=k;
                        }
                        while ((('0' <= k && k <= '9') || k == '.') && i <
len) {
                            numStr += k;
                            k = std[i++];
                            if(i==len)
                                numStr+=k;
                        }
                        if(i!=len){
                            i--;
                        }
                        double num = stod(numStr);     // 将字符串转换为浮点数
                        double temp = fmod(Num.top(), num);     // 计算取模结
果
                        Num.pop();
                        Num.push(temp);
                    } else if (k == '(') {
                        string op;
                        while (1) {
                            char stTemp = std[i++];
                            if (stTemp == ')') {
                                break;
                            }
                            op += stTemp;
                        }
                        double temp = Load_Stack(op, TempNum, TempSymbol);
    // 递归计算括号内的表达式结果
                        temp = fmod(Num.top(), temp);     // 计算取模结果
                        stack<double>().swap(TempNum);     // 清空临时操作数栈
                        stack<char>().swap(TempSymbol);     // 清空临时运算符
栈
                        Num.pop();
                        Num.push(temp);     // 将括号内表达式的结果压入操作数栈
                        break;
                    }
                    break;
                case '(':
                    string op;
                    while (1) {
                        char stTemp = std[i++];
                        if (stTemp == ')') {
                            break;
```

```
                        }
                        op+= stTemp;
                    }
                    double temp = Load_Stack(op, TempNum, TempSymbol);
// 递归计算括号内的表达式结果
                    stack<double>().swap(TempNum);     // 清空临时操作数栈
                    stack<char>().swap(TempSymbol);     // 清空临时运算符栈
                    Num.push(temp);     // 将括号内表达式的结果压入操作数栈
                    break;
                }
            }
        }

        double sum = Num.top();     // 最终计算结果
        Num.pop();
        while (!Num.empty() && !Symbol.empty()) {
            switch (Symbol.top()) {
                case '+':
                    sum += Num.top();     // 加法运算
                    Num.pop();
                    break;
                case '-':
                    sum = Num.top() - sum;     // 减法运算
                    Num.pop();
                    break;
            }
            Symbol.pop();
        }
        return sum;
}
```

将数字和运算符分别压入对应栈。

根据当前字符选择不同的操作：加法、减法或乘法或除法。对于加法和减法，将运算符压入运算符栈。对于乘法和除法，从操作数栈中弹出两个数字，计算它们的乘积/商，并将结果压回数字栈。对于左括号，进入一个新的循环来处理括号内的表达式，随后函数调用自身计算表达式结果。

5.测试数据(规模,测试次数)

规模:运算精度以及大小不超过double类型

测试次数:13

测试用例:见测试文件

6.运行结果

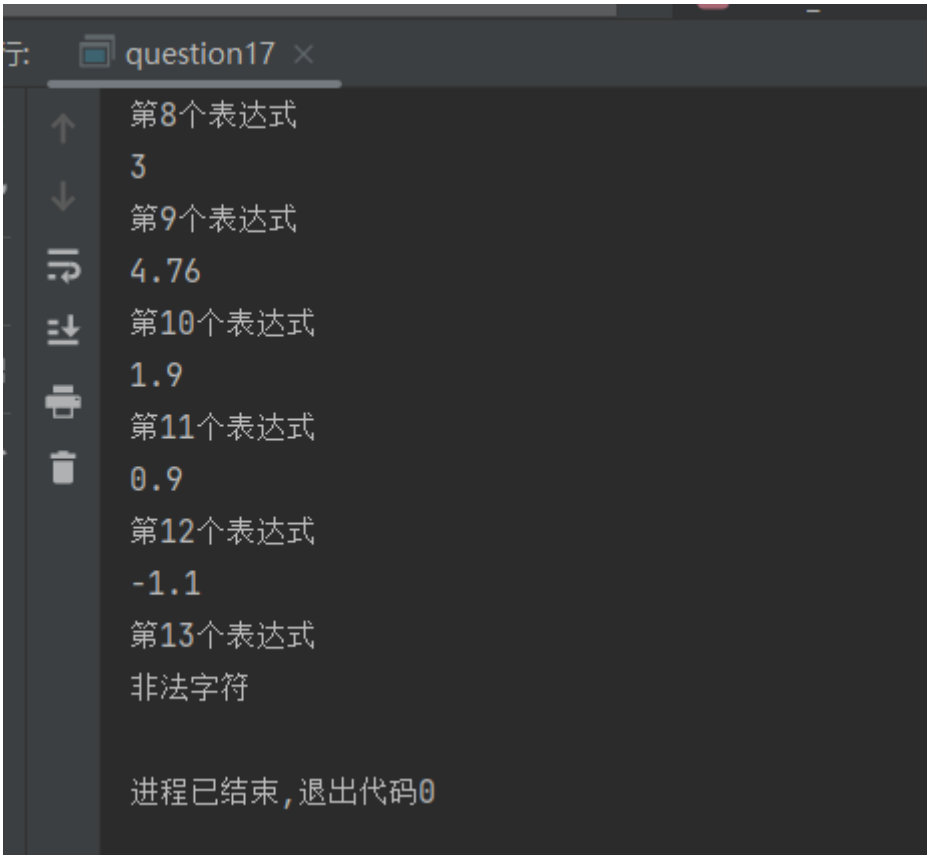
```
F:\data_structure\Choice\question17\cmake-build-debug\question17.exe
第1个表达式
非法字符
第2个表达式
1
第3个表达式
Invalid expression: Invalid operator usage.
第4个表达式
Invalid expression: Invalid operator usage.
第5个表达式
```

```
Invalid expression: Unmatched opening parenthesis '('.
第6个表达式
非法字符
第7个表达式
4.6
第8个表达式
3
第9个表达式
4.76
第10个表达式
1.9
第11个表达式
0.9
第12个表达式
-1.1
第13个表达式
非法字符

进程已结束,退出代码0

```

7.时间复杂度分析

代码的时间复杂度是O(m * n)，其中m是文件中表达式的数量，n是表达式的平均长度。

8.结果截屏图片



9.心得体会

对中缀表达式求值更加熟悉与理解。同时，对栈这一数据结构的使用更加得心应手。