

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

```
1 //判断表达式是否合理
2 bool isExpressionValid(const std::string& expression) {
3     std::stack<char> parenthesesStack;
4
5     for (char c : expression) {
6         if (c == '(') { //判断括号
7             parenthesesStack.push(c);
8         } else if (c == ')') {
9             if (parenthesesStack.empty() || parenthesesStack.top() != '(')
10 {
11                 std::cout << "Invalid expression: Unmatched closing
parenthesis ')." << std::endl;
12                 return false;
13             }
14             parenthesesStack.pop();
15         }
16     }
17
18     if (!parenthesesStack.empty()) {
19         std::cout << "Invalid expression: Unmatched opening parenthesis
'(')." << std::endl;
20         return false;
21     }
22
23     std::string operators = "+-*/%";
24     for (size_t i = 0; i < expression.length(); i++) {
25         if (operators.find(expression[i]) != std::string::npos) {
26             if (i == 0 || i == expression.length() - 1 ||
operators.find(expression[i + 1]) != std::string::npos) {
27                 std::cout << "Invalid expression: Invalid operator usage."
<< std::endl;
28                 return false;
29             }
30         } else if (expression[i] == '.' || isdigit(expression[i])) {
31             continue;
32         }
33     }
34 }
```

```

31         }else{
32             cout<<"非法字符"<<endl;
33             return false;
34         }
35     }
36
37     return true;
38 }

```

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

```

1  // 加载栈并计算结果
2  double Load_Stack(string std, stack<double>& Num, stack<char>& Symbol) {
3      int i = 0;
4      char k;
5      int len = std.length();
6      while (i < len) {
7          k = std[i++];
8          if (('0' <= k && k <= '9') || k == '.') {    // 如果是数字字符或小数点
9              string numStr;
10             if(i==len){
11                 numStr+=k;
12             }
13             while (((('0' <= k && k <= '9') || k == '.') && i < len) {
14                 // 处理数字和小数点
15                 numStr += k;
16                 k = std[i++];
17                 if(i==len)
18                     numStr+=k;
19             }
20             if(i!=len){
21                 i--;
22             }
23             double num = stod(numStr);    // 将字符串转换为浮点数
24             Num.push(num);    // 将数字压入操作数栈
25         } else {    // 如果是运算符字符
26             switch (k) {
27                 case '+':
28                 case '-':
29                     Symbol.push(k);    // 将运算符压入运算符栈
30                     break;
31                 case '*':
32                     k = std[i++];
33                     if (('0' <= k && k <= '9') || k == '.') {
34                         string numStr;
35                         if(i==len){
36                             numStr+=k;
37                         }
38                         while (((('0' <= k && k <= '9') || k == '.') && i <
39                             len) {
40                             numStr += k;
41                             k = std[i++];
42                             if(i==len)

```

```

41         numStr+=k;
42     }
43     if(i!=len){
44         i--;
45     }
46     double num = stod(numStr);    // 将字符串转换为浮点数
47     double temp = Num.top() * num;    // 计算乘法结果
48     Num.pop();
49     Num.push(temp);
50 } else if (k == '(') {
51     string op;
52     while (1) {
53         char stTemp = std[i++];
54         if (stTemp == ')') {
55             break;
56         }
57         op += stTemp;
58     }
59     double temp = Load_Stack(op, TempNum, TempSymbol);
60     // 递归计算括号内的表达式结果
61     temp *= Num.top();    // 计算乘法结果
62     stack<double>().swap(TempNum);    // 清空临时操作数栈
63     stack<char>().swap(TempSymbol);    // 清空临时运算符
64     Num.pop();
65     Num.push(temp);
66 }
67 break;
68 case '/':
69     k = std[i++];
70     if (('0' <= k && k <= '9') || k == '.') {
71         string numStr;
72         if(i==len){
73             numStr+=k;
74         }
75         while (((('0' <= k && k <= '9') || k == '.') && i <
76             len) {
77             numStr += k;
78             k = std[i++];
79             if(i==len)
80                 numStr+=k;
81         }
82         if(i!=len){
83             i--;
84         }
85         double num = stod(numStr);    // 将字符串转换为浮点数
86         double temp = Num.top() / num;    // 计算除法结果
87         Num.pop();
88         Num.push(temp);
89     } else if (k == '(') {
90         string op;
91         while (1) {
92             char stTemp = std[i++];
93             if (stTemp == ')') {
94                 break;
95             }
96             op += stTemp;
97         }

```

```

96         double temp = Load_Stack(op, TempNum, TempSymbol);
// 递归计算括号内的表达式结果
97         temp = Num.top() / temp;    // 计算除法结果
98         stack<double>().swap(TempNum);    // 清空临时操作数栈
99         stack<char>().swap(TempSymbol);    // 清空临时运算符
栈
100         Num.pop();
101         Num.push(temp);
102     }
103     break;
104     case '%':
105         k = std[i++];
106         if (('0' <= k && k <= '9') || k == '.') {
107             string numStr;
108             if(i==len){
109                 numStr+=k;
110             }
111             while (((('0' <= k && k <= '9') || k == '.') && i <
len) {
112                 numStr += k;
113                 k = std[i++];
114                 if(i==len)
115                     numStr+=k;
116             }
117             if(i!=len){
118                 i--;
119             }
120             double num = stod(numStr);    // 将字符串转换为浮点数
121             double temp = fmod(Num.top(), num);    // 计算取模结
果
122             Num.pop();
123             Num.push(temp);
124         } else if (k == '(') {
125             string op;
126             while (1) {
127                 char stTemp = std[i++];
128                 if (stTemp == ')') {
129                     break;
130                 }
131                 op += stTemp;
132             }
133             double temp = Load_Stack(op, TempNum, TempSymbol);
// 递归计算括号内的表达式结果
134             temp = fmod(Num.top(), temp);    // 计算取模结果
135             stack<double>().swap(TempNum);    // 清空临时操作数栈
136             stack<char>().swap(TempSymbol);    // 清空临时运算符
栈
137             Num.pop();
138             Num.push(temp);    // 将括号内表达式的结果压入操作数栈
139             break;
140         }
141         break;
142         case '(':
143             string op;
144             while (1) {
145                 char stTemp = std[i++];
146                 if (stTemp == ')') {
147                     break;

```

```

148         }
149         op += stTemp;
150     }
151     double temp = Load_Stack(op, TempNum, TempSymbol);
    // 递归计算括号内的表达式结果
152     stack<double>().swap(TempNum);    // 清空临时操作数栈
153     stack<char>().swap(TempSymbol);    // 清空临时运算符栈
154     Num.push(temp);    // 将括号内表达式的结果压入操作数栈
155     break;
156 }
157 }
158 }
159
160 double sum = Num.top();    // 最终计算结果
161 Num.pop();
162 while (!Num.empty() && !Symbol.empty()) {
163     switch (Symbol.top()) {
164         case '+':
165             sum += Num.top();    // 加法运算
166             Num.pop();
167             break;
168         case '-':
169             sum = Num.top() - sum;    // 减法运算
170             Num.pop();
171             break;
172     }
173     Symbol.pop();
174 }
175 return sum;
176 }

```

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

```

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

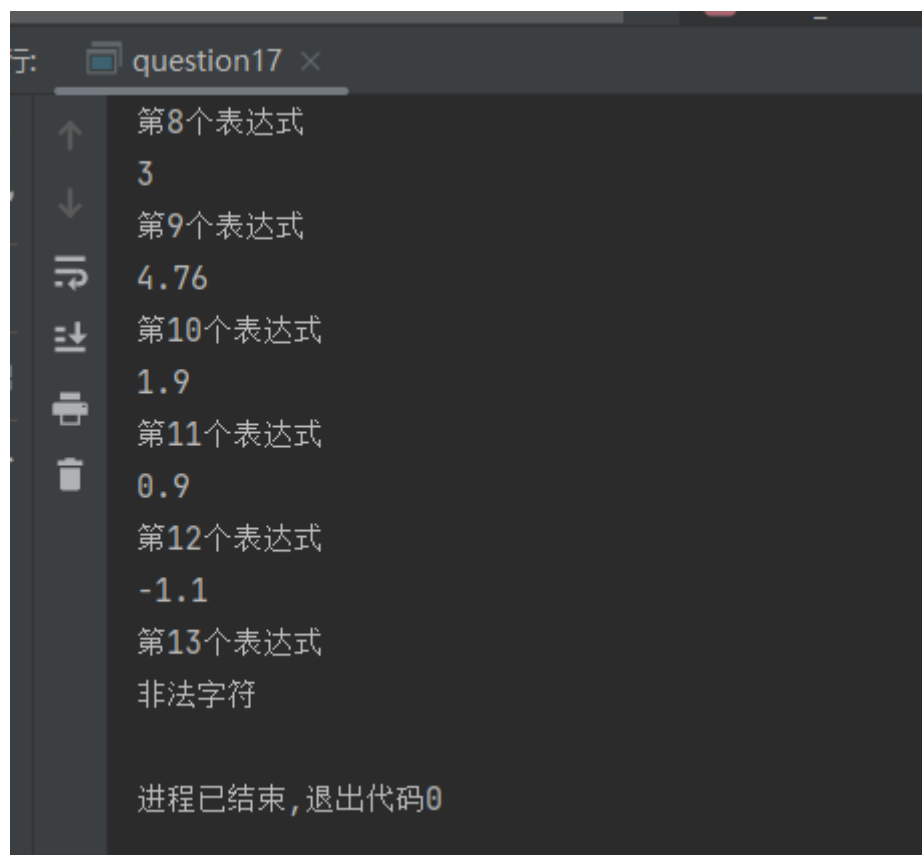
```

```
11 Invalid expression: Unmatched opening parenthesis '('.
12 第6个表达式
13 非法字符
14 第7个表达式
15 4.6
16 第8个表达式
17 3
18 第9个表达式
19 4.76
20 第10个表达式
21 1.9
22 第11个表达式
23 0.9
24 第12个表达式
25 -1.1
26 第13个表达式
27 非法字符
28
29 进程已结束,退出代码0
30
```

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

对后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。