1 设计思路

1.1 基本思路

- 使用栈来存储数字和运算符。
- 遇到数字时，将其压入数字栈。
- 遇到运算符时：
  - 如果运算符栈为空，或者运算符栈顶为左括号，则直接压入运算符栈。
  - 如果运算符栈顶为右括号，则将运算符栈中的运算符依次弹出并计算，直到遇到左括号。
  - 若为其他运算符，则比较当前运算符与栈顶运算符的优先级。若当前优先级较低，则先弹出栈顶运算符并计算，再将当前运算符压入。
- 遍历表达式结束后，将栈中剩余的运算符依次弹出并计算。

1.2 优先级规则

- 运算符优先级从高到低依次为：括号、乘除、加减。
- 括号优先级另外计算，在遇到右括号时进行计算。

2 程序实现

2.1 数据结构设计

- **数字栈：**用于存储操作数。
- **运算符栈：**用于存储运算符和括号。

2.2 算法步骤

1. 初始化两个栈：数字栈和运算符栈。
2. 从左到右扫描表达式中的字符：
   - 如果是数字，将其压入数字栈。
   - 如果是左括号或运算符，将其压入运算符栈。
   - 如果是右括号：
     - 弹出运算符栈顶的运算符，并从数字栈弹出对应的操作数进行计算。
     - 将计算结果压入数字栈，直到遇到左括号。
   - 如果是运算符：
     - 比较当前运算符与栈顶运算符的优先级。
     - 若当前运算符优先级较低，则弹出栈顶运算符并计算，重复此过程，直至满足条件后将当前运算符压入。
3. 扫描完成后，依次弹出运算符栈中的运算符进行计算，直到栈为空。

2.3 字符串转数字

解析字符串为数字是表达式求值的核心操作之一。C++ 标准库中的 `std::stod` 函数提供了高效且可靠的解决方案，其特点如下：

1. **功能全面**：`std::stod` 支持解析常见的数字格式，包括：
   - 整数和小数（例如 `"123"` 或 `"123.45"`）。
   - 负数（例如 `"-123"` 或 `"-123.45"`）。
   - 科学计数法（例如 `"1.23e4"`）。
2. **错误处理**：`std::stod` 可以处理并报告解析过程中可能出现的错误，例如：
   - 非法字符（例如 `"abc"`）。

值得注意的是，`std::stod` 遇到非法格式的输入（如 1.1.1，1e1e1）会部分解析字符串，直到遇到第一个无法解析的部分为止。因此我们需要额外处理多个小数点输入和多个指数输入的情况。

2.4 错误处理

我们根据表达式中涉及的字符进行分类。具体来说，我们将表达式中的字符分为以下几类：

- **数字字符**：包括 0-9 以及.。(这里我们认为.1 和 1. 是合法输入，这是大部分计算器的默认行为)。
- **运算符字符**：包括 +、-、*、/。
- **括号字符**：包括 (和)。
- **指数字符**：包括 e。

非法表达式是上述不同类型表达式的非法组合。

由于部分指数字符和数字字符相关的非法表达式会在 `std::stod` 解析时被处理，因此我们只需在此处对部分涉及指数字符的非法表达式进行判断。非法表达式大体可以用以下方法判断：

- 表达式结尾只能是数字字符和括号字符
- 括号数量必须匹配
- 运算符不能连用，但 1+-1 表示 1 与其相反数之和，因此 1+-1 是合法输入
- 运算符不能在字符串首位，但-1 表示 1 的相反数，因此-1 是合法输入
- 括号内部表达式也需要合法
- 两个括号之间必须有运算符
- 指数字符 e 后必须是数字字符，但 1e-1 是合法输入
- 一个数字中有复数个小数点或复数个指数字符
- 除数是 0

特别地，为了代码实现便捷，我们不允许空格出现。事实上，绝大多数计算器中也没有“空格”这一输入，因此这是合理的设计。我们将前七点在 `isValidExpression` 中实现，第八点在 `StringToNumber` 中实现，最后一点在 `applyOperator` 中实现。

2.5 代码实现

2.5.1 字符串转数字相关函数

isValidExpression 函数的实现如下：

```
bool isValidExpression(const string& expr) {
    if (expr.empty()) return false;
    if ((isOperator(expr[0]) && expr[0] != '-') || isOperator(expr.back())) return false;
    if (expr.back() == 'e') return false;

    int openParenCount = 0;
    for (size_t i = 0; i < expr.size(); ++i) {
        if (isLeftParenthesis(expr[i])) openParenCount++;
        if (isRightParenthesis(expr[i])) openParenCount--;
        if (openParenCount < 0) return false; // 右括号超过左括号
        if (isLeftParenthesis(expr[i]) && isRightParenthesis(expr[i+1])) return false; // ()不合法
        if (isOperator(expr[i])) {
            if ((i == 0 && isOperator(expr[1])) || (i > 0 && expr[i-1] == '(' && isOperator(expr[i
                +1]))) return false; // --1+1 不合法 (--1) (+-1)不合法
            if (i > 0 && isOperator(expr[i-1]) && expr[i] != '-') return false; // 1++1不合法 这里不
                会出现首二个都是运算符的情况
            if (i > 1 && isOperator(expr[i-1]) && isOperator(expr[i-2])) return false; // 1---1不合
                法
            if (i > 0 && expr[i] != '-' && expr[i-1] == '(') return false; // (+1)不合法
            if (i > 0 && expr[i-1] == '(' && expr[i+1] == ')') return false; // (-)不合法
            if (expr[i+1] == ')') return false; // (1+)不合法
        }
        if (expr[i] == ' ') return false; // 不允许空格
        if (i > 0 && expr[i] == '(' && (!isOperator(expr[i-1]) && expr[i-1] != '(')) return false;
            // )( 不合法 +(  ((
        if (i < expr.size() - 1 && expr[i] == ')' && (!isOperator(expr[i+1]) && expr[i+1] != ')'))
            return false;
        if (i < expr.size() - 1 && (expr[i] == 'e' || expr[i] == 'E') && (!isdigit(expr[i+1]) &&
            expr[i+1] != '-')) return false; // 1e+ 不合法 1e-1 合法
    }
    return openParenCount == 0;
}
```

StringToNumber 函数的实现如下：

```
double StringToNumber(const std::string& expr, size_t& i) {
    std::string num = "";
    int dotCount = 0; // 用于记录小数点数量
    int eCount = 0;
    if (expr[i] == '-') {
            num += expr[i++];
```

```
    }
    while (i < expr.size() && (isdigit(expr[i]) || expr[i] == '.' || expr[i] == 'e' || expr[i] == 'E
        ')
            || (expr[i] == '-' && (expr[i-1] == 'e' || expr[i-1] == 'E'))) {
        if (expr[i] == '.') {
            if (++dotCount > 1) {
                throw std::invalid_argument("Invalid number format: more than one decimal point.");
            }
        }
        if (expr[i] == 'e' || expr[i] == 'E') {
            if (++eCount > 1) {
                throw std::invalid_argument("Invalid number format: more than one exponent.");
            }
        }
        num += expr[i++];
    }
    try {
        return std::stod(num); // 转换为 double
    } catch (const std::exception& e) {
        throw std::invalid_argument("Invalid number format");
    }
    }
```

applyOperator 的实现如下：

```
double applyOperator(double a, double b, char op) {
    switch (op) {
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/':
            if (b == 0) throw runtime_error("division by zero");
            return a / b;
        default: return 0;
    }
}
```

2.5.2 计算器实现代码

计算器的核心代码实现如下：

```
double evaluateExpression(const string& expr) {
    stack<double> values; // 存储操作数
    stack<char> operators; // 存储运算符

    size_t i = 0;
```

```
    while (i < expr.size()) {
        if (isspace(expr[i])) {
            i++;
            continue;
        }

        if (isNumber(expr[i])) {
            // 解析数字
            try {
            double value_ = StringToNumber(expr, i);
            values.push(value_);
            } catch (const std::invalid_argument& e) {
                throw std::invalid_argument("Invalid number format");
            }
        }
        else if (expr[i] == '-' && (i == 0 || expr[i-1] == '(')){
            try{
            double value_ = StringToNumber(expr, i);
            values.push(value_);
            } catch (const std::invalid_argument& e) {
                throw std::invalid_argument("Invalid number format");
            }
        }
        else if(i > 0 && expr[i] == '-' && isOperator(expr[i-1])){
            // 处理负数
            try {
            double value_ = StringToNumber(expr, i);
            values.push(value_);
            } catch (const std::invalid_argument& e) {
                throw std::invalid_argument("Invalid number format");
            }
        }
        else if (isLeftParenthesis(expr[i])) {
            operators.push(expr[i]);
            i++;
        }
        else if (isRightParenthesis(expr[i])) {
            // 处理右括号，计算括号内的表达式
            while (!operators.empty() && operators.top() != '(') {
                double b = values.top(); values.pop();
                double a = values.top(); values.pop();
                char op = operators.top(); operators.pop();
                values.push(applyOperator(a, b, op));
            }
```

```
            if (!operators.empty()) operators.pop(); // 弹出左括号
            i++;
        }
        else if (isOperator(expr[i])) {
            // 处理运算符
            while (!operators.empty() && precedence(operators.top()) >= precedence(expr[i])) {
                double b = values.top(); values.pop();
                double a = values.top(); values.pop();
                char op = operators.top(); operators.pop();
                values.push(applyOperator(a, b, op));
            }
            operators.push(expr[i]);
            i++;
        }
        else {
            throw runtime_error("Invalid character in the expression.");
        }
    }

    // 处理剩余的运算符
    while (!operators.empty()) {
        double b = values.top(); values.pop();
        double a = values.top(); values.pop();
        char op = operators.top(); operators.pop();
        values.push(applyOperator(a, b, op));
    }

    return values.top();
}
```

3 测试与结果

3.1 测试用例

为了测试结果真实可靠，我们使用了大量的测试用例，以求保证我们的计算器能在各式各样的环境下成功运行。以下是一些测试用例：

```
    //四则运算
    "1+2", "2-3", "3*4", "6/3", "6/0", "0/0",
    //四则运算复合
    "1+2*3", "1*2+3", "1*2/3", "1/2*3", "1/2+3", "1+2/3", "1-2*3", "1*2-3",
    "1-2/3", "1/2-3", "1+2-3", "1-2+3", "1+2+3", "1-2-3", "1*2*3", "1/2/3",
    //负数加减与运算符连用
    "1+-2", "1--2", "1*-2", "1/-2", "1**2", "1//2", "1++2",
    "-1+2", "-1-2", "-1*2", "-1/2", "-1**2", "-1//2", "-1++2",
```

```
    "1+++2", "1---2", "1***2", "1///2", "1++++2", "1----2", "1****2", "1////2",
    //负数的四则运算复合
    "-1+-2*3", "-1*-2+3", "-1*-2/3", "-1/-2*3", "-1/-2+3", "-1+-2/3", "-1--2*3", "-1*-2-3",
    "-1--2/3", "-1/-2-3", "-1+-2-3", "-1--2+3", "-1+-2+3", "-1--2-3", "-1*-2*3", "-1/-2/3",
    "1+-2*3", "1*-2+3", "1*-2/3", "1/-2*3", "1/-2+3", "1+-2/3", "1--2*3", "1*-2-3",
    "1--2/3", "1/-2-3", "1+-2-3", "1--2+3", "1+-2+3", "1--2-3", "1*-2*3", "1/-2/3",
    //小数参与运算
    "1.2+3.4", "1.2-3.4", "1.2*3.4", "1.2/3.4", "1.2/0.0",
    "1.5+2.5", "1.5-2.5", "1.5*2.5", "1.5/2.5", "1.0+0.2",
    "1.1.1", "1.1.1+2.2.2", "1.1.1-2.2.2", "1.1.1*2.2.2", "1.1.1/2.2.2",
    ".1+2.5", ".1-2.5", ".1*2.5", ".1/2.5", ".1/0.0",
    //科学计数法
    "2e-1", "1.5e2+3", "-3.2e+1", "-3.2e*1", "-3.2e/1", "-3.2e-1",
    "1.5e+2", "1.5e-2", "-1.5e2+2",
    "1.5e2*-2", "1.5e-2/2", "1.5e0",
    "1.2e-3/0.0", "-1.2e-3+2", "-1.2e-3-2",
    "1.2ee1", "1.2e1e1", "1.1e1.2", "e1","1-2e-2+3",
    "1-2e+3","1-2e-3","1-2e3"
    //括号
    "(1+2)", "(1+2)*3", "1+(2*3)", "1+2*3", "1+2*(3-4)", "(1+2)*(3-4)", "(1+2)*(3-4)/2", "(1+2)
        *(3-4)/2+5",
    "((1+2)*(3-4))/2+5", "((1+2)*(3-4))/2+5*6", "((1+2)*(3-4))/2+5*6/7", "((1+2)*(3-4))/2+5*6/7+8",
    "1+(2*(3-4)", "(1+(2*(3-4))", "1+(2*3(", "1+)", "(1+2))", "()",
    "(1+3)(4-2)", "(-1)", "(+1)", "(+)", "(1+)", ")(", "()()",
    "(1e2)*(2e3)", "(1e2)+(2e3)", "(1e2)-(2e3)", "(1e2)/(2e3)",
    "(1ee)*(2e3)", "(1e2)*(2ee3)", "((1e2)*(2e3))", "((1e2)*(2e3)))))",
    //其他非法
    "", "1+", "+1", "1+*2", "1/0/0", "1..2+3", "1.2.3+4", "1e+2+3", "1ee2+3", "(1+2)3",
    "(1-)", "(1*)", "(1/)", "((()))", "((((1+2))))",
    "1    +     1 ", "1412edwqf2342ewd",
    "11411 edews", "+121+12123", "-1+1231", "*1231+123", "/1231+123", "--1+2"
```

这些用例涵盖了各式各样常见或不常见的使用案例，包括正负数、小数、科学计数法、括号嵌套、非法字符等，可以全面测试表达式的解析和计算功能。我们按如下规则对结果进行输出：

- 正确结果：输出”= 结果”
- 非法输入：输出” ILLGAL”

3.2 运行结果

```
1+2=3
2-3=-1
3*4=12
6/3=2
6/0 ILLGAL
```

```
0/0 ILLGAL
1+2*3=7
1*2+3=5
1*2/3=0.666667
1/2*3=1.5
1/2+3=3.5
1+2/3=1.66667
1-2*3=-5
1*2-3=-1
1-2/3=0.333333
1/2-3=-2.5
1+2-3=0.00000
1-2+3=2
1+2+3=6
1-2-3=-4
1*2*3=6
1/2/3=0.166667
1+-2=-1
1--2=3
1*-2=-2
1/-2=-0.5
1**2 ILLGAL
1//2 ILLGAL
1++2 ILLGAL
-1+2=1
-1-2=-3
-1*2=-2
-1/2=-0.5
-1**2 ILLGAL
-1//2 ILLGAL
-1++2 ILLGAL
1+++2 ILLGAL
1---2 ILLGAL
1***2 ILLGAL
1///2 ILLGAL
1++++2 ILLGAL
1----2 ILLGAL
1****2 ILLGAL
1////2 ILLGAL
-1+-2*3=-7
-1*-2+3=5
-1*-2/3=0.666667
-1/-2*3=1.5
-1/-2+3=3.5
```

```
-1+-2/3=-1.66667
-1--2*3=5
-1*-2-3=-1
-1--2/3=-0.333333
-1/-2-3=-2.5
-1+-2-3=-6
-1--2+3=4
-1+-2+3=0.00000
-1--2-3=-2
-1*-2*3=6
-1/-2/3=0.166667
1+-2*3=-5
1*-2+3=1
1*-2/3=-0.666667
1/-2*3=-1.5
1/-2+3=2.5
1+-2/3=0.333333
1--2*3=7
1*-2-3=-5
1--2/3=1.66667
1/-2-3=-3.5
1+-2-3=-4
1--2+3=6
1+-2+3=2
1--2-3=0.00000
1*-2*3=-6
1/-2/3=-0.166667
1.2+3.4=4.6
1.2-3.4=-2.2
1.2*3.4=4.08
1.2/3.4=0.352941
1.2/0.0 ILLGAL
1.5+2.5=4
1.5-2.5=-1
1.5*2.5=3.75
1.5/2.5=0.6
1.0+0.2=1.2
1.1.1 ILLGAL
1.1.1+2.2.2 ILLGAL
1.1.1-2.2.2 ILLGAL
1.1.1*2.2.2 ILLGAL
1.1.1/2.2.2 ILLGAL
.1+2.5=2.6
.1-2.5=-2.4
```

```
.1*2.5=0.25
.1/2.5=0.04
.1/0.0 ILLGAL
2e-1=0.2
1.5e2+3=153
-3.2e+1 ILLGAL
-3.2e*1 ILLGAL
-3.2e/1 ILLGAL
-3.2e-1=-0.32
1.5e+2 ILLGAL
1.5e-2=0.015
-1.5e2+2=-148
1.5e2*-2=-300
1.5e-2/2=0.0075
1.5e0=1.5
1.2e-3/0.0 ILLGAL
-1.2e-3+2=1.9988
-1.2e-3-2=-2.0012
1.2ee1 ILLGAL
1.2e1e1 ILLGAL
1.1e1.2 ILLGAL
e1 ILLGAL
1-2e-2+3=3.98
1-2e+3 ILLGAL
1-2e-3=0.998
1-2e3(1+2) ILLGAL
(1+2)*3=9
1+(2*3)=7
1+2*3=7
1+2*(3-4)=-1
(1+2)*(3-4)=-3
(1+2)*(3-4)/2=-1.5
(1+2)*(3-4)/2+5=3.5
((1+2)*(3-4))/2+5=3.5
((1+2)*(3-4))/2+5*6=28.5
((1+2)*(3-4))/2+5*6/7=2.78571
((1+2)*(3-4))/2+5*6/7+8=10.7857
1+(2*(3-4) ILLGAL
(1+(2*(3-4)) ILLGAL
1+(2*3( ILLGAL
1+) ILLGAL
(1+2)) ILLGAL
() ILLGAL
(1+3)(4-2) ILLGAL
```

```
(-1)=-1
(+1) ILLGAL
(+) ILLGAL
(1+) ILLGAL
)( ILLGAL
()() ILLGAL
(1e2)*(2e3)=200000
(1e2)+(2e3)=2100
(1e2)-(2e3)=-1900
(1e2)/(2e3)=0.05
(1ee)*(2e3) ILLGAL
(1e2)*(2ee3) ILLGAL
((1e2)*(2e3))=200000
((1e2)*(2e3))))) ILLGAL
 ILLGAL
1+ ILLGAL
+1 ILLGAL
1+*2 ILLGAL
1/0/0 ILLGAL
1..2+3 ILLGAL
1.2.3+4 ILLGAL
1e+2+3 ILLGAL
1ee2+3 ILLGAL
(1+2)3 ILLGAL
(1-) ILLGAL
(1*) ILLGAL
(1/) ILLGAL
((())) ILLGAL
((((1+2))))=3
1    +     1  ILLGAL
1412edwqf2342ewd ILLGAL
11411 edews ILLGAL
+121+12123 ILLGAL
-1+1231=1230
*1231+123 ILLGAL
/1231+123 ILLGAL
--1+2 ILLGAL
```

4 特殊说明

考虑到本计算器用途有限，我们仅支持五位有限小数运算，这对于实际用途而言已经足够。同时，由于 `double` 数据类型的限制，本计算器无法正确处理超出其表示范围（大约为 9e15）的数值。但是考虑到本计算器并非用于科学计算的科学计算器，这样的精度依然是可以接受的。

5 结论

我们所实现的计算器满足课程作业的全部要求，包括：

- 支持多重括号和四则运算。
- 支持有限位小数运算
- 识别非法的表达式，如括号不匹配、运算符连续使用、表达式以运算符开头或结尾以及除数是 0 等。
- 考虑了负数，比如：1+-2.1 是合法的，但 1++2.1 是非法的。
- 考虑了科学计数法，比如：-1+2e2 是合法的。

同时，大量测试结果还证明了本计算器针对各种无效输入的错误检测和处理具有鲁棒性。