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。
- 左括号的优先级最低，用于标记优先计算范围。

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，尤其是支持小数与科学计数法时。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"`）。
   - 科学计数法（例如 `"1.23e4"`）。
2. **内置异常处理**：`std::stod` 在解析过程中会自动检测非法格式，若输入字符串无法解析为有效数字，将抛出 `std::invalid_argument` 异常。对于超出范围的数字，会抛出 `std::out_of_range` 异常。
3. **高效性**：作为标准库函数，`std::stod` 在性能上经过优化，适用于大多数应用场景。
4. **附加功能**：`std::stod` 可以通过一个指针参数返回解析结束的位置，从而方便进一步处理表达式中剩余部分的字符。

2.4 代码实现

3 测试与结果

3.1 测试用例

- 测试表达式1：$3 + 5 \times 2 - (6/3)$
- 测试表达式2：$(2 + 3) \times (5 - 2)$
- 测试表达式3：$4 \times (6 + 2)/8$

3.2 运行结果

- 表达式1结果：10
- 表达式2结果：15
- 表达式3结果：4