# 1 expression_evaluator.h 的设计思路

我自定义了一个类 arithmetic，用于接收外界输入的字符串，完成表达式的中缀、后缀转化，错误检验，表达式计算求值，以及最后输出的功能。

先阐释分析类的 private 部分成员。

## 1.1 成员变量

1. stack<char> operators 存放运算符，用于中缀转后缀时的运算符优先级比较与输出。
2. string postfix_expression 存放转换后的后缀表达式。
3. stack<double> numbers 存放操作数，用于后缀表达式的计算求值。
4. indicator 用于标记表达式是否合法，若合法则为 true，否则为 false。如果表达式不合法，将直接返回，停止该表达式的计算，进入测试程序中下一个表达式的计算。

## 1.2 `priority` 函数

预备函数，用于定义运算符的优先级，在中后缀转化时可以直接调用。同时兼具判断未定义字符的功能。

## 1.3 `right_bracket` 函数

预备函数，用于集中处理右括号的情况。因为右括号处理的逻辑都是一致的，都是不断弹出运算符，连接到后缀表达式，直到遇见对应左括号为止（左括号不入序列），故可集成处理。同时，兼具判断括号是否匹配的功能，若运算符全部输出，仍未遇到匹配的左括号则表达式不合法。

## 1.4 `transform` 函数

本函数是整个类的**核心函数**之一，用于将中缀表达式转化为后缀表达式。因为需要判断表达式的合法性，采用比较朴实的方法，定义了四个指示变量，用于记录表达式的不同部分和运算阶段。

1. `flag` 用于指示运算符的连续出现个数。为了应对开头是运算符的情况，将初值设为 1。正常情况下（负号另行考虑），flag 为 0-1 变量，若 flag 为 2 则运算符连续出现，表达式不合法。
2. `digit` 用于小数点前位数计数，检验是否符合科学计数法规范，与下一个 `point` 变量配合使用。
3. `point` 用于标记是否进入小数部分，检验是否符合科学计数法规范。
4. `notion` 用于标记是否进入科学计数法的幂次部分。

接下来，根据输入的中缀表达式逐个字符处理，根据字符的不同情况分类讨论。

### 1.4.1 数字情形

数字，小数点，以及科学计数法 e 实际都为数字部分，整体考虑，除了非法判断分别处理。都直接加入后缀表达式中，同时将 flag 置零，因为只有数字能打断运算符的连续出现。各指示变量按照实际情况进行维护。

如果科学计数法的 e 前的整数位数超过一位，或幂次不为整数，则表达式不合法。都通过 notion、point、digit 变量配合判断。

#### 1.4.2 运算符'+"-"*' 情形

根据优先级统一处理，维护各指示变量。运算符及括号标志数字结束，如果前面的数字为科学计数法，在末尾加'E' 作为标记，便于后续计算。若运算符连续出现，直接退出报错。同时为了分隔各操作数，在运算符前加入空格，便于后续计算。

#### 1.4.3 '-' 情形

负号的处理比较特殊，可能是负号或减号，通过 flag 变量的值进行区分。

若 flag 为 1，根据我的程序定义逻辑为负号，直接加入后缀表达式中。若 flag 为 0，则为减号，按运算符处理。

若 flag 大于 1，说明运算符连续出现，不合法，直接退出报错。

#### 1.4.4 括号情形

左括号推入栈中，维护各指示变量。右括号调用 `right_bracket` 函数处理。

#### 1.4.5 未定义字符情形

直接退出报错。

### 1.5 `calculate` 函数

本函数是整个类的另一个**核心函数**，用于计算后缀表达式的值。表达式本身的格式合法与否已由 `transform` 函数完成判断，本函数无需考虑。

根据后缀表达式的特点，从左到右逐个扫描，遇到数字直接入栈，遇到运算符弹出两个数字进行运算，结果入栈。0-1 指示变量 flag 用于表示数字即将开始。根据 `transform` 函数的定义，后缀表达式中的空格只出现在数字（包括负数）之前。为保持逻辑一致，初值设为 1。如果遇到空格或 e，flag 置 1，表示数字即将开始。

#### 1.5.1 数字情形

Line 214-229 的 if-else 语句，整体用于处理数字（包括负数）的情况，并转化为浮点数。直接将负号、数字、小数点加入字符串 num 中，遇到其它符号时，表示数字部分结束，将字符串转化为浮点数，入栈。

**注意：**负号情况下，flag 不可置零！必须出现数字才能置零。因为该组判断与下一组 if-else 的运算判断独立，置零后会满足下一组 if-else 条件，引发段错误！

#### 1.5.2 运算符情形

Line 231-250 的 if-else 语句，整体用于处理运算符的情况。

此处将 `transform` 中加入的指示符'E' 也当作运算符处理，用于计算科学计数法的结果（因为事实上，也是 e 前后两个操作数完成的计算）。数字栈中弹出两个操作数，基本运算直接调用 `fundamental` 函数，同时对除数为零的情况进行判断和报错。

### 1.6 两个打印函数

`print` 用于打印后缀表达式，主要用于编程过程中检查中间步骤的正确性。

`print_result` 函数打印计算结果，如果 indicator 指示报错则不打印。

public 部分：

### 1.7 `calculation` 函数

集成了接收外界输入，调用私有函数完成计算，输出结果的功能。同时在各步骤过程中都及时判断是否出错，若出错则暂停该表达式求值，进入测试程序下一个表达式。

## 2 测试函数

测试了以下几种类型的表达式，详见 `test.cpp` 文件。

1. 基本的加、减、乘、除运算以及括号的处理。
2. 检验中括号的处理。
3. 检验大括号。
4. 括号不配对的情况。
5. 更多括号不配对。
6. 检验除数为零的情况。
7. 表达式中包含未定义字符的情况。
8. 检验运算符连续出现。
9. 检验运算符在表达式开头的情况。
10. 表达式中间包含负数的情况。
11. 开头包含负数的情况。
12. 开头包含科学计数法。
13. 科学计数法，幂次为负数。
14. 检验科学计数法不合规范的情况。
15. 包含小数的运算。
16. 检验科学计数法幂次非法的情况。

# 3 运行结果

如下图所示，测试程序运行结果正确，所有测试用例均符合理论预期。

```
summer_hare@localhost:/mnt/d/浙大本科学习/大二上/数据结构与算法分析/My Project/arithmetic
$ ./test
expression:12+(11+25)*3+41/5
12  11 25+ 3*+ 41 5/+
128.2
expression:[12+(11+25)*3]/2+41/5
 12  11 25+ 3*+ 2/ 41 5/+
68.2
expression:{[12+(11+25)*3]/2+41/5}*3
  12  11 25+ 3*+ 2/ 41 5/+ 3*
204.6
expression:12+(11+25]*3+41/5
ILLEGAL
expression:({12+[11+25)*3]/2+41/5}*3
ILLEGAL
expression:(11+25)/0+41/5
 11 25+ 0/ 41 5/+
ILLEGAL
expression:(11+25)*$$+41/&
ILLEGAL
expression:(11++25)*15+41/2
ILLEGAL
```

(a) a

```
expression:+25*15+41/2
ILLEGAL
expression:(1+-25)*15+41/2
 1 -25+ 15* 41 2/+
-339.5
expression:-25*15+41/2
-25 15* 41 2/+
-354.5
expression:2e3*15+41/2
2e3E 15* 41 2/+
30020.5
expression:2e-4*150+41/2
2e-4E 150* 41 2/+
20.53
expression:26.1e5*15+41/2
ILLEGAL
expression:2.6*10.6+40.5/2.4
2.6 10.6* 40.5 2.4/+
44.435
expression:2.6e2.6*10.6+40.5/2.4
ILLEGAL
```

(b) b

图 1: 测试程序运行结果

## 完结撒花!