1 expression_evaluator 的功能及实现

请注意测试时请不要出现空格，同时一味的跳过空格是不合理的，如 **1 2** 在平常看来如果是为了便于书写，而将运算符和数字之间预留空隙，这样子实际上很难判断其为 **12** 还是实数域上的默认的标准乘法，因此我们规定在输入时不要输入空格。

首先我的 **expression_evaluator** 实现了以下几个所要求的基本功能:

- 支持多重括号和四则运算。
- 支持有限位小数运算，但可以不考虑负数作为输入。
- 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。
- 能处理含有加 (+)、减 (-)、乘 (*)、除 (/) 和括号 (()) 的中缀表达式。

再者额外的一些功能：

- 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。(注：**E** 也是可以的下同)
- 考虑了科学计数法正负号，比如：**-1+2e+2,-1+2e-2** 是合法的。(注：**2e** 是非法的)
- 考虑了 { }，[ ] 匹配问题，但像 { (}) 是非法的。
- 考虑了各种不同的非法情况，并告诉用户何种错误情况。

1.1 前言

虽然王老师所讲为将中缀表达式转化为后缀表达式，同时利用 **stack** 去储存运算，但我觉得在实现上可能并不如 **List** 来的直观，这也是我如何去实现以及为什么使用 **List** 的原因。不过遗憾的是，在书写报告时我才发现这种算法在实现类似 **(2+3)*(1+2)** 类型时，即多非包含括号时会出现一些问题，时间匆忙，因此最后我还是以栈来完成了这一部分的缺漏，为了方便代码我也选择保留，也是希望以后有机会能对其再度完善。

1.2 思路简介

首先我初步的想法是去递归运算，如何递归呢？我想到在没有括号的情况下，问题将会被大大简化，从而能被轻松解决。因此我设想一个 **List**, 能够在遇到从左遇到第一个左括号时，再去寻找一个与之匹配的右括号，将括号外部分复制入 **List**，同时预留一个 **subval node**, 括号内算式的存入 **sublist**, 递归下去，最终必将不含括号，然后将 **childlist** 运算的值后返回 **parentList** 的 **subval node**, 从而 **List** 中不存在括号，可以运算，逐次返回 **childList** 的 **val**, 最终至 **List**。**return List** 的 **val**, 完成运算。

而在 **stack** 实现中最复杂的部分只需要两个函数，一个函数用来转化，一个用来计算。其中计算是较为方便实现的，如果已经实现了转化为后缀表达式，这时只需要按照已经完成的后缀表达式，遇到数字便 **push stack**, 遇到运算符便 **poptop**, 因为当初实现后缀表达式时，如果两个运算符连在一起，前者的运算优先级总是高于后者的，因此在实现时我们需要对加减法，与乘除法的优先级进行定义，遇到左括号时先压入栈，再压入运算符，而遇到压入的运算符优先级低于或等于时，**pop** 栈顶运算符，若是右括号时则不输出只 **pop** 至左括号，中间的运算符均 **pop** 且输出但括号无需输出。最后转化为后缀表达式了之后，将运算数字经过 **stod** 转化，依次压入栈内直至遇到第一个运算符，这是 **pop** 栈顶两个元素进行相应运算，并再将运算结果 **push top**。最后直至运算直至没有运算符且只有一个数，**pop** 并 **return** 即可得到答案。

1.3 主要成员函数

成员函数：

vector<string> List::transforinto_postfix();

void judge();
int acquire_prefrence(char op);
double List::operation(double &a,char opera,double &b)
double List::calculate_postfix(const vector<string> data)

2 测试及运行结果

2.1 针对错误输入的测试

如下：
- **1$1+5** 非法字符
- **ILLEGAL CHARACTER $**
- **1++2**6** 非法运算
- **ILLEGAL OPERATION**
- **((2+3+5)*2** 括号缺少
- **ILLEGAL PARENTHESIS MATCHING**
- **{5+2*(3+4}+2)** 括号匹配问题
- **ILLEGAL PARENTHESIS MATCHING**
- **5+2*(3+2*)** 子列缺少运算后数字
- **ILLEGAL OPERATION**
- **3+5*** 缺少运算后数字 - **ILLEGAL OPERATION**
- **5+2*3+(1*{5+2*(3+4}+2)+2+1)** 子列括号匹配问题
- **ILLEGAL PARENTHESIS MATCHING**
- **3+5.2.1**(太多点)
- **ILLEGAL OPERATION**
- **3e2e3**(太多 e)
- **ILLEGAL OPERATION**
- **2/(2-2)**(除 0)
- **A divisor of 0 is ILLEGAL**
- **1+2(*2+3)**(缺少运算前数字)
- **ILLEGAL OPERATION**
- ***1+2**
- **ILLEGAL OPERATION**

2.2 针对运算功能的测试

如下：
- **1+-2.1** (负数运算)
- 答案是：**-1.1** correct
- **-1+2e2** (科学计数法)
- 答案是：**199** correct
- **123456789*987654321**(大数运算)
- 答案是：**1.21933e+17** correct
- **0.11+2e-2+2E+2**(科学计数法再检验及小数)

- 答案是：**200.13** correct
- **5+2*(3+(1*5+2*(3+4)+2)+2+1)**(复杂运算)
- 答案是：**59** correct
- **((5+3)*2)/(4+(3-1))** (除法且为 List 难以消去 bug 类)
- 答案是：**2.66667** correct
- **{5+2}*(3+4+2/[3+1])** (带有多种括号的匹配问题)
- 答案是：**52.5** correct

3 结言

通过我的报告和代码，我认为已经完成了尽可能全面的四则运算。总的来讲，我认为这次实现四则运算器是一个不小的挑战。但无论如何，最终还是完成了，并通过实现运算器复习了之前所学的知识，也激起了对数据结构还有算法方面的兴趣，也是一次不错的跨学科尝试！