1 expression_evaluator 的功能及实现

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。
- 考虑了科学计数法正负号，比如：**-1+2e+2,-1+2e-2** 是合法的。(注：为方便起见我们认为 **2e** 是 **2**，如：**2e*2** 就是 **2*2** 为 **4**)
- 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。

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，倒着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。

1.3 主要成员函数与重要状态量

成员函数：

```
void judge(string str)
void behind_copy(iter subend , iter end , Node *&subval);
void front_copy(iter begin , iter subbegin , Node *&subval);
List *sublist = nullptr;
void copy(iter begin , iter subbegin, iter subend , iter end);
void copy(iter begin , iter end);
string substr(string str,iter begin,iter end);
void transfor();
bool judge_element(iter it);
bool judge_operator(iter it);
void readin(string str)
void run();
void Inputexpression();
double operation(double &a,char opera,double &b)
```

double calculate()
void myjudge()(这个函数是我用来测试输出结果的函数，虽然可能单独提及一个函数显得有些突兀，而且理应我应在完成作业时将其删除，不过这个函数在完成我的作业上起到了决定性作用，还是选择将其保留并特此一表。)
状态量：
int which_kind_of_copy = 0;
int is_sublist_copy = 0;

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*)** 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**

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+2e*2**(科学计数法再检验)
- 答案是：**204.13** **correct**

3 结果及分析

我运行了数次之后，取平均值之后并保留三位小数得到：

	my heapsort time	std::sort_heap time
random sequence	0.100s	0.060s
ordered sequence	0.046s	0.028s
reverse sequence	0.045s	0.057s
repetitive sequence	0.082s	0.056s

不难发现，除了 **reverse sequence,my sortheap** 都比 **std** 效率低 (因为 **myheapsort** 是最小堆，最后实现的是逆序，于是我已经将 **mysortheap** 结果对调，以保证 **worst** 与 **best** 情况对应) 我猜测，**std** 存在一些优化策略, 比如随机化: 探测若干项是否满足序关系，如果的确如此，那么正如 **ordered** 所显示的那样，时间会大大缩短。相反如果不满足序关系占大多数，**std** 可能会先对此进行一个随机化操作，此时时间复杂度是 $O(1)$ 不会对整体复杂度产生太大影响。因此 **std** 的 **random,repetitive,reverse** 的时间近乎于相等，而我的程序却没有对此的优化，导致不同程序之间运算时间相差过大. 除了 **best** 情况剩下的均比 **std** 要慢。通过查询资料我了解到，**std** 在运算时可能有并行化向量化减少运算时间，而且同时优化内存访问模式来提高缓存命中率，这些可能也是 **std** 效率比 **myheapsort** 快的原因。