面向对象设计与构造第三次作业

第一部分：训练目标

通过对数学意义上的表达式结构进行建模，完成多项式的括号展开与函数调用、化简，进一步体会层次化设计的思想的应用和工程实现。

第二部分：预备知识

- 1、Java 基础语法与基本容器的使用。
- 2、扩展 BNF 描述的形式化表述。
- 3、正则表达式、递归下降或其他解析方法。

第三部分：题目描述

本次作业中需要完成的任务为：读入**一系列自定义函数的定义**以及一个包含幂函数、三角函数、自定义函数调用、求导算子的**表达式**，输出**恒等变形展开所有括号后**的表达式。

在本次作业中，**展开所有括号**的定义是：对原输入表达式 E*E* 做**恒等变形**，得到新表达式 E'*E*'。其中，E'*E*' 中不再含有自定义函数，不再含有求导算子，且只包含**必要的括号**（必要括号的定义见**公测说明-正确性判定**）。

在本次作业中，**自定义函数** 指自定义递推函数和自定义普通函数。

第四部分:迭代内容概览

在第二次作业基础上，本次迭代作业**增加**了以下几点：

- 本次作业支持求导操作,新增求导算子 。
  - 根据**第五部分形式化表述**，求导因子可以出现在很多位置，包括函数调用实参，三角函数内部等，注意考虑周全。
  - 为了限制难度，在输入中，求导算子不会在自定义函数中出现，具体见**第六部分-数据限制。**
- 本次作业函数表达式中支持调用其他“已定义的”自定义普通函数（保证不会出现递归调用，具体见**第六部分-数据限制**）。
- 本周实验会着重指导求导将如何层次化实现。

第五部分：基本概念

一、基本概念的声明

- **带符号整数 支持前导 0**的**十**进制带符号整数（若为正数，则正号可以省略），无进制标识。如：`+02`、`-16`、`19260817`等。
- **因子**
  - **变量因子**
  - **幂函数**

- **一般形式** 由自变量 x，指数符号 `^` 和指数组成，指数为一个**非负**带符号整数，如：`x ^ +2`,`x ^ 02`,`x ^ 2` 。
- **省略形式** 当指数为 1 的时候，可以省略指数符号 `^` 和指数，如：`x` 。
- **三角函数**
  - **一般形式** 类似于幂函数，由 `sin(<因子>)` 或 `cos(<因子>)` 、指数符号 `^` 和指数组成，其中：
  - 指数为符号不是 `-` 的整数，如：`sin(x) ^ +2` 。
  - **省略形式** 当指数为 1 的时候，可以采用省略形式，省略指数符号 `^` 和指数部分，如：`sin(x)` 。
  - 本指导书范围内的"三角函数"**仅包含** `sin` **和** `cos` 。
- **自定义递推函数**
  - 自定义递推函数的**定义**形如

```
f{n}(x, y) = 递推表达式
f{0}(x, y) = 函数表达式
f{1}(x, y) = 函数表达式
```

三者顺序任意，以换行分隔。递推表达式和函数表达式的定义见"形式化表述"部分。

定义中默认 n>1n>1。

例如

```
f{0}(y) = y
f{1}(y) = 1
f{n}(y) = 1*f{n-1}(sin(y)) - 4*f{n-2}(y^2) + 1
```

```
f{0}(x, y) = x - y
f{n}(x, y) = 0*f{n-1}(x, y) + 35*f{n-2}(x, y^2)
f{1}(x, y) = x^3 + y
```

  - `f` 是递推函数的**函数名**。在本次作业中，保证函数名只使用 `f`，且每次只有1个自定义递推函数。 `n` 、 `0` 、 `1` 是递推函数的**序号**。
- `x`、`y` 是递推函数的形参。在本次作业中，**形参个数为 1~2 个**。形参**只使用x，y**，且同一函数定义中不会出现重复使用的形参。对一个自定义递推函数的定义来说，`f{n}`、`f{n-1}`、`f{n-2}`、…、`f{1}`、`f{0}` 等一系列函数的形参统一，不会出现同一系列中函数形参不同的情况。
  - 递推表达式是一个关于形参的表达式，**保证其中** `f{n-1}` **和** `f{n-2}` **各被调用且只被调用 1 次**，并且调用前需要 **和一个常数因子相乘。（新增）允许调用自定义普通函数。**函数表达式是一个关于形参的表达式。二者的一般形式见**形式化定义**。
  - 自定义递推函数的**调用**形如 `f{常数因子}(因子, 因子)`，比如 `f{3}(x^2)`，`f{5}(-1, sin(x^2))`。
  - 大括号中的 `常数因子` 为函数调用时的**序号**，你需要根据自定义递推函数的定义找到序号对应的函数，才能计算。保证0≤序号≤5。
  - 小括号中的 `因子` 为函数调用时的**实参**，包含任意一种因子。
- **自定义普通函数（新增）**

- 自定义普通函数的**定义**形如 `g(x, y) = 函数表达式` ，比如 `g(y) = y^2`，`g(x, y) = sin(x)*cos(y^2)`，`h(x, y) = x + y` 。
- `g`、`h` 是函数的**函数名**。在本次作业中，保证函数名**只使用 `g`，`h`，且不出现同名函数的重复定义**（因此每次最多只有 2 个自定义普通函数）。更具体的约束信息请看第六部分中的数据限制部分。
- `x`、`y` 为函数的**形参**。在本次作业中，**形参个数为 1~2 个**。形参**只使用 `x`，`y`**，且同一函数定义中不会出现重复使用的形参。
- 函数表达式为一个关于形参的表达式。为了限制难度，我们规定**求导因子不会出现在函数表达式中**。函数表达式的一般形式参见**形式化定义**。
- 函数表达式中允许调用自定义普通函数，但保证不出现递归调用的情况，如

```
g(x) = h(x) + 6
h(x) = g(x) * sin(x)
```

不合法。

不允许先调用函数，再进行声明，也不允许调用未声明的函数，如

```
g(x) = h(x, x) + 666
h(x, y) = x*sin(y)
```

不合法。

- 自定义普通函数的**调用**形如 `g(因子, 因子)` ，比如 `g(x^2)`，`g(sin(x^2), cos(x))`，`h(1, 0)` 。
- `因子` 为函数调用时的**实参**，包含任意一种因子。

  - **常数因子** 包含一个带符号整数，如：`233`。
  - **表达式因子** 用一对小括号包裹起来的表达式，可以带指数，且指数为一个**非负**带符号整数，例如 `(x^2 + 2*x + x)^2` 。表达式的定义将在表达式的相关设定中进行详细介绍。
- **项** 由乘法运算符连接若干因子组成，如 `x * 02`。此外，**在第一个因子之前，可以带一个正号或负号**，如 `+ x * 02`、`- +3 * x`。注意，**空串不属于合法的项。**
- **表达式** 由加法和减法运算符连接若干项组成，如：`-1 + x ^ 233 - x ^ 06 +x` 。此外，**在第一项之前，可以带一个正号或者负号，表示第一个项的正负**，如：`- -1 + x ^ 233`、`+ -2 + x ^ 19260817`。注意，**空串不属于合法的表达式。**
- **求导因子（新增）** 由`dx(表达式)`这种算符构成，含义为对表达式中的x变量求导，其中可以出现多次求导的情况，详见求导因子的形式化描述。为了限制难度，我们规定**求导算子不会出现在自定义函数的函数表达式和递推表达式中。**例如
  - `g(x)=x+dx(x)`属于不合文法的自定义函数
- **空白字符** 在本次作业中，空白字符仅包含空格 `<space>`（ascii 值 32）和水平制表符 `\t`（ascii 值 9）。其他的空白字符，均属于非法字符。

  对于空白字符，有以下几点规定：
  - 带符号整数内不允许包含空白字符，注意带符号整数本身的符号与整数之间也不允许包含空白字符。
  - 函数保留字内不允许包含空白字符，即 `sin`，`cos`，`f{序号}`，`dx`关键字内不可以含有空白字符。

二、设定的形式化表述

- 表达式 → 空白项 [加减 空白项] 项 空白项 | 表达式 加减 空白项 项 空白项
- 项 → [加减 空白项] 因子 | 项 空白项 '*' 空白项 因子
- 因子 → 变量因子 | 常数因子 | 表达式因子 | 求导因子
- 变量因子 → 幂函数 | 三角函数 | 函数调用
- 函数调用 → 自定义递推函数**调用** | 自定义普通函数**调用**
- 常数因子 → 带符号的整数
- 表达式因子 → '(' 表达式 ')' [空白项 指数]
- 幂函数 → 自变量 [空白项 指数]
- 自变量 → 'x'
- 三角函数 → 'sin' 空白项 '(' 空白项 因子 空白项 ')' [空白项 指数] | 'cos' 空白项 '(' 空白项 因子 空白项 ')' [空白项 指数]
- 指数 → '^' 空白项 ['+'] 允许前导零的整数 **(注：指数一定不是负数)**
- 带符号的整数 → [加减] 允许前导零的整数
- 允许前导零的整数 → ('0'|'1'|'2'|...|'9'){'0'|'1'|'2'|...|'9'}
- 空白项 → {空白字符}
- 空白字符 → （空格） | `\t`
- 加减 → '+' | '-'
- 换行 → `\n`

自定义递推函数相关(相关限制见“公测数据限制”)

- 自定义递推函数定义 → 定义列表
- 定义列表 → 初始定义 换行 初始定义 换行 递推定义 | 初始定义 换行 递推定义 换行 初始定义 | 递推定义 换行 初始定义 换行 初始定义
- 初始定义 → 'f' '{' 初始序号 '}' 空白项 '(' 空白项 形参自变量 空白项 [',' 空白项 形参自变量 空白项] ')' 空白项 '=' 空白项 函数表达式
- 初始序号 → '0' | '1'
- 递推定义 → 'f{n}' 空白项 '(' 空白项 形参自变量 空白项 [',' 空白项 形参自变量 空白项] ')' 空白项 '=' 空白项 递推表达式
- 序号 → '0'|'1'|'2'|'3'|'4'|'5'
- 形参自变量 → 'x' | 'y'
- 自定义递推函数**调用** → 'f{' 序号 '}' 空白项 '(' 空白项 因子 空白项 [',' 空白项 因子 空白项] ')'
- 自定义递推函数调用n-1 → 'f{n-1}' 空白项 '(' 空白项 因子 空白项 [',' 空白项 因子 空白项] ')' **（注：本次作业中此处的因子不允许出现任何自定义递推函数调用，但允许出现自定义普通函数调用）**
- 自定义递推函数调用n-2 → 'f{n-2}' 空白项 '(' 空白项 因子 空白项 [',' 空白项 因子 空白项] ')' **（注：本次作业中此处的因子不允许出现任何自定义递推函数调用，但允许出现自定义普通函数调用）**
- 递推表达式 → 常数因子 空白项 * 空白项 自定义递推函数调用n-1 空白项 加减 空白项 常数因子 空白项 * 空白项 自定义递推函数调用n-2 [空白项 '+' 空白项 函数表达式]
- 函数表达式 → 表达式（将自变量扩展为形参自变量，且一定不含求导因子）**（注：本次作业中函数表达式不允许出现任何自定义递推函数调用，但允许出现自定义普通函数调用，保证不会出现递归调用的情况）**

自定义普通函数相关(相关限制见“公测数据限制”)

- 自定义普通函数**定义** →→ 自定义普通函数名 空白项 '(' 空白项 形参自变量 空白项 [',' 空白项 形参自变量 空白项] ')' 空白项 '=' 空白项 函数表达式
- 形参自变量 → 'x' | 'y'
- 自定义普通函数**调用** →→ 自定义普通函数名 空白项 '(' 空白项 因子 空白项 [',' 空白项 因子 空白项] ')'
- 自定义普通函数名 → 'g' | 'h'
- 函数表达式 → 表达式（将自变量扩展为形参自变量，且一定不含求导因子）**(注：本次作业中函数表达式不允许出现任何自定义递推函数调用，但允许出现自定义普通函数调用，保证不会出现递归调用的情况)**

求导算子相关(相关限制见“公测数据限制”)

- 求导因子 → 求导算子 空白项 '(' 空白项 表达式 空白项 ')'
- 求导算子 → 'dx'

形式化表述中`{}[]()|`符号的含义已在第一次作业指导书中说明，不再赘述。

式子的具体含义参照其数学含义。

若输入字符串能够由“表达式”推导得出，则输入字符串合法。

除了满足上述形式化表述之外，我们本次作业的输入数据的**额外限制**请参见**第六部分：输入/输出说明 的数据限制部分。**

三、求导公式

本次作业可能用到的求导公式有：

$$\text{I.当}f(x)=c\text{（}c\text{为常数）时，}f'(x)=0$$

$$\text{II.当}f(x)=x^n\text{（}n\neq 0\text{）时，}f'(x)=nx^{n-1}$$

$$\text{III.当}f(x)=cos(x)\text{时，}f'(x)=-sin(x)$$

$$\text{IV.当}f(x)=sin(x)\text{时，}f'(x)=cos(x)$$

$$\text{V.链式法则：}[f(g(x))]'=f'(g(x))g'(x)$$

$$\text{VI.乘法法则：}[f(x)g(x)]'=f'(x)g(x)+f(x)g'(x)$$

第六部分：输入/输出说明

一、公测说明

输入格式

本次作业的输入数据包含若干行：

- 第一行为一个整数 n $(0 \leq n \leq 2)$ ，表示**自定义普通函数定义的个数**。
- 第 2 到第 $n+1$ 行，每行为一行字符串，表示一个自定义普通函数的定义。
- 第 $n+2$ 行为一个整数 $m(0 \leq m \leq 1)$，表示 **自定义递推函数定义的个数**。
- 第 $n+3$ 到第 $n+2+3m$ 行，每行一个字符串，每三行表示一组自定义递推函数的定义。
- 第 $n+3+3m$ 行，一行字符串，表示待展开表达式。

输出格式

输出展开括号之后，不再含有自定义函数，不再含有求导算子，且只包含**必要的括号**的表达式。（必要括号的定义见**公测说明-正确性判定**）。

数据限制

- 输入表达式**一定满足**基本概念部分给出的**形式化描述**。
- **自定义函数定义**满足以下限制：
  - 不会出现重名函数，自定义递推函数的函数名一定为 `f`。
  - 函数表达式**与上次作业不同**，允许调用其他**已定义的**自定义**普通**函数，且不允许出现求导因子，下面是几个不合法的例子。
  - 函数定义时`g(x,y) = g(x,2)+y+1`，出现递归调用，不合法。
  - 函数定义时`h(x,y) = g(x,y)+y,g(x,y)=x+y`，h先定义，g后定义，h在定义时调用了未定义的函数g，不合法。
  - 函数定义时`g(x)=x+dx(x)`，包含了求导因子，不合法
  - 函数形参不能重复出现，即无需考虑 `f(x,x)=x^2+x` 这类情况
  - 函数定义式中出现的变量都必须在形参中有定义
- 对于规则 “指数 → ^ 空白项 ['+'] 允许前导零的整数”，我们本次要求**输入数据的指数不能超过 8。**
- 在表达式化简过程中，如果遇到了 `0^0` 这种情况，默认 `0^0` = 1。
- 为了避免待展开表达式或函数表达式过长。最后一行输入的待展开表达式的**有效长度**至多为 200 个字符，每个自定义普通函数**定义**的**有效长度**至多为 50 个字符，自定义递推函数定义时，每个定义的**有效长度**至多为 75 个字符。其中**有效长度**指的是去除掉所有**空白符**后剩余的字符总数。
- 根据文法可以注意到，整数的范围并不一定在`int`或`long`范围内。

判定模式

本次作业中，对于每个测试点的判定分为**正确性判定**和**性能判定。**其中，正确性判定总分为 85 分，性能判定总分为 15 分，本次作业得分为二者之和。需要特别说明的是，本次作业的强测环节将**适当放宽对程序性能的要求。**

注意：**获得性能分的前提是，在正确性判定环节被判定为正确。**如果被判定为错误，则性能分为0分。

- 正确性判定

- 输出的表达式须符合表达式的**形式化描述**，需要**展开所有括号**且与保持原表达式**恒等**。
- **展开所有括号**的定义：对原输入表达式 E 做**恒等变形**，得到新表达式 E'。其中，E' 中不再含有自定义函数，且只包含**必要的括号**。
  - 三角函数调用时必要的一层括号：`sin()` 与 `cos()`。
  - 三角函数**对应的嵌套因子**为**不带指数的表达式因子**时，该表达式因子两侧必要的一层括号：`sin((x+x))` 与 `cos((x*x))`。（注意是"不带指数"的表达式因子，如果是`sin((x+1)^2)`,这**并不符合必要括号**的定义，你必须将其展开为`sin((x^2+2*x+1))`这种类似的形式才是合法的）
  - 同样，例如 `sin(1)` 与 `sin((1))` 均为展开形式，但 `sin(((1)))` 不是，因为后者除了函数调用和三角嵌套表达式因子的一层括号外，还包括了表达式内嵌套表达式的括号
- 本次作业中对于恒等的定义：设 $f(x)$ 的定义域为 D_1，D_1 包含于 R，$g(x)$ 的定义域为 D_2，D_2 包含于 R，对任意 $x \in D_1 \cap D_2$，$f(x) = g(x)$ 成立。

性能判定：

- 在本次作业中，性能分的唯一评判依据是**输出结果的有效长度**，有效长度的定义在**数据限制部分**已经给出。
- 设某同学给出的**正确答案**的有效长度为 L_p ，目前**所有人**给出的正确答案中有效长度**最小的**为 L_{min} 。

  记 $x = \frac{L_p}{L_{min}}$ ，则该同学**性能分百分比**为：

$$r(x) = 100\% \cdot \begin{cases} 1 & x = 1 \\ -31.8239x^4 + 155.9038x^3 - 279.2180x^2 + 214.0743x - 57.9370 & 1 < x \leq 1.5 \\ 0 & x > 1.5 \end{cases}$$

  举例来说，就是这样：

x	$r\left(x\right)$
1.0	100.0%
1.05	79.9%
1.1	60.5%
1.2	29.0%
1.3	10.9%
1.4	4.5%
1.5	0.0%

  该答案得到的性能分即为 $r(x) \times 15$。

二、互测说明

互测时，你可以通过提交**输入数据**和**预期正确输出**，该组数据会被用来测试同一个互测房间中的其他同学的程序。输入数据必须符合上述的文法规则,并且满足代价函数要求。提交的预期输出只需要包含一行。

- **数据限制**
  - 为了限制难度，输入数据中，求导因子最多只能出现一次。
  - 输入表达式**一定满足**基本概念部分给出的**形式化描述**。
  - 自定义普通函数限制其有效长度至多**30**个字符，其余与公测相同，见上文公测数据限制。
  - 对于规则 "指数 → ^ 空白项 ['+'] 允许前导零的整数"，我们本次要求**输入数据的指数不能超过8**。

- 最终输入表达式的**有效长度**至多为 **40** 个字符。其中输入表达式的**有效长度**指的是输入表达式去除掉所有**空白字符**后剩余的字符总数。（**本条与第二次作业的限制不同**）
- 自定义递推函数定义时，递推定义的有效长度至多**50**个字符，初始定义的有效长度至多**30**个字符。有效长度定义同上
- 除此之外，我们要求输入表达式的代价 `Cost(Expr) <= 3000`，同时要求自定义函数的代价 `Cost(Func) <= 1000`。注意，对于递推函数 `f{n}`，需要保证$0\le n\le 4$，并且对于任意$0\le i\le 4$，`cost(f{i})<=1000`（本次作业的强测环节也将同步放宽对程序性能的要求）。其中表达式和自定义函数代价的计算方法如下：（**本条与公测部分、第二次作业的限制不同**）

代价函数

- `Cost(常数) = max(1, len(常数))`（常数的前导零不算在其长度内）
- `Cost(x) = 1 = Cost(y)`（保证 `y` 仅在自定义函数中出现）
- `Cost(a + b) = Cost(a - b) = Cost(a) + Cost(b)`
- `Cost(a * b) = Cost(a) * Cost(b)`（多项相乘时从左到右计算）
- `Cost(sin(a)) = Cost(cos(a)) = Cost(a) + 1`
- `Cost(a ^ b) =`
- 若a是单变量因子，`Cost(a ^ b) = 1`
- 若a是表达式因子`(c)`，`Cost(a ^ b) = max(Cost(c), 2) ^ max(b,1)`
- 若a是三角函数因子，`Cost(a ^ b) = 2 ^ b + Cost(a)`
- `Cost(+a) = Cost(-a) = Cost(a) + 1`
- `Cost(h) = Cost(h')*2`，h是自定义函数调用，其中 `h'` 是将调用 `h` 的参数**作为表达式因子**代入后，所得到的表达式。同时注意h的实参代价不能超过阈值 300（**本条与第二次作业的限制不同**）。
- `Cost(f) = Cost(e)`，其中f是自定义函数，`e`是自定义函数f中`=`右部的表达式，本条规则的意思是自定义函数的代价等于右端表达式的代价
- `Cost(dx(a)) = 2 ^ Cost(a)`

如果提交的数据不满足上述数据限制，则该数据将被系统拒绝，且不会用来对同屋其他被测程序进行测试。

三、样例

#	输入	输出	说明
1	0 0 dx((x+1)*(x+2))	2*x+3	对(x+1)*(x+2)求导后，恒等变形展开为2*x+3
2	1 g(x)=x^2+1 0 g(x)+dx(g(x))	x^2+2*x+1	先将g(x)替换为x^2+1，再对其求导得到2*x，最后合并
3	2 g(x,y)=x*sin(y) h(x)=cos(x)^2 0 g(x,h(x))+1	x*sin(cos(x)^2)+1	自定义普通函数的嵌套调用：g(x,h(x)) = x*sin(cos(x)^2)
4	0 1 f{0}(x,y)=x+y f{1}(x,y)=x-y f{n}(x,y)=1*f{n-1}(x^2,y)-2*f{n-2}(x,y^2)+1 f{2}(x,1)	x^2-2*x-2	自定义递推函数展开：f{2}(x,1) 由f{1}与f{0}迭代计算并合并简化
5	1 g(x,y)=x*y 1 f{0}(x)=x^2 f{1}(x)=1 f{n}(x)=2*f{n-1}(x^2)+3*f{n-2}(x^3)+-1 g(f{2}(x),dx(x^2))	6*x^7+2*x	既有自定义普通函数又有自定义递推函数，并含求导算子，最终全部展开
6	1 g(x)=x+y 0 x+1	Wrong Format	函数定义中出现了未在形参中声明的变量y，违反文法规则

第七部分：设计建议

- 在Java程序中，不建议使用静态数组。推荐使用 `ArrayList` 、 `HashMap` 、 `HashSet` 等容器来高效率管理数据对象。
- 在处理输入解析时，可以考虑采用**递归下降解析法**或是正则表达式作为工具。递归下降方法已在先导课程的作业中得到了详尽的介绍与充分的实践，**建议同学们继续沿用**这一方法。而对正则表达式相关的 API 可以了解 `Pattern` 和 `Matcher` 类。
- 这次作业新增的求导算子，可以这样考虑：
  - 对于层次化结构类编写求导方法，对应不同求导法则。
  - 注意求导因子可能出现的地方，特别地：**自定义函数定义时的函数表达式和递推表达式中不会出现求导因子。**

第八部分：提示与警示

一、提示

- Java 内的原生整数类型有 `long` 和 `int`，长度分别为 64 位和 32 位，遇到整数过大的问题，可以使用BigInteger存储。
- 要善于利用java语言内置的工具（工具函数或者工具类等），不要重复造轮子！
- 我们鼓励大家通过 Baidu、Google、Stack Overflow 等方式自行学习和解决问题。比如: 程序运行异常的原因很容易通过搜索引擎了解明白。

- 请注意一个往年遇到比较多的问题，使用`Integer.parseInt(String input)`函数抛出了`java.lang.NumberFormatException`异常，一般是因为传入的`input`字符串其中含有非数字字符，以至于函数无法将之转换为数字。
- 如果还有更多的问题，请到讨论区提问。但是**请善用讨论区**，并在此之前认真阅读包括但不限于课程要求文档、指导书、搜索引擎结果等的内容。关于如何提问。

二、警示

- 如果在互测中发现其他人的代码疑似存在**抄袭**等行为，可向课程组举报，课程组感谢同学们为 OO 课程建设所作出的贡献。