BUAA_OO_第一单元总结

前言

本次面向对象第一单元总结文档将从以下几个部分展开：

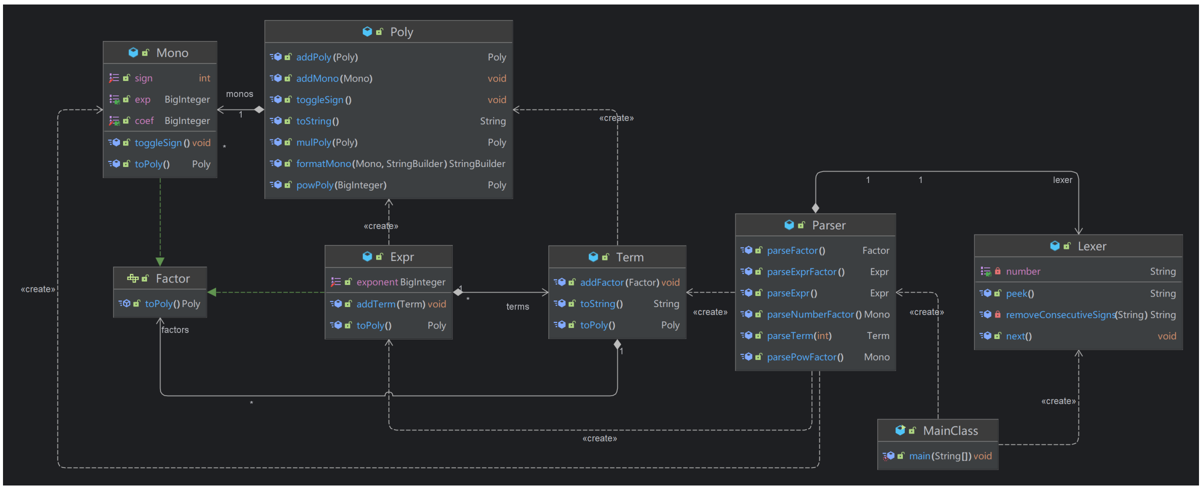
1. 基本思路和框架
2. UML类图
3. 代码规模和复杂度分析
4. 自动化测试概览
5. 互测与hack攻略
6. 心得体会

第一次作业

基本思路和框架

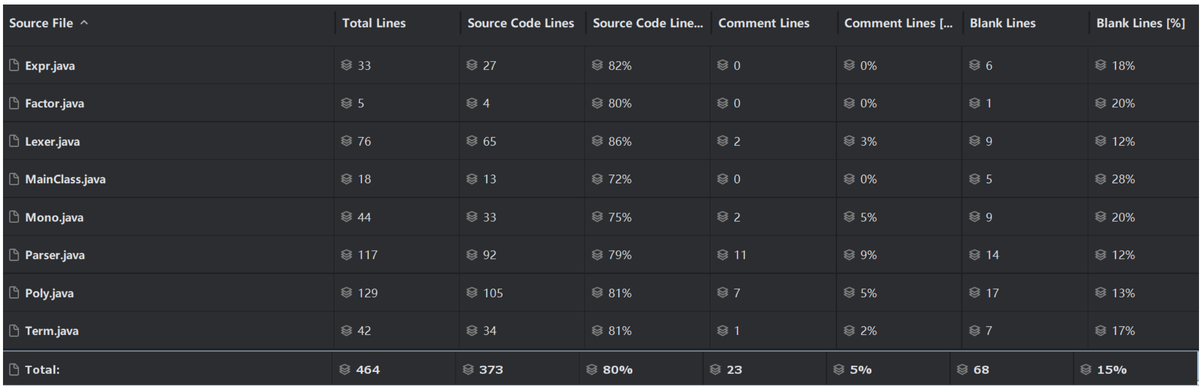
- 目标：读入一个包含加、减、乘、乘方以及括号（其中括号的深度**至多为 1 层**）的**单变量**表达式，输出**恒等变形展开所有括号后**的表达式。性能要求为化简到最简的恒等形式。
- 明确：**开发与测试需求相分离**，是理解题目的关键。题目非常长，尤其是关于设定的形式化表述规则非常复杂。阅读题目的时候一定要明确：哪些是开发需求、哪些是测试规范。其实，本题的**开发需求非常清晰：拿到数学表达式并去括号化简，其中可能有连续加减符号、数字可能有前导0。**其他复杂的格式约束（例如项的构成、前面可以有1个或多个 [加减 空白项] 等）全部为**测试需求**，换言之是构建测试用例（评测机）时需要关注的，和功能开发无关。如果在做开发需求时一直受困于测试需求，那 将无从下手。一句话总结：**抓住开发本质，测试符合规范。**
- 思路：递归下降。**要解析一个表达式（Expr）的时候，可以递归下降解析表达式中的每一个项（Term）；当要解析项的时候，可以递归下降每一个因子（Factor）。**因子是一个接口，下面实现着 Expr 类和 Mono 类，**Mono类为单项式，是形如 `a*x^b` 这种小的运算单元**。
- 框架：参考了练习 `advance` 中的框架和递归下降，并按照 OOLens 公众号中的指导，构建了 Mono 和 Poly 两个运算单元，使整体代码具有良好的结构和可扩展性。 `advance` 中是使用递归下降搭建表达式树，进行后序遍历，其实我们只需要把 `toString` 改为 `toPoly` 就好，**根据运算规律在每一层计算好该层的最简形式，递归输出就是最终的最简形式。**
- 细节：
  - 设计模式很重要，我们使用**工厂模式**。
  - `Lexer` 分词前先进行预处理。用正则表达式**删去所有空白符**，并初步**合并连续加减符号**，在 `getNumber` 中**删去数字的先导0**。但是注意，**如果这个数字就是 0 则要保留，不能删为空，以防他为指数**。我们可以在 Poly 中处理系数为0的情况。
  - 设计简化考虑：顶层 `Expr` 全部由相加的 `Term` 组成，因此 `Term` **中需要定义一个新属性来存储这个项的符号。** `Term` 由因子相乘组成，其中表达式因子无前导符号，而单项式因子（统一变量因子、常数因子、幂函数）有符号。但没必要在 `Mono` 中添加符号属性了，因为直接把符号放在单项式系数中即可。
  - 指数要 `BigInteger`。引用一个往届博客经典案例：`((((((((((x^8)^8)^8)^8)^8)^8)^8)^8)^8)^8)^8`，当然我们这次不允许嵌套。
  - 特别小心 `Parser` 中关于 `Lexer.next()` 的细节。因为这个 de 了好久。

UML类图



`Expr`、`Mono` 实现 `Factor` 接口便于实现多态进行递归调用，`Parser`类（语法分析）依赖 `Lexer` 类（词法分析）进行解析。

代码规模



Source File ^	Total Lines	Source Code Lines	Source Code Line...	Comment Lines	Comment Lines [...	Blank Lines	Blank Lines [%]
Expr.java	33	27	82%	0	0%	6	18%
Factor.java	5	4	80%	0	0%	1	20%
Lexer.java	76	65	86%	2	3%	9	12%
MainClass.java	18	13	72%	0	0%	5	28%
Mono.java	44	33	75%	2	5%	9	20%
Parser.java	117	92	79%	11	9%	14	12%
Poly.java	129	105	81%	7	5%	17	13%
Term.java	42	34	81%	1	2%	7	17%
Total:	**464**	**373**	**80%**	**23**	**5%**	**68**	**15%**

应该来说实现的还是较为简洁，具有良好的可扩展性。

复杂度分析

方法复杂度

先定义三个概念：

> **ev(G)是程序的基本复杂度**，用来衡量程序的非结构化程度。基本复杂度越高，程序越难管理，出现新问题的可能性也越大。
>
> **iv(G)是程序的模块设计复杂度**，衡量的是模块之间的关系和相互调用。也就是我们常说的”高内聚低耦合“的耦合度。
>
> **v(G)是程序的圈复杂度**，用来衡量模块判断结构的复杂性。它表示独立路径的数量，也就是测试程序时需要验证的最少路径数。圈复杂度越大，说明代码更复杂，错误的风险也越高，程序的质量和可维护性可能较差。

Method	CogC	ev(G)	iv(G)	v(G)
Lexer.Lexer(String)	0	1	1	1
Lexer.getNumber()	6	2	5	6
Lexer.next()	4	2	3	10

Method	CogC	ev(G)	iv(G)	v(G)
Lexer.peek()	0	1	1	1
Lexer.removeConsecutiveSigns(String)	17	1	6	9
MainClass.main(String[])	0	1	1	1
Parser.Parser(Lexer)	0	1	1	1
Parser.parseExpr()	8	1	6	6
Parser.parseExprFactor()	3	1	3	3
Parser.parseFactor()	3	3	3	3
Parser.parseNumberFactor()	2	1	3	3
Parser.parsePowFactor()	3	1	3	3
Parser.parseTerm(int)	1	1	2	2
expr.Expr.Expr()	0	1	1	1
expr.Expr.addTerm(Term)	0	1	1	1
expr.Expr.setExponent(BigInteger)	0	1	1	1
expr.Expr.toPoly()	2	1	3	3
expr.Mono.Mono(BigInteger, BigInteger)	0	1	1	1
expr.Mono.getCoef()	0	1	1	1
expr.Mono.getExp()	0	1	1	1
expr.Mono.setCoef(BigInteger)	0	1	1	1
expr.Mono.setSign(int)	1	1	2	2
expr.Mono.toPoly()	0	1	1	1
expr.Mono.toggleSign()	0	1	1	1
expr.Poly.Poly()	0	1	1	1
expr.Poly.addMono(Mono)	0	1	1	1
expr.Poly.addPoly(Poly)	14	4	7	7
expr.Poly.formatMono(Mono, StringBuilder)	3	1	3	3
expr.Poly.mulPoly(Poly)	7	5	3	5
expr.Poly.powPoly(BigInteger)	2	2	3	3
expr.Poly.toString()	12	2	6	6
expr.Poly.toggleSign()	1	1	2	2
expr.Term.Term(int)	0	1	1	1

Method	CogC	ev(G)	iv(G)	v(G)
expr.Term.addFactor(Factor)	0	1	1	1
expr.Term.toPoly()	2	1	3	3
expr.Term.toString()	1	1	2	2
Total	92.0	49.0	85.0	98.0
Average	2.56	1.36	2.36	2.72

整体复杂度不高，比较理想。就是预处理的函数还可以进一步优化解耦。

类复杂度

> OCavg是类平均圈复杂度, 继承类不计入
>
> OCmax应该是类最大圈复杂度
>
> WMC是类总圈复杂度

class	OCavg	OCmax	WMC
expr.Expr	1.5	3.0	6.0
expr.Mono	1.14	2.0	8.0
expr.Poly	3.5	7.0	28.0
expr.Term	1.75	3.0	7.0
Lexer	3.2	6.0	16.0
MainClass	1.0	1.0	1.0
Parser	2.86	5.0	20.0
Total			86.0
Average	2.39	3.86	12.29

自动化测试概览

使用 Python 脚本自动化生成用例， 用 sympy 库进行化简，但注意 sympy 不支持前导0，且幂次方为 **。

互测与hack攻略

提供几个较为容易 `valuable` 的样例:

1. (1-1)^0
2. (1-1)^1   这个天璇星输出1*0，他输入0输出就是1*0，性能分没了，没有刀到任何人
3. `(x^8*x^8*x^8*x^8*x^8*x^8*x^8*x^8*x^8*x^8*x^8*x)^8`
4. (+ + x)+ (- + x)- (x)+ (- - x) 这个天璇星输出x*0，这个开阳星输出为空，错误。我hack了他两次
5. -(x-1)*x^8   没有刀到任何人

6. `(x^2-x^1+002)^1`
7. -1
8. 5161*x   正则表达式
9. -00002*   x - +32*x^   005
10. (999*x)^8   我自己有问题，bug原因是把1*去掉了，别人没有问题
11. `-  +x^0*(-0*0)*(--7*0*x^+1+-0*x^0*09-x^1*5*0*-09--0)` 刀了一个人

第一次作业出现的Bug

1. 不能最后用正则表达式删除1*，比如 `9*9*x` 就会有问题。一个好的解决方法是不要区分第一项，而是只要遇到正项就 append “+”，然后最后用正则表达式删除 “+1*”和“-1*”。
2. 不能在 `Poly.addPoly` 中计算为0就 remove，否则在后面乘方乘法遇到当前为空则会直接 return other。导致连续乘法会有问题，如(1-1)^2*x。
3. 在 `Poly.mulPoly` 中不能遇到0就 Continue。不然 0*1*1 会有问题。原因与上相似。
4. **正项提前**。比如 -x+1 要比 1-x 长。

第二次作业

在第一次的基础上进行迭代，再引入两类因子：三角函数因子和自定义递推函数因子。

自定义递推函数因子是25春课程与往届最大的改动，往届是定义许多个多变量自定义函数，本届是最多定义一个自定义函数，但是要求实现地推定义（实质上就是递归），最多5层。

基本思路和框架

- **明确：**只要是对因子进行扩展的迭代，我们就不需要过多思考。基于现有架构的唯一做法就是扩展 `Mono` 类。毕竟我们上一次的优良传统就是构造一个小的运算单元 `Mono` ，将所有因子转化为 `Mono` 的形式。
- **基本思路：**分为两大步：三角函数和自定义递推函数。

对于**三角函数**，要对 `Mono` 的形式进行扩展。分析考虑采用这样的形式：

$$mono = ceof * x^{exp} * \prod_i sin(poly_i)^{BigInteger_i} * \prod_j cos(poly_j)^{BigInteger_j}$$

究竟如何存储新增的三角函数呢？考虑数据结构的综合复用。三角函数类似于幂函数，由于三角函数的指数一定非负，因此我们浪费了指数的一个属性。不妨考虑将正负属性表示为三角函数的类型，实现数据结构利用的集约化和紧凑化。例如：`private HashMap<Poly, BigInteger> triMap;   // 值正为sin，负为cos`。

加下来对三角函数要做的基本扩展有：

☑ 首先，在 `Parser.parseFactor` 中新增三角函数解析。

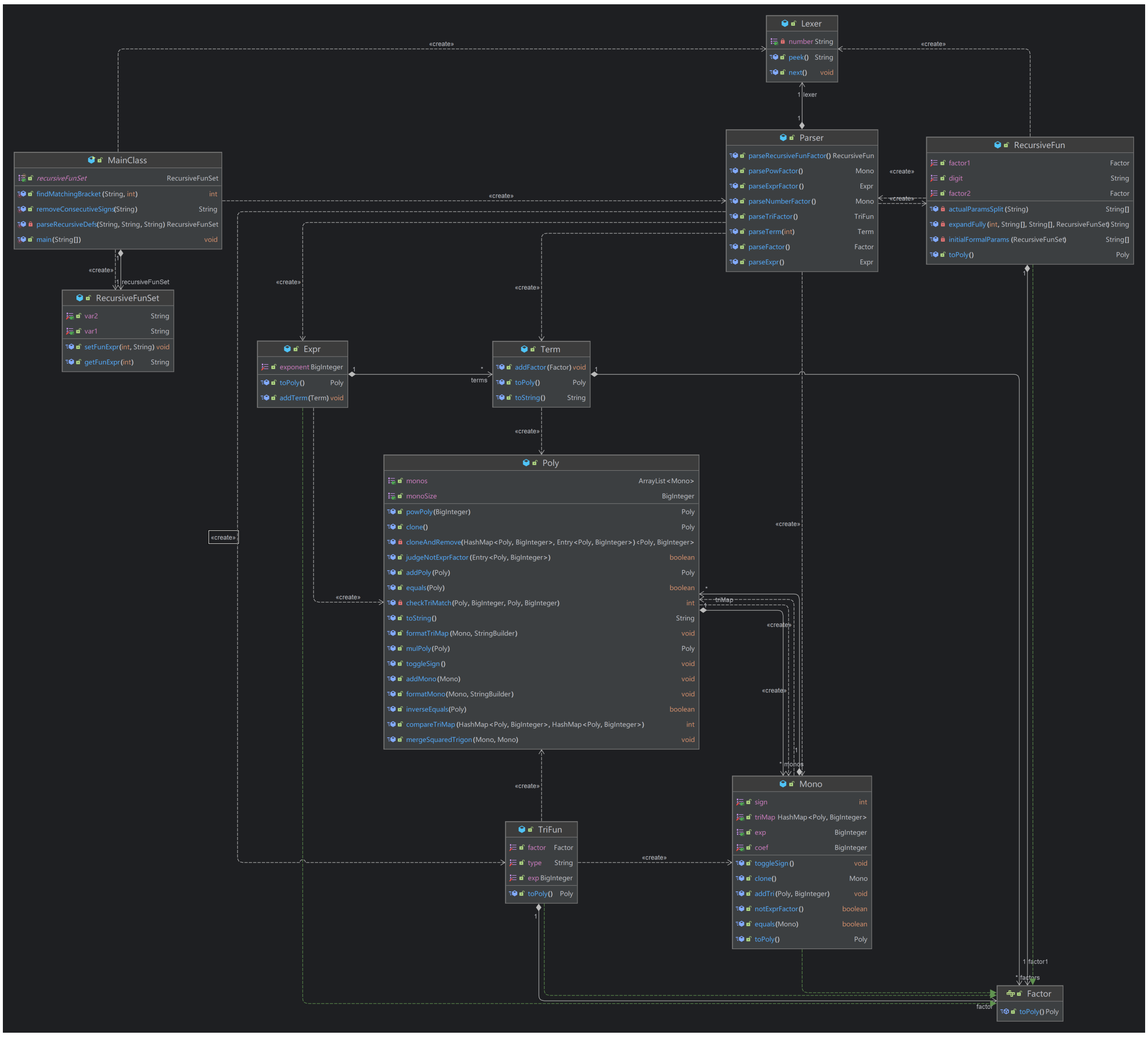
☑ 然后，新建一个三角函数类，实现 `toPoly` 方法。

☑ 接着，在 `Poly` 类中加法、乘法进行修改。考虑到三角函数的计算。

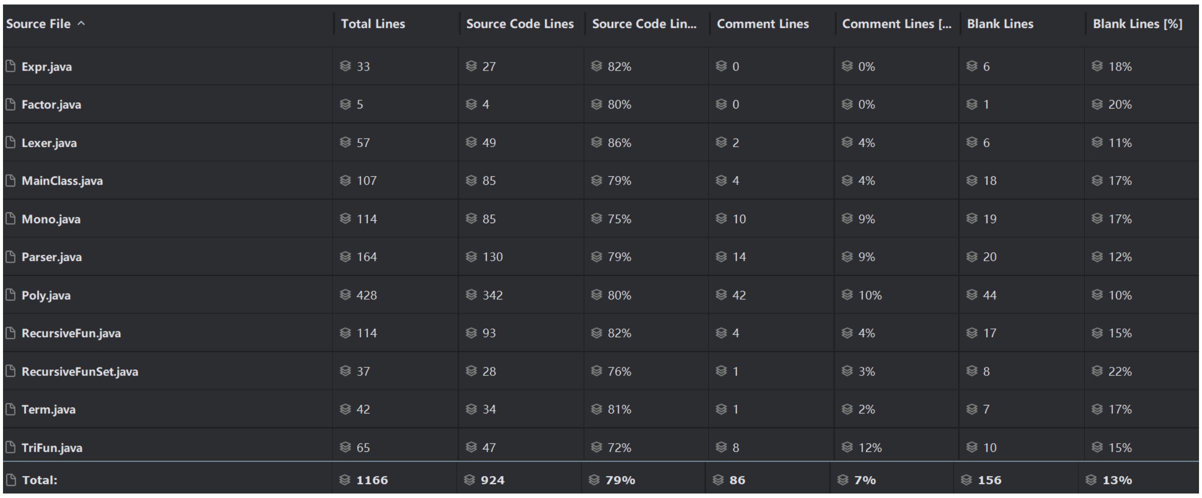
还可以进行进阶优化，例如 `merge: s^2 + c^2 = 1`。优化过程均体现在第三步中。建议打好基本盘，不做过度优化。我只做了 `merge` 优化，没有做提取公因式、二倍角、和差化积等高阶优化。事实证明，越高阶越容易错。

对于**自定义递推函数**，首先定义一个自定义函数模板类`RecursiveFunSet`，用于存储自定义递推函数声明。然后定义一个`RecursiveFun`类作为接口的实现之一，用于递归化简自定义函数、形参换为实参等等。

UML类图



代码规模



Source File ^	Total Lines	Source Code Lines	Source Code Lin...	Comment Lines	Comment Lines [...	Blank Lines	Blank Lines [%]
Expr.java	33	27	82%	0	0%	6	18%
Factor.java	5	4	80%	0	0%	1	20%
Lexer.java	57	49	86%	2	4%	6	11%
MainClass.java	107	85	79%	4	4%	18	17%
Mono.java	114	85	75%	10	9%	19	17%
Parser.java	164	130	79%	14	9%	20	12%
Poly.java	428	342	80%	42	10%	44	10%
RecursiveFun.java	114	93	82%	4	4%	17	15%
RecursiveFunSet.java	37	28	76%	1	3%	8	22%
Term.java	42	34	81%	1	2%	7	17%
TriFun.java	65	47	72%	8	12%	10	15%
Total:	**1166**	**924**	**79%**	**86**	**7%**	**156**	**13%**

第二次作业出现的bug

细节决定成败。出现了诸多bug，包括提交前自己发现的，和互测时被🔪的。下面做一个汇总：

1. `Merge` 中有一个复杂的条件嵌套 `(( && ) || ( && )) && ...` 中括号嵌套出现了笔误导致逻辑错误。
2. 在`Poly.equals` 方法中不应该管系数为零的情况。因为在我的处理结构下，完全可能有很多系数为0的项，导致两个 `Poly.monos` 的 `size` 不相等。不能单纯通过 `size` 判断。
3. `Poly.cloneAndRemove` 方法不应该判断 `Key(Poly)` 是否相等，而是应该直接利用自带的 `equals` 方法判断两个键值对是否相等。否则可能出现问题。
4. 扫描自定义递推函数时不应该在任何地方出现正则表达式的方法（不管是在 `Main` 中还是在处理过程中），因为可能会出现括号匹配错误。应该老老实实写一个栈用于括号匹配。感谢 wcr 的提醒。
5. `sin(0)^0`我输出0。经检查原因是我 `TriFun.toPoly` 中判断如果是 `sin(0)` 就返回0的时候忘记 `&& exp.compareTo(BigInteger.ZERO) != 0`。
6. 替换形参时应该先替换 x 再替换 y！如果相反，则会把之前将 y 换成有关 x 的实参中的那个 x 也替换成新的实参！
7. `Poly.judgeNotExprFactor` 中正确写法是 `mono.getCoef().compareTo(BigInteger.ZERO) != 0` 。误写成了 `mono.getCoef() != BigInteger.ZERO`。

互测与hack攻略

用例：

```
0
sin((-sin((-x))))

sin(sin(x))      // 天权、天玑、开阳、摇光、玉璇没化简，天枢输出错误
```

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

10       //全部正确
```

```
0
sin((  sin(x)^2+cos(x)^2  ))^2+cos(1)^2

1        //除了天枢都没化简，但是天枢运行时间过长
```

```
0
sin(x)^2+cos(x)^2+sin(x)^2+cos(x)^2+sin(x)^2+cos(x)^2+sin(x)^2+cos(x)^2+sin(x)^2+cos(x)^2+sin(x)^2+cos(x)^2

6        // 全部正确，除了天枢都没化简，但是天枢运行时间过长
```

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

0       // 正确
```

```
0
-cos(2)*cos(x)-cos(x)

-cos(2)*cos(x)-cos(x)   //天枢运行时间过长
```

```
0
sin(-1111111111111111111111111111111111111111111)

-sin(1111111111111111111111111111111111111111111)   //开阳错，已提交
```

```
0
sin((-x))^2

sin(x)^2    //天枢输出错误，已提交
```

```
0
cos(sin(cos(sin(cos(sin(cos(sin(cos(0)^2))))))))

cos(sin(cos(sin(cos(sin(cos(sin(cos(x))))))))      // 全对
```

本来我有bug，现已修复：

```
0
-sin(x)^+2+-sin(x)*cos(x)^+2++cos(x)*x

x*cos(x)-sin(x)^2-cos(x)^2*sin(x)       // 全对
```

本来我的另一个bug，现已修复，罪魁祸首是正则表达式：

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

sin(x)-4*x^2+1      //全对
```

```
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{5}(sin((x+1)^2)^2, x)

```

```
1225*sin((x^2+2*x+1))^6+1225*x^4    //天权错！！！摇光错！！！玉璇错！！！

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{3}(sin(((x+1)^2))^2， x)

35*sin((x^2+2*x+1))^6+35*x^2        // 已提交

1
f{0}(x， y) = x - y
f{n}(x， y) = 0*f{n-1}(x， y) + 35*f{n-2}(x， y^2)+sin(0)^0
f{1}(x， y) = x^3 + y
f{3}(((x+1)^2)^2， x)        //玉璇错，天权错，摇光错，已提交

36+7700*x^3+17325*x^4+420*x+2345*x^2+27720*x^5+32340*x^6+27720*x^7+17325*x^8+420*x^11+35*x^12+7700*x^9+2310*x^10
```

```
0
cos(sin((2*(x^2))))^2

cos(sin((2*x^2)))^2     //天枢错！少了一层括号，已提交

0
sin((-x))^2+cos(sin((2*(x^2))))^2   //交的这个
```

```
0
((((((((((x^8)^8)^8)^8)^8)^8)^8)^8)^8)^8)^8

x^68719476736       //天权错，开阳错
```

```
0
sin(0)^0

1       //天权错，我自己也有bug
```

自己构造：

```
1
f{0}(x， y) = sin(x)
f{1}(x， y) = 1
f{n}(x， y) = 1*f{n-1}(sin(y),1) - 4*f{n-2}(y^2,x) + 1
f{5}(x,sin(x))

12*sin(1)-3+16*sin(sin(sin(x))^2)       //全对
```

```
1
f{1}(x, y) = cos((x+y)) + x
f{n}(x,y)   = 2*f{n-1}((x*y), (x-y))-1*f{n-2}(sin(x), cos(x))+y
f{0}(x, y) = x*y
f{2}((x+2), f{3}((x*sin(x)), 3))+7
```

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

7-27*x^2    //全对
```

```
1
f{0}(x, y) = y
f{1}(x, y) = y
f{n}(x, y) = 1*f{n-1}(x,1) - 4*f{n-2}(y,y) + 1
f{1}((x+2), f{1}(x, sin(3)))+7

sin(3)+7
```

上面这个是自己化简的

```
0
sin(0)^2+cos(0)^2       // 天枢错，已提交
```

```
1
f{n}(y) = 1*f{n-1}(y) - 4*f{n-2}(y) + 1
f{0}(y) = sin(((y))^2)
f{1}(y) = y
f{2}(f{2}(sin((x+1)^2)^2))      //已提交，水数据

sin((x^2+2*x+1))^2-4*sin(sin((x^2+2*x+1))^4)+2-4*sin((sin((x^2+2*x+1))^4-8*sin((x^2+2*x+1))^2*sin(sin((x^2+2*x+1))^4)+2*sin((x^2+2*x+1))^2+16*sin(sin((x^2+2*x+1))^4)^2-8*sin(sin((x^2+2*x+1))^4)+1))
```

我的bug:

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

```
0
sin(0)^0
```

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

现已全部修复。

第三次作业

本次作业新增求导因子和自定义普通函数因子（最多2个，g 和 h）。

基本思路和框架

明确：求导因子和自定义普通函数在指导书中都明确标识为“因子”。所以不需要过多思考，扩展做法就是**在 `Parser.parseFactor` 中增加两种情况**，然后分别实现 `Derivation` 类和 `OrdinaryFun` 类，均实现 `Factor` 接口，均具有 `toPoly` 方法。

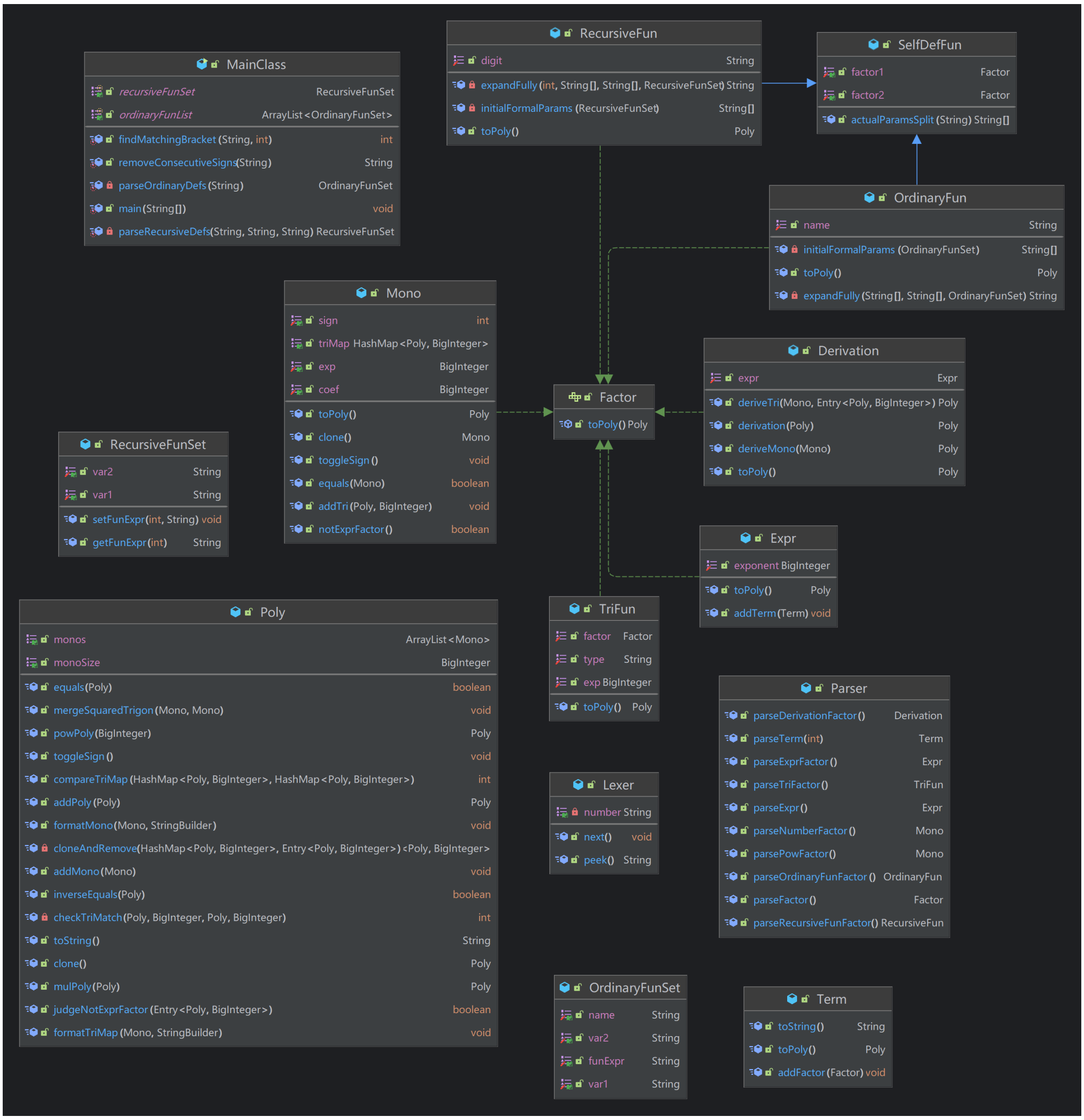
思路：

☀️ 对于求导因子的 `toPoly` 方法，可以直接 `return derivation(expr.toPoly());` ，其中 `expr` 是 `dx()` 括号中的表达式（注意不是表达式因子，注意区分），其中 `derivation` 是该类中应用于多项式的求导法则。

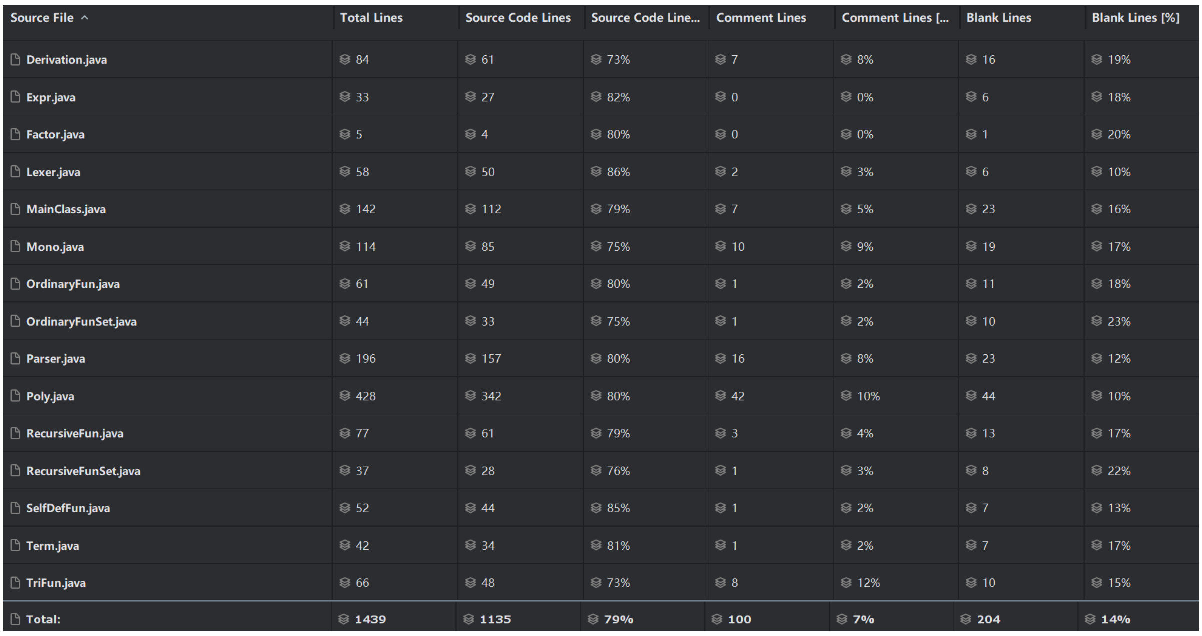
☀️ 对于自定义普通函数因子，完全是自定义递推函数的简化版，唯一要变的是 `MainClass` 中要定义一个存储容器，因为最多可能有2个自定义普通函数。实现策略、形参实参替换策略均和上一次相同。

☀️ 由于自定义普通函数和自定义递推函数有太多属性和方法上的相同，从设计模式的角度可以提取出一个父类`SelfDefFun`，**继承公共属性（实参因子）和方法（替换等），然后让两种自定义函数分别继承父类且实现 `Factor` 接口**，和理论课讲的模式相对应。

UML 类图


RecursiveFun
digit String
expandFully (int, String[], String[], RecursiveFunSet) String
initialFormalParams (RecursiveFunSet) String[]
toPoly() Poly
SelfDefFun
factor1 Factor
factor2 Factor
actualParamsSplit (String) String[]
MainClass
recursiveFunSet RecursiveFunSet
ordinaryFunList ArrayList<OrdinaryFunSet>
findMatchingBracket (String, int) int
removeConsecutiveSigns(String) String
parseOrdinaryDefs (String) OrdinaryFunSet
main(String[]) void
parseRecursiveDefs(String, String, String) RecursiveFunSet
OrdinaryFun
name String
initialFormalParams (OrdinaryFunSet) String[]
toPoly() Poly
expandFully (String[], String[], OrdinaryFunSet) String
Mono
sign int
triMap HashMap<Poly, BigInteger>
exp BigInteger
coef BigInteger
toPoly() Poly
clone() Mono
toggleSign() void
equals(Mono) boolean
addTri(Poly, BigInteger) void
notExprFactor() boolean
Derivation
expr Expr
deriveTri(Mono, Entry<Poly, BigInteger>) Poly
derivation(Poly) Poly
deriveMono(Mono) Poly
toPoly() Poly
Factor
toPoly() Poly
RecursiveFunSet
var2 String
var1 String
setFunExpr(int, String) void
getFunExpr(int) String
Expr
exponent BigInteger
toPoly() Poly
addTerm(Term) void
Poly
monos ArrayList<Mono>
monoSize BigInteger
equals(Poly) boolean
mergeSquaredTrigon(Mono, Mono) void
powPoly(BigInteger) Poly
toggleSign() void
compareTriMap (HashMap<Poly, BigInteger>, HashMap<Poly, BigInteger>) int
addPoly(Poly) Poly
formatMono(Mono, StringBuilder) void
cloneAndRemove(HashMap<Poly, BigInteger>, Entry<Poly, BigInteger>) <Poly, BigInteger>
addMono(Mono) void
inverseEquals(Poly) boolean
checkTriMatch(Poly, BigInteger, Poly, BigInteger) int
toString() String
clone() Poly
mulPoly(Poly) Poly
judgeNotExprFactor(Entry<Poly, BigInteger>) boolean
formatTriMap (Mono, StringBuilder) void
TriFun
factor Factor
type String
exp BigInteger
toPoly() Poly
Parser
parseDerivationFactor() Derivation
parseTerm(int) Term
parseExprFactor() Expr
parseTriFactor() TriFun
parseExpr() Expr
parseNumberFactor() Mono
parsePowFactor() Mono
parseOrdinaryFunFactor() OrdinaryFun
parseFactor() Factor
parseRecursiveFunFactor() RecursiveFun
Lexer
number String
next() void
peek() String
OrdinaryFunSet
name String
var2 String
funExpr String
var1 String
Term
toString() String
toPoly() Poly
addFactor(Factor) void

代码规模



Source File ^	Total Lines	Source Code Lines	Source Code Line...	Comment Lines	Comment Lines [...	Blank Lines	Blank Lines [%]
Derivation.java	84	61	73%	7	8%	16	19%
Expr.java	33	27	82%	0	0%	6	18%
Factor.java	5	4	80%	0	0%	1	20%
Lexer.java	58	50	86%	2	3%	6	10%
MainClass.java	142	112	79%	7	5%	23	16%
Mono.java	114	85	75%	10	9%	19	17%
OrdinaryFun.java	61	49	80%	1	2%	11	18%
OrdinaryFunSet.java	44	33	75%	1	2%	10	23%
Parser.java	196	157	80%	16	8%	23	12%
Poly.java	428	342	80%	42	10%	44	10%
RecursiveFun.java	77	61	79%	3	4%	13	17%
RecursiveFunSet.java	37	28	76%	1	3%	8	22%
SelfDefFun.java	52	44	85%	1	2%	7	13%
Term.java	42	34	81%	1	2%	7	17%
TriFun.java	66	48	73%	8	12%	10	15%
Total:	1439	1135	79%	100	7%	204	14%

```
0
0
dx(sin(x)^0)

0
```

```
0
0
dx(sin(dx(x*sin(x))))

cos(x)*cos((sin(x)+x*cos(x)))+cos((sin(x)+x*cos(x)))*cos(x)-x*cos((sin(x)+x*cos(x)))*sin(x)
```

```
0
0
dx(sin(dx(x*sin(dx(x)))))

0
```

```
2
g(x) = sin(x)
h(x) = g(x)
0
(g(x))^2+(h(x)-sin(x)+cos(x))^2

1
```

```
2
h(x)=x^2
g(x) = h(x^4)+1
0
g((dx(x)+sin(x)))

2+8*sin(x)+28*sin(x)^2+56*sin(x)^3+70*sin(x)^4+56*sin(x)^5+28*sin(x)^6+8*sin(x)^7+sin(x)^8
```

我认为应该可以：

```
1
g(x,y)= y+sin(x)
1
f{0}(x,y) = 1
f{n}(x,y) = 0*f{n-1}(0,0)+0*f{n-2}(0,0)
f{1}(x,y) = g(1,cos(x))+y
f{1}(1,dx(cos(x)))

cos(1)+sin(1)-sin(x)
```

```
1
g(x,y)=x*y
1
f{0}(x)=x^2
f{1}(x)=sin(x)^2
f{n}(x)=2*f{n-1}(x^2)+3*f{n-2}(x^3)+-1
g(g(f{2}(x),dx(x^2)),dx(g(f{2}(x),dx(x^2)) + cos((2*dx(2*x)))))

16*x*sin(x^2)^4+64*x^3*cos(x^2)*sin(x^2)^3+192*x^7*sin(x^2)^2-16*x*sin(x^2)^2+96*x^9*cos(x^2)*sin(x^2)+252*x^13-96*x^7-32*x^3*sin(x^2)*cos(x^2)+4*x
```

```
2
g(y,x)=x*sin(y)
h(y,x)=g(g(cos(x),y^2),4)+sin(((0)))^00
1
f{1}(y,x)=g(y,x)
f{0}(y,x)=x+sin(y)
f{n}(y,x)=3*f{n-1}(x,x^2)+2*f{n-2}(sin(x),x)+-1
f{3}(dx(x+2), f{3}(dx(g(x,x^2)),3))+dx(sin(h(g(x,cos(x)^2),dx(x))))

测试说明
1.函数表达式支持调用其他已定义的自定义普通函数
2.函数表达式支持嵌套调用其他已定义的自定义普通函数
3.求导因子可以出现在函数调用实参
4.求导因子可以出现在三角函数内部
5.自定义递推函数顺序可以任意
6.递推表达式最后可以加函数表达式
7.自定义递推函数调用可以嵌套
8.自定义普通函数调用可以嵌套
9.求导因子可以嵌套自定义普通函数


```

```
2541865828329*sin(9)^4*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin
(9)+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9
))^2+72*sin(sin(3))*sin(sin(9))+36*sin(sin(3))^2))+697356880200*sin((531441*s
in(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(sin(3))*sin(9)+2500+600
*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(3))*sin(sin(9))+36*s
in(sin(3))^2))*sin(9)^3+83682825624*sin(sin(9))*sin((531441*sin(9)^2+72900*si
n(9)+8748*sin(9)*sin(sin(9))+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600
*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(3))*sin(sin(9))+36*sin(sin(3))^2))*s
in(9)^3+83682825624*sin(9)^3*sin(sin(3))*sin((531441*sin(9)^2+72900*sin(9)+87
48*sin(9)*sin(sin(9))+8748*sin(9)*sin(sin(3))+2500+600*sin(sin(9))+600*sin(si
n(3))+36*sin(sin(9))^2+72*sin(sin(3))*sin(sin(9))+36*sin(sin(3))^2))+71744535
000*sin(9)^2*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*s
in(9)*sin(sin(3))+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*si
n(sin(9))*sin(sin(3))+36*sin(sin(3))^2))+17218688400*sin(9)^2*sin((531441*sin
(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*sin(9)*sin(sin(3))+2500+600*s
in(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin
(sin(3))^2))*sin(sin(9))+17218688400*sin(9)^2*sin((531441*sin(9)^2+72900*sin(
9)+8748*sin(9)*sin(sin(9))+8748*sin(9)*sin(sin(3))+2500+600*sin(sin(9))+600*s
in(sin(3))+36*sin(sin(9))^2+72*sin(sin(3))*sin(sin(9))+36*sin(sin(3))^2))*sin
(sin(3))+1033121304*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))
+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^
2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))*sin(9)^2*sin(sin(9))^2+206624
2608*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*sin(9)*si
n(sin(3))+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(3)
)*sin(sin(9))+36*sin(sin(3))^2))*sin(sin(3))*sin(9)^2*sin(sin(9))+1033121304*
sin(9)^2*sin(sin(3))^2*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin
(9)+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9
))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))+3280500000*sin((531441*sin
(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(sin(3))*sin(9)+2500+600*s
in(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin
(sin(3))^2))*sin(9)+1180980000*sin(sin(9))*sin((531441*sin(9)^2+72900*sin(9)+
8748*sin(9)*sin(sin(9))+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(
sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))*sin(9)
+1180980000*sin(sin(3))*sin(9)*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin
(9))*sin(9)+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*s
in(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))+141717600*sin(sin(
9))^2*sin(9)*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*s
in(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*si
n(sin(9))*sin(sin(3))+36*sin(sin(3))^2))+283435200*sin(9)*sin((531441*sin(9)^
2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*sin(sin(3))*sin(9)+2500+600*sin(s
in(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(3))*sin(sin(9))+36*sin(sin
(3))^2))*sin(sin(3))*sin(sin(9))+141717600*sin(sin(3))^2*sin((531441*sin(9)^2
+72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(sin(3))*sin(9)+2500+600*sin(si
n(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(
3))^2))*sin(9)+5668704*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(
9))+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9
))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))*sin(9)*sin(sin(9))^3+17006
112*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*sin(9)*sin
(sin(3))+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(3))
*sin(sin(9))+36*sin(sin(3))^2))*sin(9)*sin(sin(9))^2*sin(sin(3))+17006112*sin
(sin(9))*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*sin(s
in(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(si
n(3))*sin(sin(9))+36*sin(sin(3))^2))*sin(9)*sin(sin(3))^2+5668704*sin(9)*sin(
sin(3))^3*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*sin(
sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(s
in(3))*sin(sin(9))+36*sin(sin(3))^2))+56250000*sin((531441*sin(9)^2+72900*sin
(9)+8748*sin(9)*sin(sin(9))+8748*sin(9)*sin(sin(3))+2500+600*sin(sin(9))+600*
sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))+27
000000*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(sin
(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(
3))*sin(sin(9))+36*sin(sin(3))^2))*sin(sin(9))+27000000*sin((531441*sin(9)^2+
72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(sin(3))*sin(9)+2500+600*sin(sin
(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3
))^2))*sin(sin(3))+4860000*sin(sin(9))^2*sin((531441*sin(9)^2+72900*sin(9)+87
48*sin(sin(9))*sin(9)+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(si
n(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))+9720000*
sin((531441*sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(9)*sin(sin
(3))+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin
(sin(3))+36*sin(sin(3))^2))*sin(sin(3))*sin(sin(9))+4860000*sin((531441*sin(9
)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(9)*sin(sin(3))+2500+600*sin
(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(3))*sin(sin(9))+36*sin(s
in(3))^2))*sin(sin(3))^2+388800*sin(sin(9))^3*sin((531441*sin(9)^2+72900*sin(
9)+8748*sin(sin(9))*sin(9)+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*s
in(sin(3))+36*sin(sin(9))^2+72*sin(sin(3))*sin(sin(9))+36*sin(sin(3))^2))+116
6400*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*sin(sin(3
))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(3)
)*sin(sin(9))+36*sin(sin(3))^2))*sin(sin(9))^2*sin(sin(3))+1166400*sin(sin(3)
)^2*sin(sin(9))*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+874
8*sin(9)*sin(sin(3))+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72
*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))+388800*sin(sin(3))^3*sin((531441*
sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(sin(3))*sin(9)+2500+60
0*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*
sin(sin(3))^2))+11664*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(
9)+8748*sin(9)*sin(sin(3))+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9)
)^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))*sin(sin(9))^4+46656*sin(sin
(3))*sin(sin(9))^3*sin((531441*sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(9))+
8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2
+72*sin(sin(3))*sin(sin(9))+36*sin(sin(3))^2))+69984*sin(sin(3))^2*sin((53144
1*sin(9)^2+72900*sin(9)+8748*sin(sin(9))*sin(9)+8748*sin(9)*sin(sin(3))+2500+
600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+3
6*sin(sin(3))^2))*sin(sin(9))^2+46656*sin(sin(9))*sin((531441*sin(9)^2+72900*
sin(9)+8748*sin(9)*sin(sin(9))+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+6
00*sin(sin(3))+36*sin(sin(9))^2+72*sin(sin(3))*sin(sin(9))+36*sin(sin(3))^2))
*sin(sin(3))^3+11664*sin(sin(3))^4*sin((531441*sin(9)^2+72900*sin(9)+8748*sin
(sin(9))*sin(9)+8748*sin(sin(3))*sin(9)+2500+600*sin(sin(9))+600*sin(sin(3))+
36*sin(sin(9))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2))+3188646*sin(9)
^2+437400*sin(9)+52488*sin(sin(9))*sin(9)+52488*sin(sin(3))*sin(9)+14996+3600
*sin(sin(9))+3600*sin(sin(3))+216*sin(sin(9))^2+432*sin(sin(9))*sin(sin(3))+2
16*sin(sin(3))^2+6*sin(sin((531441*sin(9)^2+72900*sin(9)+8748*sin(9)*sin(sin(
9))+8748*sin(9)*sin(sin(3))+2500+600*sin(sin(9))+600*sin(sin(3))+36*sin(sin(9
))^2+72*sin(sin(9))*sin(sin(3))+36*sin(sin(3))^2)))+1458*sin(sin((729*sin(9)+
50+6*sin(sin(9))+6*sin(sin(3)))))*sin(9)+100*sin(sin((729*sin(9)+50+6*sin(sin
(9))+6*sin(sin(3)))))+12*sin(sin((729*sin(9)+50+6*sin(sin(9))+6*sin(sin(3))))
)*sin(sin(9))+12*sin(sin((729*sin(9)+50+6*sin(sin(9))+6*sin(sin(3)))))*sin(si
n(3))-16*cos(x)^3*cos((sin(cos(1))*sin(x)^2*cos(x)^4))*cos((4*sin((cos(x)^4*s
in(x)^2*sin(cos(1))))+1))*sin(x)^3*sin(cos(1))+8*cos(x)^5*cos((cos(x)^4*sin(c
os(1))*sin(x)^2))*sin(cos(1))*cos((4*sin((cos(x)^4*sin(x)^2*sin(cos(1))))+1))
*sin(x)
```

```
0
0
dx(sin(x)*cos(x)*x^2)

2*x*cos(x)*sin(x)-x^2*sin(x)^2+x^2*cos(x)^2
```

天璇星有问题。

```
2
g(y,x)=x*sin(y)
h(y,x)=g(g(cos(x),y^2),4)
1
f{1}(y,x)=g(y,x)
f{0}(y,x)=x+sin(y)
f{n}(y,x)=3*f{n-1}(x,x^2)+2*f{n-2}(sin(x),x)+-1
f{3}(dx(x+2), f{3}(dx(g(x,x^2)),3))+dx(sin(h(g(x,cos(x)^2),dx(x))))

2
g(y,x)=x*sin(y)
h(y,x)=g(g(cos(x),y^2),4)
0
dx(sin(h(g(x,cos(x)^2),dx(x))))

2
g(y,x)=x*sin(y)
h(y,x)=g(g(cos(x),y^2),4)
0
dx(h(g(x,cos(x)^2),1))

2
g(y,x)=x*sin(y)
h(y,x)=g(g(cos(x),y^2),4)
0
h(g(x,cos(x)^2),1)

0
0
dx(4*sin((cos(x)^4*sin(x)^2*sin(cos(1)))))

0
0
dx(4*sin((cos(x)^4*sin(x)^2)))

0
0
dx(4*(cos(x)^3*sin(x)))

最终拿下数据：
0
0
12*sin(x)^2*cos(x)^2

12*sin(x)^2*cos(x)^2    // 天璇星二倍角化简错误
```

```
0
0
dx(cos(sin(cos(sin(cos(sin(cos(sin(cos(cos(sin(cos(sin(cos(sin(cos(sin(cos(x))
)))))))))))))))))
```