

# 面向对象设计与构造2022第一单元总结

## 零、任务简介

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、自定义函数调用以及求和函数的表达式，输出恒等变形展开所有括号后的表达式。

展开所有括号的定义是：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与求和函数，且只包含必要的括号。

示例输入：

```
2
f(x)=x**2
g(y)=sin(y)
f(g(x**2))
```

示例输出：

```
sin(x**2)**2
```

## 一、Task1

### 1) 任务说明

- 不存在自定义函数，三角函数，求和函数
- 括号最多嵌套一层
- 指数次数最高为8次

### 2) 整体结构

本任务架构以计算为核心目标

本任务架构基于**递归下降语法分析**，采用边解析边运算的方法，但是为了设计方便起见，依然保留了中间结果储存这一环节。在解析的过程中，现有层次的单元的数据被不断进行更新，最终得到一个包含表达式全部信息的顶层单元。

**(递归下降语法分析：为每一个语法成分编写一个可递归调用的分析程序，进行自顶向下分析的语法分析)**

以下是本任务的基本逻辑构成，以流程图形式给出。注意到，笔者将字符串的复杂处理，放在了最开始。将原有的表达式字符串转换成了一个以自定义形式构建的字符串，减小了后续流程中，字符串解析的难度和压力。



三次作业的基本逻辑构成区别不大，且在第一次作业中，笔者就已经支持部分后续功能，因此在 Task2 及 Task3 中不再赘述，仅叙述重复部分。

基于题干给出的形式化表达，以表达式，项，因子作为三个基本单元进行分析。








注意到三者并非线性的父子关系，因子的属性中依然可能包含表达式，这是一个递归嵌套结构。

UML类图如下所示，部分类中的私有函数与整体逻辑无关，已隐去。

### 3) 效能分析

| class ▲   | OCavg       | OCmax       | WMC          |
|---|-------------|-------------|--------------|
|  Expr      | 1.60        | 4           | 16           |
|  Factor    | 3.00        | 17          | 42           |
|  MainClass | 7.33        | 12          | 22           |
|  Term      | 2.00        | 5           | 18           |
| <b>Total</b>  |             |             | <b>98</b>    |
| <b>Average</b>  | <b>2.72</b> | <b>9.50</b> | <b>24.50</b> |

类复杂度分析如上，注意到MainClass因为包含print函数的缘故，OCavg比较高。

| method ▼  | CogC | ev(G) | iv(G) | v(G) |
|---|------|-------|-------|------|
|  Factor.generateAnswer()                     | 53   | 1     | 16    | 17   |
|  MainClass.printI(int, BigInteger[], StringB | 41   | 1     | 12    | 12   |
|  Factor.generateExpr()                      | 16   | 1     | 10    | 10   |
|  MainClass.printAnswer(BigInteger[])       | 15   | 3     | 8     | 9    |
|  Term.generateAnswer()                     | 8    | 1     | 4     | 5    |
|  Factor.Factor(String)                     | 6    | 1     | 4     | 4    |
|  Term.multi(BigInteger[])                  | 6    | 1     | 4     | 4    |

方法(部分)分析如上，注意到部分函数的复杂度较高。但由于方法较多，总体平均值还算良好。

|                |             |             |             |             |
|----------------|-------------|-------------|-------------|-------------|
| <b>Total</b>   | <b>154</b>  | <b>38</b>   | <b>95</b>   | <b>98</b>   |
| <b>Average</b> | <b>4.28</b> | <b>1.06</b> | <b>2.64</b> | <b>2.72</b> |

### 4) 互测体验

强测AC

Hack一次，被hack一次。

- 发现bug：错误的合并同类型，乘方运算相消时没有清零，导致乘方运算中有概率会出现bug。由于乘方相消概率不大，并未被强测或作者检出。
- 被发现bug：解析错误，误认为+--1情况不会出现。

### 5) 性能优化

- $x^{**2} \rightarrow x*x$
- $-1+x \rightarrow x-1$  (优先输出正项)

### 6) 额外完成任务

- 支持括号嵌套

## 二、Task2

## 1) 任务说明

- 自定义函数，求和函数不会相互嵌套
- 三角函数中仅为因子
- 括号最多嵌套一层
- 指数次数单次出现最高为8次

## 2) 整体结构

与Task1相同部分不再赘述。























自定义函数，求和函数的处理方法，为字符串替换。特别的，在替换完成后，在自变量前后，以及表达式前后，添加括号。











(此处需要支持括号嵌套)

此处新建了两个哈希表来存储正弦余弦值，内容为字符串。

此架构对三角函数优化并不友好，笔者本着正确性优先的原则，放弃了部分三角函数优化。

## 3) 效能分析

| class ▲   | OCavg | OCmax | WMC   |
|---|-------|-------|-------|
|   Coefficient | 3.00  | 13    | 21    |
|   CosList     | 2.60  | 8     | 13    |
|   Expression  | 2.83  | 5     | 17    |
|   Factor      | 6.40  | 10    | 32    |
|   Item        | 5.62  | 12    | 45    |
|   Main        | 10.50 | 12    | 21    |
|   MainClass   | 4.89  | 9     | 44    |
|   NumberLis   | 1.00  | 1     | 3     |
|   SinList     | 2.60  | 8     | 13    |
|   UserDefine  | 2.33  | 6     | 14    |
|   ValueList   | 1.00  | 1     | 2     |
| Total   |       |       | 225   |
| Average   | 3.88  | 7.73  | 20.45 |

| method  | ▼ | CogC | ev(G) | iv(G) | v(G) |
|---|---|------|-------|-------|------|
|   Main.printl(int, BigInteger[], StringBuilder) |   | 41   | 1     | 12    | 12   |
|  CoefficientList.equals(Object)  |   | 33   | 13    | 13    | 19   |
|  Item.multiply(HashMap<CoefficientList, I  |   | 31   | 1     | 18    | 18   |
|  MainClass.plusCos(StringBuilder, Entry<C  |   | 31   | 1     | 11    | 11   |
|  MainClass.plusSin(StringBuilder, Entry<C  |   | 31   | 1     | 11    | 11   |
|  Item.indexJudge(String, UserDefineFunc  |   | 29   | 1     | 9     | 9    |
|  Item.setPoly(String, UserDefineFunction)  |   | 16   | 1     | 9     | 9    |
|   Factor.deleteUseLess(String)                  |   | 15   | 5     | 5     | 11   |

不难想见，由于采取了递归下降的做法，会出现部分类，部分方法的复杂度远超平均的现象。由于内容相比 Task1 明显增多，各复杂度均有上升。

|         |      |      |      |      |
|---------|------|------|------|------|
| Total   | 392  | 97   | 220  | 263  |
| Average | 6.76 | 1.67 | 3.79 | 4.53 |

## 4) 互测体验

强测AC

Hack一次，被hack零次。

- 发现bug：sin(0) - > 1

## 5) 性能优化

- sin(0) - > 0
- cos(0) - > 1
- sin(-x) - > -sin(x)
- cos(-x) - > cos(x)

## 6) 额外完成任务

- 支持括号嵌套
- 支持大指数
- 支持自定义函数/求和函数的嵌套

# 三、Task3

## 1) 任务说明

- 无限制

## 2) 整体结构











与Task2相同部分不再赘述。








由于在Task2中已经实现大部分内容，只需要支持三角函数内部为表达式即可。

由于在Task2中，定义三角函数内部为字符串，故调用表达式的 toString方法，并对其重写即可。

不难发现，Task3的架构与Task2基本相同。

### 3) 效能分析

| class ▲   | OCavg       | OCmax       | WMC          |
|---|-------------|-------------|--------------|
|  Coefficient | 3.25        | 13          | 26           |
|  CosList     | 1.00        | 1           | 3            |
|  Expression  | 3.87        | 9           | 58           |
|  Factor      | 4.25        | 10          | 34           |
|  Item        | 4.70        | 10          | 47           |
|  MainClass   | 1.50        | 2           | 3            |
|  NumberLis   | 1.00        | 1           | 3            |
|  SinList     | 1.00        | 1           | 3            |
|  UserDefine  | 4.00        | 8           | 24           |
|  ValueList   | 1.00        | 1           | 2            |
| <b>Total</b>  |             |             | <b>203</b>   |
| <b>Average</b>  | <b>3.38</b> | <b>5.60</b> | <b>20.30</b> |

| method ▼  | CogC        | ev(G)       | iv(G)       | v(G)        |
|---|-------------|-------------|-------------|-------------|
|  CoefficientList.equals(Object)            | 33          | 13          | 13          | 19          |
|  Expression.plusCos(StringBuilder, Entry<  | 31          | 1           | 11          | 11          |
|  Expression.plusSin(StringBuilder, Entry<C | 31          | 1           | 11          | 11          |
|  Item.indexJudge(String, UserDefineFunct   | 29          | 1           | 9           | 9           |
|  Item.multiply(HashMap<CoefficientList, I  | 25          | 1           | 16          | 16          |
|  Factor.calculate()                        | 23          | 1           | 14          | 16          |
|  Item.setPoly(String, UserDefineFunction)  | 16          | 1           | 9           | 9           |
| <b>Total</b>  | <b>345</b>  | <b>97</b>   | <b>223</b>  | <b>253</b>  |
| <b>Average</b>  | <b>5.75</b> | <b>1.62</b> | <b>3.72</b> | <b>4.22</b> |

虽然相比于Task2，功能变动很少。笔者在Task3的时间里顺手做了一些优化，效能有小幅改进。

### 4) 互测体验

强测AC

Hack三次，被hack零次。

- 发现bug\_0：sum(i, BigInteger, BigInteger, i)
- 发现bug\_1：错误的误认为表达式为因子，忘记了括号
- 发现bug\_2：Task1残留bug，连续正负号处理错误，不知为何这位作者在前几次task中未被Hack到

### 5) 性能优化

与Task2相同，并无额外优化

## 6) 额外完成任务

无额外目标

## 四、互测

---

笔者在互测中，使用了测评机和对拍机。但由于主要工作并非本人完成，笔者仅仅是做了一些可用性的辅助，故而不再献拙，在此只提供一些Hack技巧。

评测机暴力Hack，大概率是Hack不下来的。假如这位同学有评测机，那么大概率在A房，地毯式轰炸并不一定能保证效果。对于高手互博，往往功能本身不会出错，只有想怪数据，反常识数据，才可克敌制胜。

既然如此，看代码就是必不可少的。和刀我的同学交流，发现他也是通过看代码才刀到。

据我不完全观察，防御力较低的代码遵循以下规律。大家在日后的互测中也可以针对以下部分着重分析。

- 优化多
- 字符串处理零碎
- 父子类之间很少调用方法
- 重复代码过多

## 五、心得感想

---

—(坐夫牢)—

第一单元的OO课程，可以说相当硬核，我学到了许多面向对象编程的方法和思维，更加熟练地掌握了Java语言。有幸三次强测完全没有bug，互测也只被刀一次，刀人无数。当然，这一单元的训练也让我认识到了自己的许多不足，比如思路不清晰，在Task1到Task2中血泪重构。希望下一单元电梯，可以吸取教训，更上一层楼。