一、基于度量来分析自己的程序结构

第一次作业：

C语言部分：

由于很久没有写C语言代码，写起来还稍有生疏，在字符串处理获取多项式信息的功能中花费了很多时间，这也与后面Java中的字符串对比形成了鲜明的对比。

体会：代码必须经常写，要不然会手生的厉害！

Java部分：

之前，接触过一点点Java基本的实现一些经典算法还是能实现的，在菜鸟教程上提前做了一点小小的预习，谈不上了解，但是大概知道讲的是什么了。上课的时候，纪导讲的也挺清楚详细。不过真正上手实践还是不一样，花了不少时间思考如何“面向对象”实现功能，反而具体实现功能的时间并不多。构架参照了老师的结构，同时也添加了一些新的东西。但是由于对于Java这门语言的确不了解，其实是做了很多无用功的，比如正则表达式的使用并不熟练，反而在我拿到的互测作业中获取了不少知识。例如Pattern和Matcher的使用，可以利用Matcher.find和Macher.group获取匹配的字符串，减少字符串处理的复杂程度。另外关于正则爆栈也应该是提前想到的事情。

体会：enum很好用很直观！结构设计很重要！要理解Java中的类变量的本质是指针！

第二次作业：

第二次作业就已经进入了面向对象，比较直观的感受就是指导书的复杂。大概用来理解笑话指导书所想要表达的意思的过程占据了整个作业周期的1/3。最后也是在和几位同学的交流讨论中慢慢的理解了要求。不过在理解了要求之后，有了第一次作业的一边摸索一边写作业，这一次的写起来思路还是清晰的。

第三次作业：

不得不说第三次作业对于理解能力的要求进一步提高。我知道周一晚上才能想清楚大概的思路，周一周二连着两个晚上熬夜才最终解决。另外不得不说这一次的测试数据起到了至关重要的作用，同学的测试数据帮我找到了至关重要的BUG。但是这次的作业的复杂性让我的代码复杂度骤然攀升。有很多地方出现了冗余不说，为了实现一些feature不得不设计了一些很丑的代码，甚至到最后我自己都不想看自己的代码。下手前的设计步骤的重要性不言而喻。

在这次作业中，由于必须同时执行多条指令按照输入顺序的要求不得已在Elevator类中设置了缓冲队列，根据后面的输入才决定执行的情况。

二、分析自己程序的bug

第一次和第三次中得益于提前进行了大量的本地测试均未被扣分。第二次中发生了一些小的失误，关于空行（只用空格或直接回车）的处理我的第一版中进行的处理是当作非法输入进行报错，后由于测评机不能识别前导空行，将设计改为忽略空行，但是对于readme的更改虽然上传至gitlab，但是没有即时拉去到OJ上导致了readme和实际行为的不一致。不过后来经测试者提醒，空行作为错误输入似乎是被规定为无效输入了（不过我是没有看到，这也体现了看消息的重要性。

对于课下的bug最重要的还是理解清楚题目要求，同时考虑清楚边界条件。理解错题目的意思，很可能给后期改代码带来巨大的麻烦。而边界条件则是任何时候我们都必须注意\_的事。

三、分析自己发现别人程序bug所采用的策略

第一次作业，由于该同学的输出格式不对，公测过的点数不多，不过由于公测不强，关于计算，我还是找到了不少bug，比较直接的错误原因还是边界情况考虑得不够周全。但是由于对于OJ使用的不熟练，我有两个bug其实是给错了的，寄希望于被测试者申诉来取消bug，但是可惜被测者最终未进行申诉。借此也希望后台学长们尽快加入撤回bug的功能。

第二次的同学作业无懈可击，我还其中获得了一些启发。

第三次作业的同学也十分优秀，算是我见过的最“面向对象”的代码，但是我再阅读readme时发现了他的代码和readme存在行为不一致，因此提出了bug。

策略：

1. 我自己的数据测试一遍，对于大佬基本无效。
2. 根据readme说明测试边界数据，有时候有效。
3. 阅读代码构造bug，更多时候读不懂，但是能学习到很多。

四、心得体会：

1）学会用工具debug还是很重要的，现在的代码规模已经不像是过去c语言和数据结果的代码量了，不能高效的使用debug工具非常影响效率。

2）大学的各种考试包括作业都有信息战的趋势，构造或拿到一份好的测试集往往就能救人于水火之中，能不能关注到Issues的信息也是成败的关键因素。指导书的更改和通知群的消息都是十分重要的。磨刀不误砍柴工，有时候先进行足够的思考，看完所有的Issue和群里的问题再下手，可以减少写代码的时候要求不明确时带来的反复修改，也降低debug的难度。

3）还是要严谨。Readme之于程序就像说明书之于电器，如果readme和电器实际的操作不一致，顾客是不能接受的。不管程序还是readme都应该尽量做到完备，而不能应付了事。

4）git其实还没有真正利用起来，以后应该加强对于git的学习和运用。

5）现在的代码其实还远远不够简洁，很多由于需求增加而临时设置的变量或者其他冗余内容。模块化做的也不好，应该找时间进行代码简化甚至代码重构。

6）在与大佬的代码进行了比较之后，发现自己在代码规范还有差距，可读性应该进一步提高。同时对于拿到好的代码不光要查bug，更要吸收借鉴，拿来己用。