编译原理实践项目报告

学号： 10215101402

姓名： 徐宸

1. 项目亮点参考
   1. 代码量和规范写法

一共分为四大部分，其中，词法分析占据740行，LL语法分析占据1084行，LR语法分析占887行（实现了LR(0)与LR(1)两种情况）。

本项目采用Java进行编写，严格遵守面向对象的语法，并且对各种parser进行了封装，并在后续进行复用，比如LR parser中，使用了LL parser的内容，并且代码风格好，命名遵循驼峰规范，注释齐全。

* 1. 理论课算法的自动化实现

此处，从LL parser到LR parser，我们自动化实现了从文法到计算first集、follow集以及建立LL分析表与LR分析表的部分，占据了很大的篇幅，在实现自动化算法的过程中，我对课上的内容有了进一步的深入了解，并且，可以使得我的算法能够适应多种文法，只需要修改输入的文法即可，代码复用率明显变高，并且，之前的代码可以在后续所延用，也使得我的开发效率随着项目进度的推进越来越快。

* 1. 特殊的符号表存储方式
  2. 额外的错误处理机制

对于LL parser和LR parser来说，我能够自动处理大部分缺少符号的情况，对于符号输入错误的情况，能够进行报错，并且退出，同时提示用户在某行发生错误，发生在某个符号附近，和我常用的一些编译器已经较为类似了。错误提示统一格式，对于报错，我抛出异常，并且通过打印在控制台的方式来告知用户。

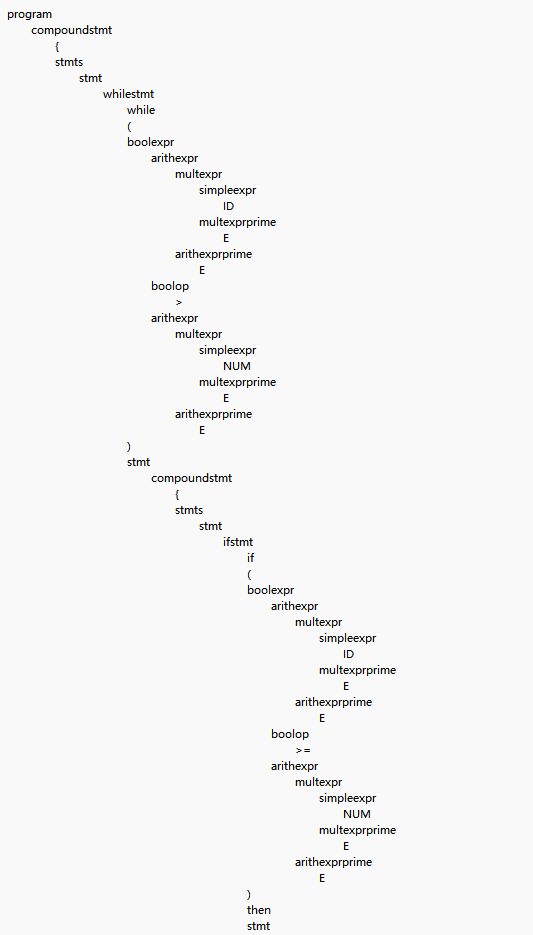
1. 是否使用更多的测试

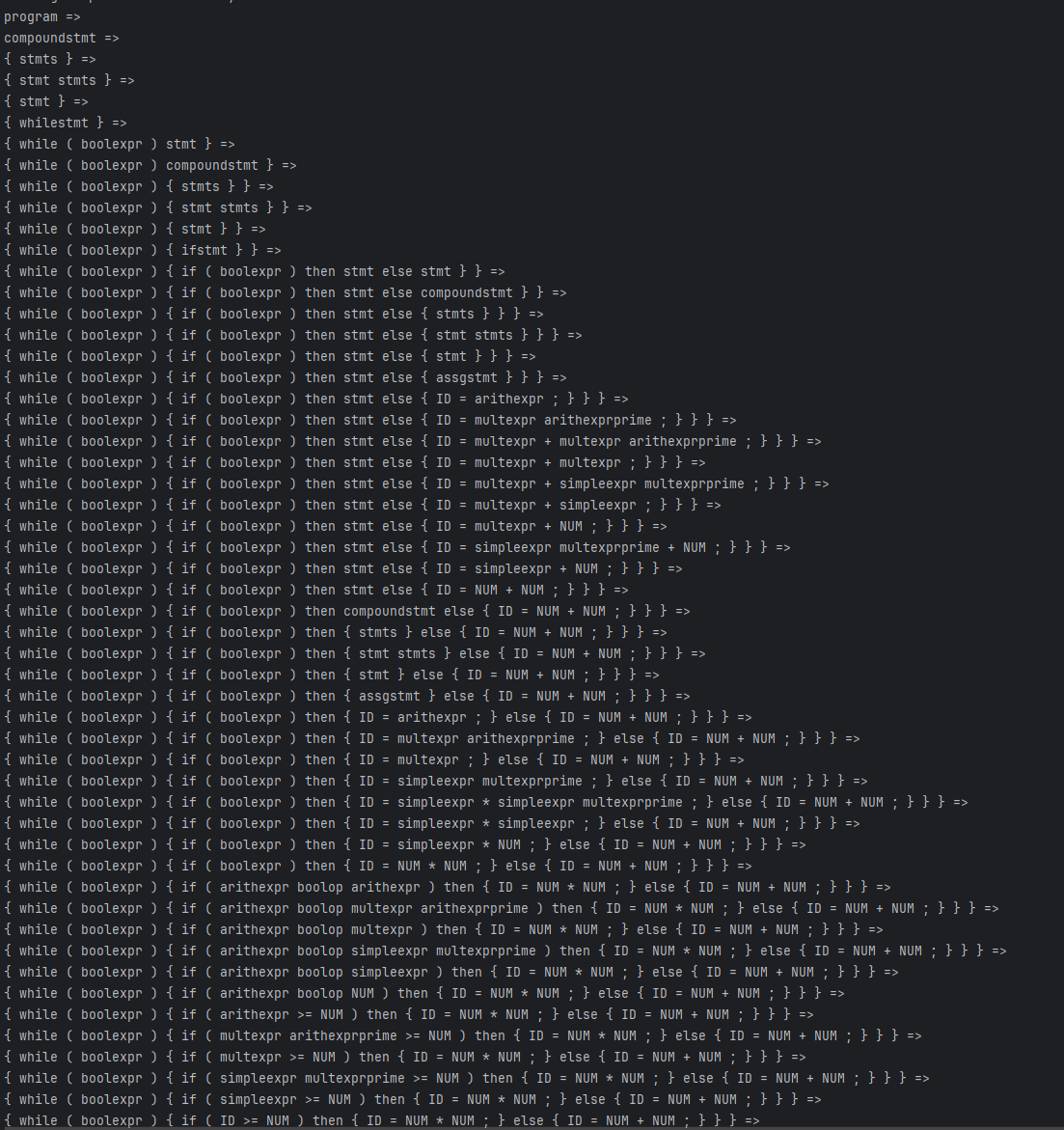
在测试过程中，我进行了更多测试用例的使用，其中大部分为合理且能编译的测试用例，以及部分测试错误处理机制的测试用例，一共在75个测试用例上进行测试，接下来我将挑选一组具有代表性的进行讲解：

正确测试用例：

1. {
2. **while** (ID > NUM)
3. {
4. **if** (ID >= NUM) then
5. {
6. ID = NUM \* NUM ;
7. }
8. **else**
9. {
10. ID = NUM + NUM ;
11. }
12. }
13. }

这是一个比正常测试用例复杂的循环if嵌套测试用例，我的LL parser和LR parser都在其上进行过测试，分别结果如下：

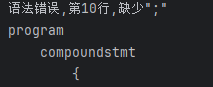




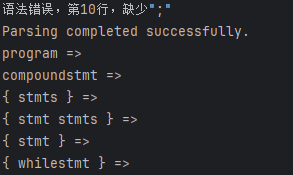
由于实在太大，都只截取部分，后续我们的操作都在这个测试用例的基础上修改，分别如下：

* 缺少分号（第十行）

1. {
2. **while** (ID > NUM)
3. {
4. **if** (ID >= NUM) then
5. {
6. ID = NUM \* NUM ;
7. }
8. **else**
9. {
10. ID = NUM + NUM
11. }
12. }
13. }



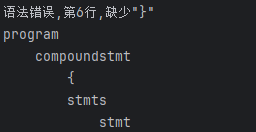
LL parser遇到缺少分号会提示并自动帮忙补齐



LR parser也会提示并帮忙补齐

* 缺少右括号

1. {
2. **while** (ID > NUM)
3. {
4. **if** (ID >= NUM) then
5. {
6. ID = NUM \* NUM ;
7. **else**
8. {
9. ID = NUM + NUM
10. }
11. }
12. }



LL parser遇到缺少右括号会提示并自动帮忙补齐



由于时间问题，目前LR parser暂未做帮忙补齐的部分，目前会提示报错的地点

* 输入错误单词/单词多余

1. {
2. **while** (ID > NUM)
3. {
4. **if** (ID >= NUM) then
5. {
6. ID = NUM \* NUM ;
7. els
8. {
9. ID = NUM + NUM
10. }
11. }
12. }



LR parser会指出错误并退出



LL parser也会同样指出错误

通过这样的测试用例的不断测试，我能够保障我的代码能够正确运行，并且不断完善，由于时间问题，目前暂时完善结果如上，自动修复的部分还没完全完成，不过后续应该能够更好地进行完善和修改，目前已经发布github仓库。