实验目的

通过自己编写、调试一个语法分析程序，对 token 序列进行语法分析。对于输入的每一个 token 进行查表分析，进而获得每一步的推导产生式。

内容描述

本程序是进行一个对 token 序列的语法分析。在程序里对产生式进行消除左递归，求非终结符的 first 和 follow，生成 LL（1）文法的预测分析表。生成预测分析表后，对输入的 token 序列进行查表分析。

思路方法

本人使用第三种实现方法：Generating programs based on your own YACC。

1. 首先读入产生式文件，获得产生式；

2. 查询产生式中是否存在直接左递归，使产生式改变成不含直接左递归的；

3. 将无直接左递归和无二义性的产生式生成 PPT（预测分析表）。中间穿插了各个非终结符的 first 和 follow 求算。

4. 读取输入的 token 序列，进行查表分析。我这里构造的使 LL（1）文法的预测分析表。故只需要一个推导过程中的符号栈即可。中间 token 序列分析出错会有错误提示。

5.将查表过程中使用到的推导式写入到结果文件中；如果是错误的 token 序列，将错误信息写入结果文件中。（控制台也会打印相关信息）。

假设

假设输入的产生式都是无二义性的。

相关 FA

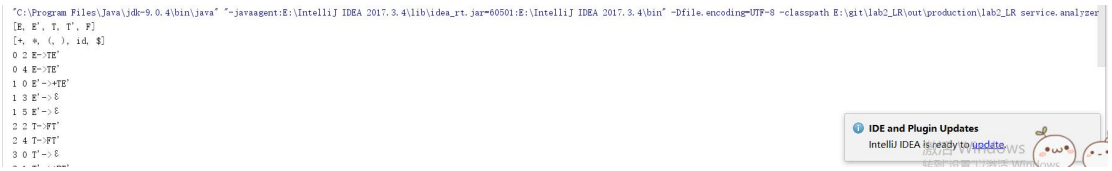
预测分析表：

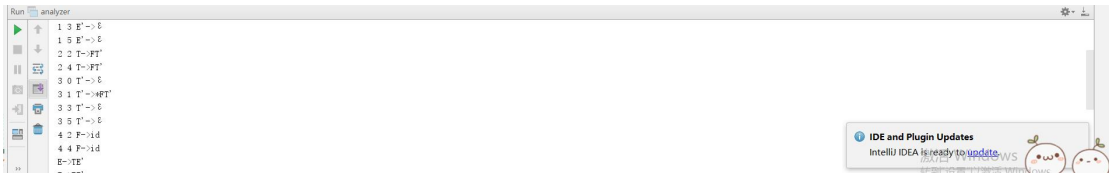
	+	*	（	）	id	$
E			E ->TE’		E ->TE’	
E’	E’->+T E			E’-> ε		E’-> ε
T			T->FT’		T->FT’	
T’	T’-> ε	T’->*F T’		T’-> ε		T’-> ε
F			F->id		F->id	

程序中分析产生式后，得到的预测分析表打印如下：

前两行分别是对应非终结符和终结符集合；

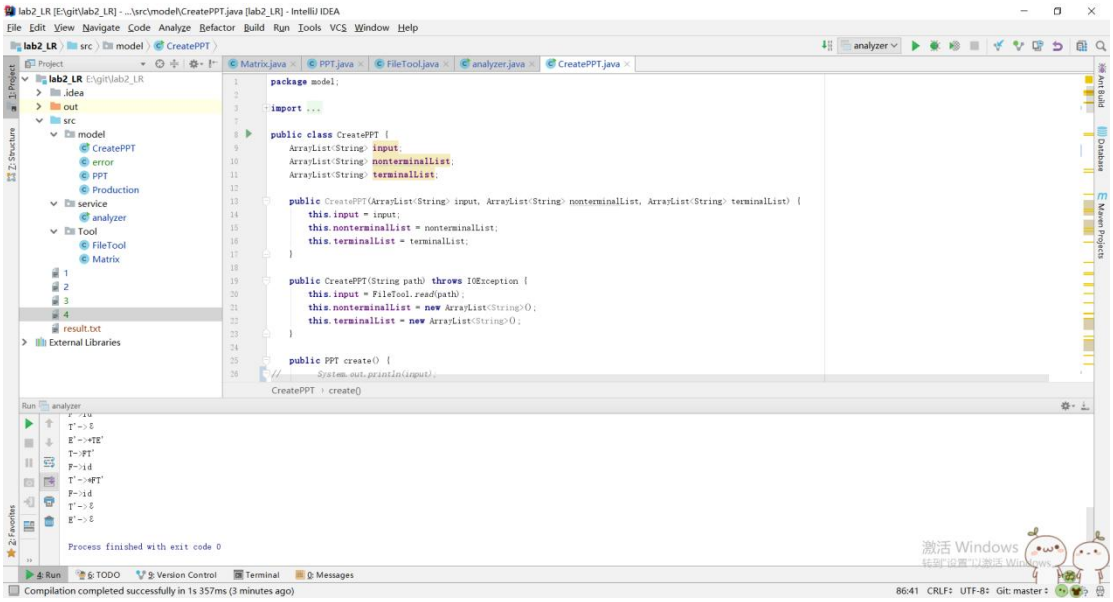
后面的每行，前面两个数字对应产生式对应在预测分析表矩阵中的位置。与上面的表格完全对应。



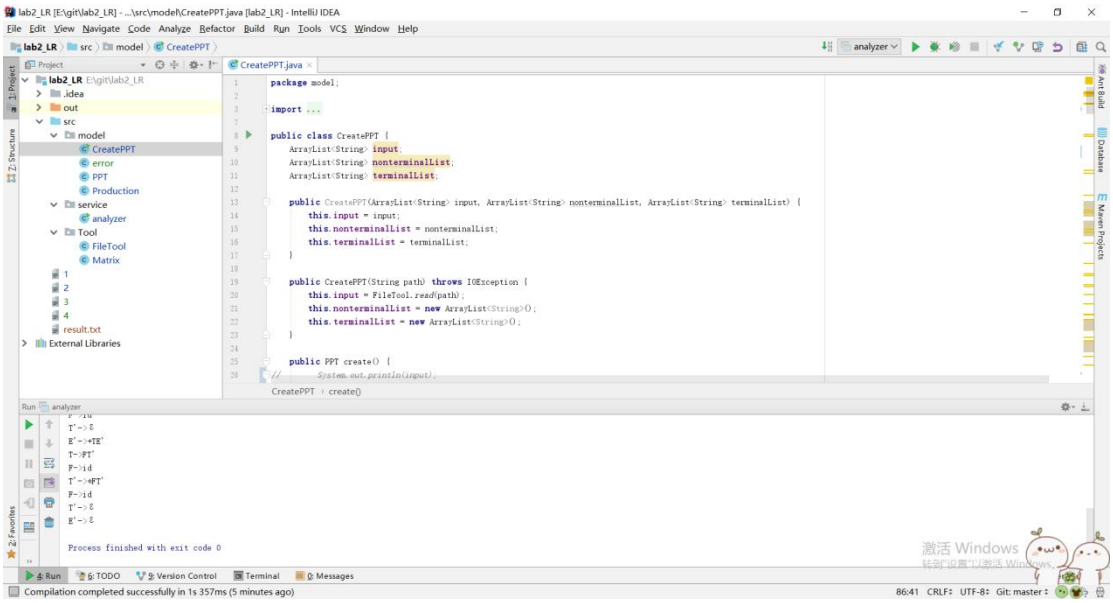


重要数据结构描述

总共有 7 个类。



CreatePPT 类：



这个类的主要功能是根据输入的产生式来生成预测分析表。

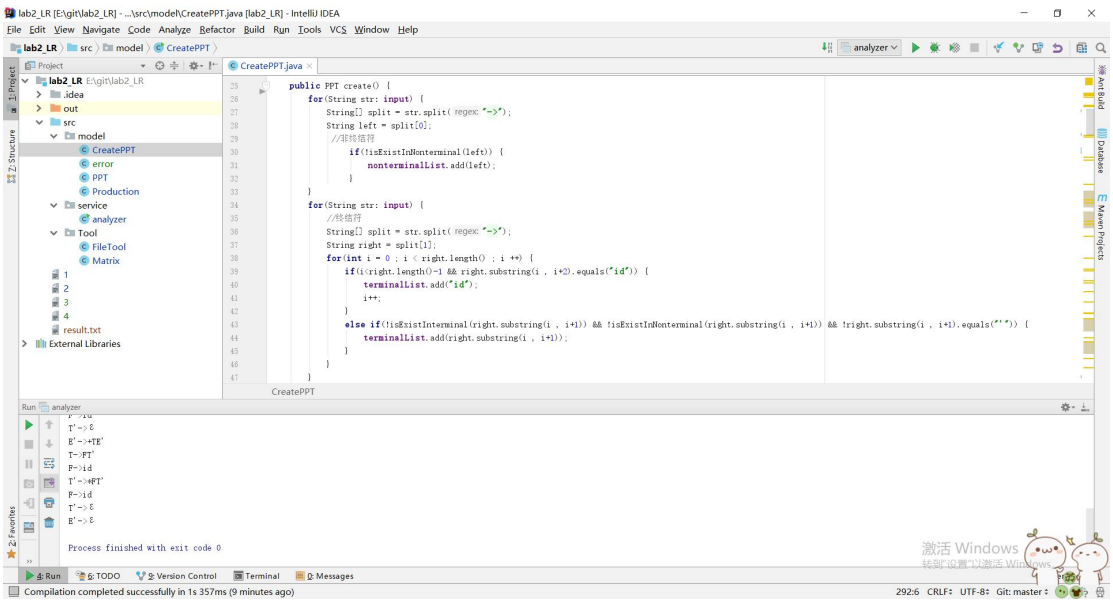
类属性：input 是输入的产生式集合；

nonterminalList 是非终结符元素的集合。

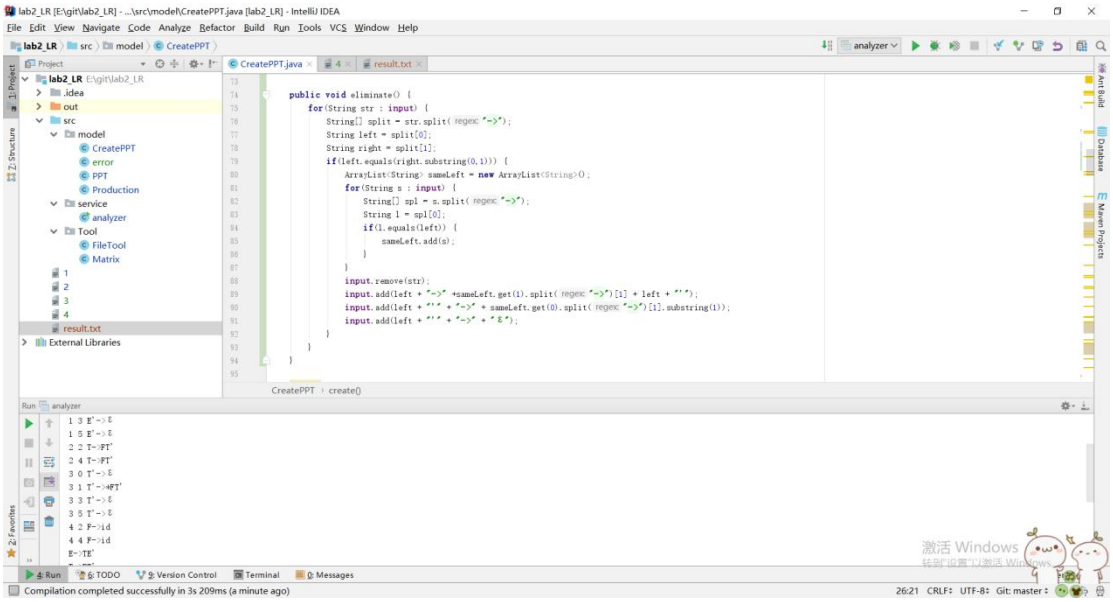
terminalList 是终结符元素的集合。

主要方法说明：

create 方法：是产生 ppt 预测分析表。



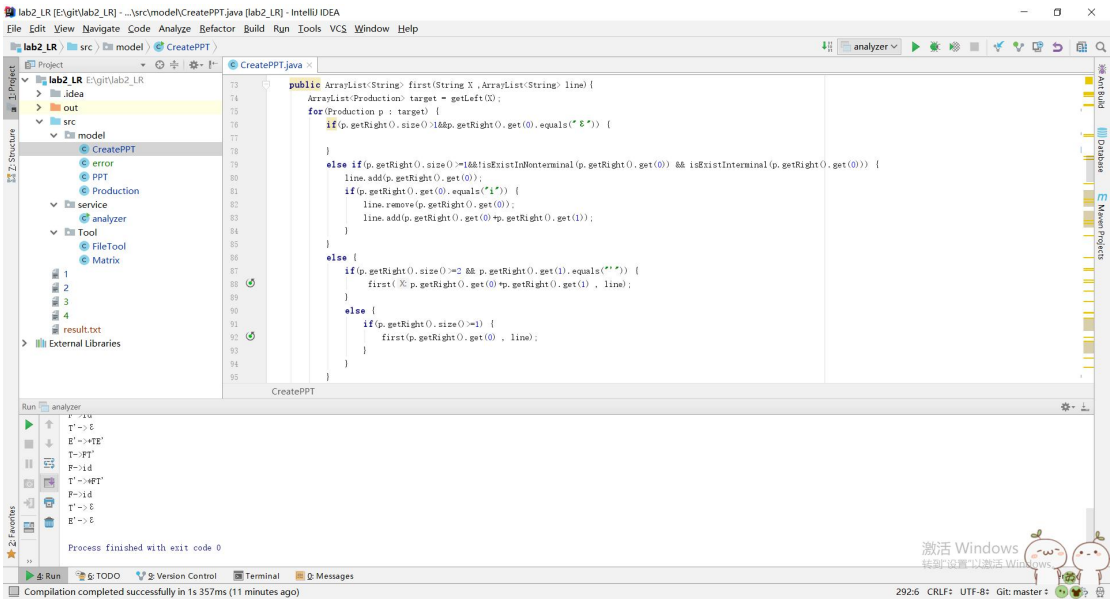
预处理 eliminate 方法：消除左递归。将含有直接左递归的产生式拆解成不含直接左递归。



first 方法：计算各个非终结符的 first 值，为生成 ppt 作准备；

采用了递归。

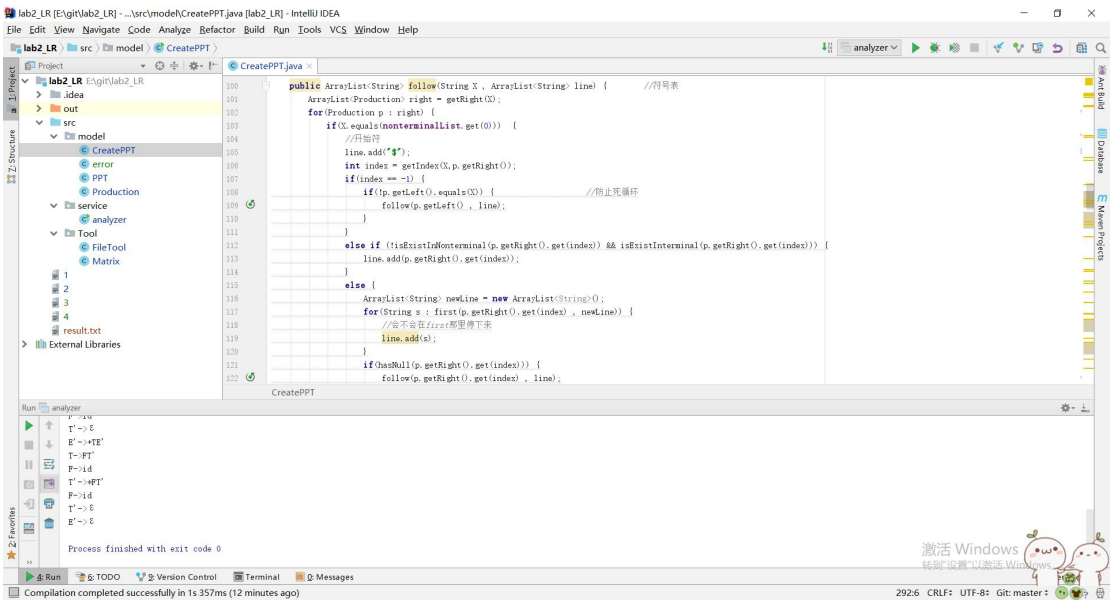
参数是非终结符和存取值的 list。



follow 方法：计算各个非终结符的 follow 值，为生成 ppt 作准备；

采用了递归以及调用了 first 函数。

参数是非终结符和存取值的 list。

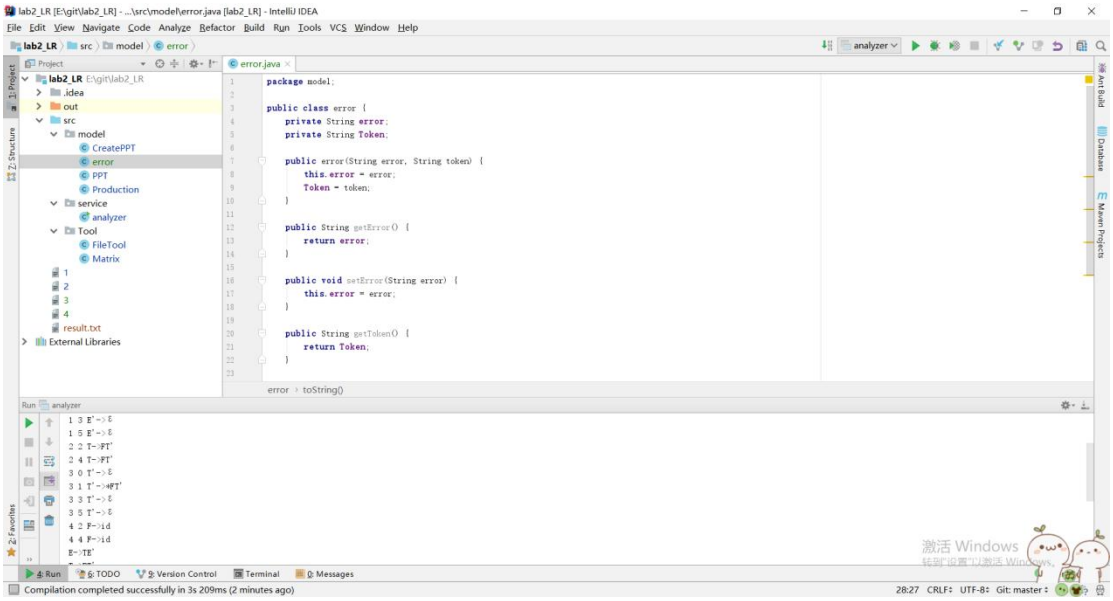


其他两百行左右的代码是为了辅助主要方法而写的辅助方法。

error 类：

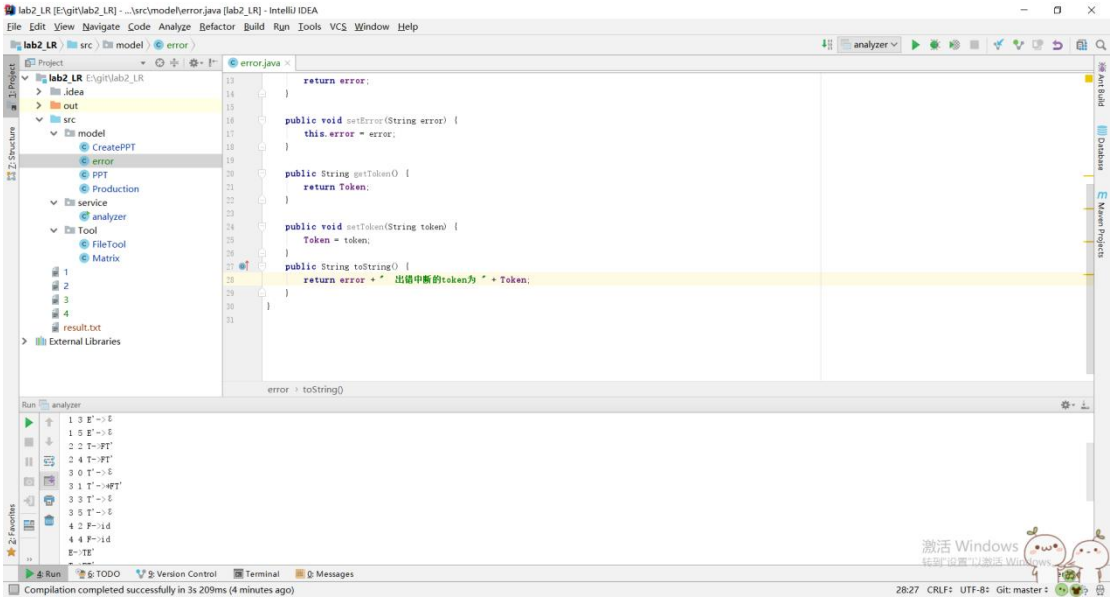
类属性：error　报错提示

Token　引起语法分析中断的 token



主要方法分析：

重写 toString 的方法，将 error 转为字符串。

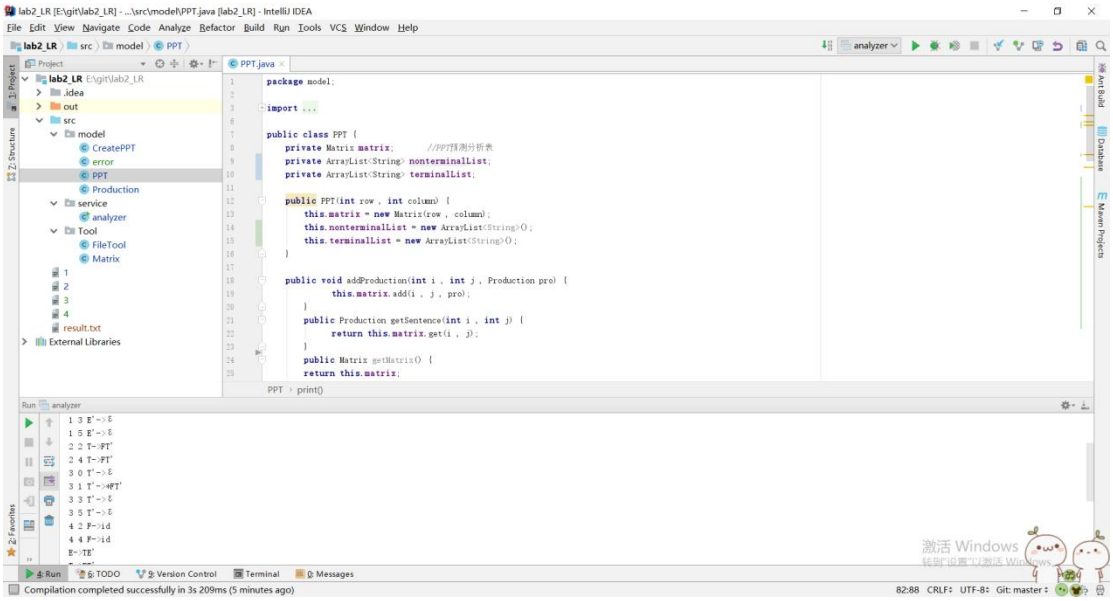


PPT 类

类属性：matrix 存放预测分析表数据的矩阵；

nonterminalList 非终结符集合

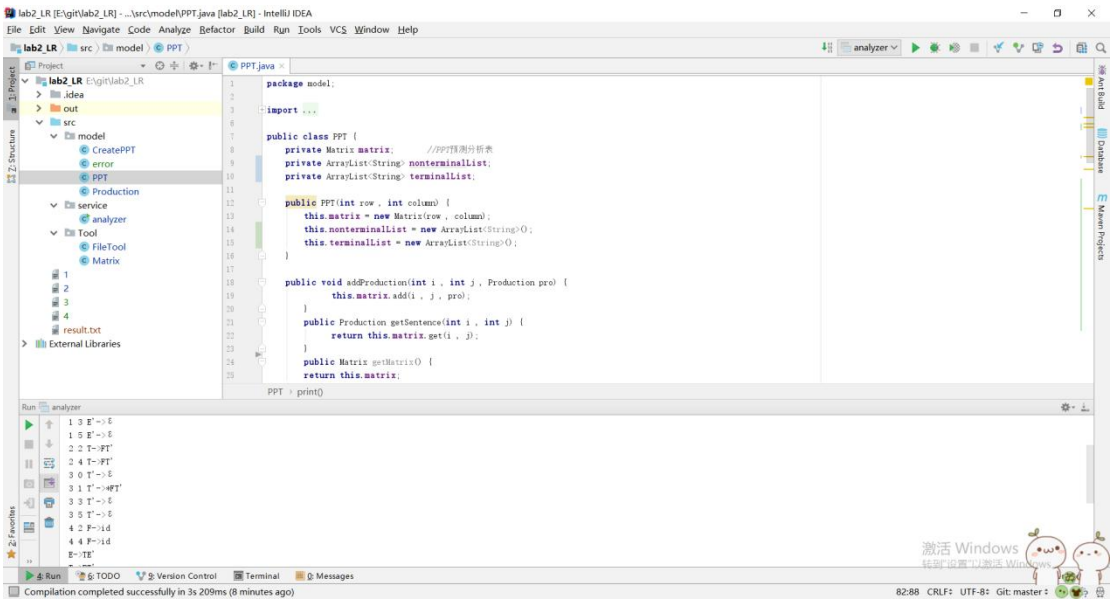
terminalList 终结符集合



主要方法：

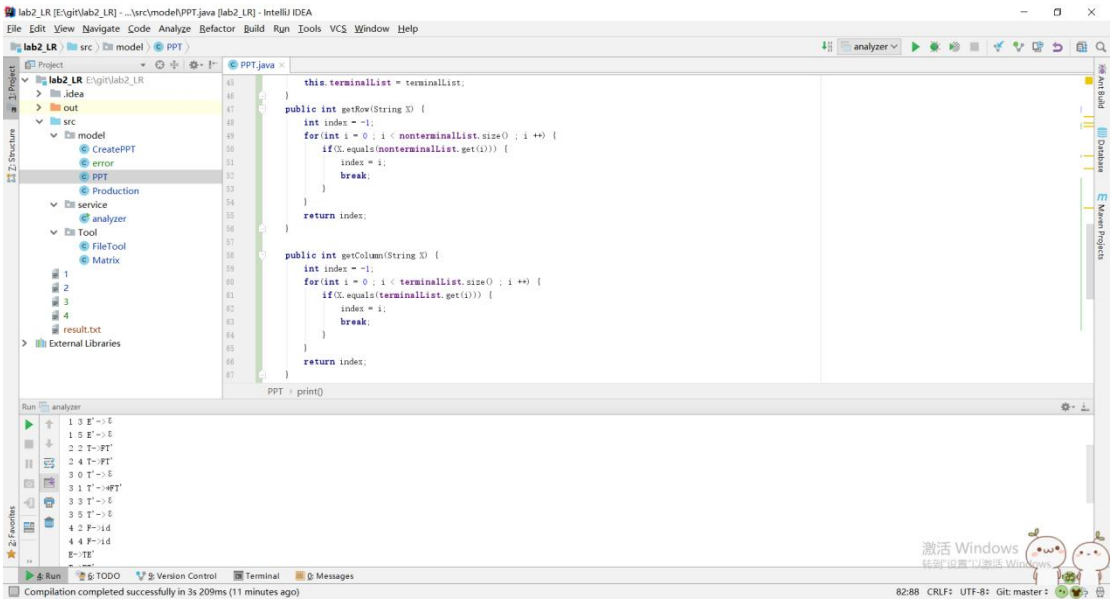
（鉴于方法比较简单，简单描述一下。）

addProduction ： 向语法分析表中加入产生式



getRow：获取对应非终结符在预测分析表中的行。

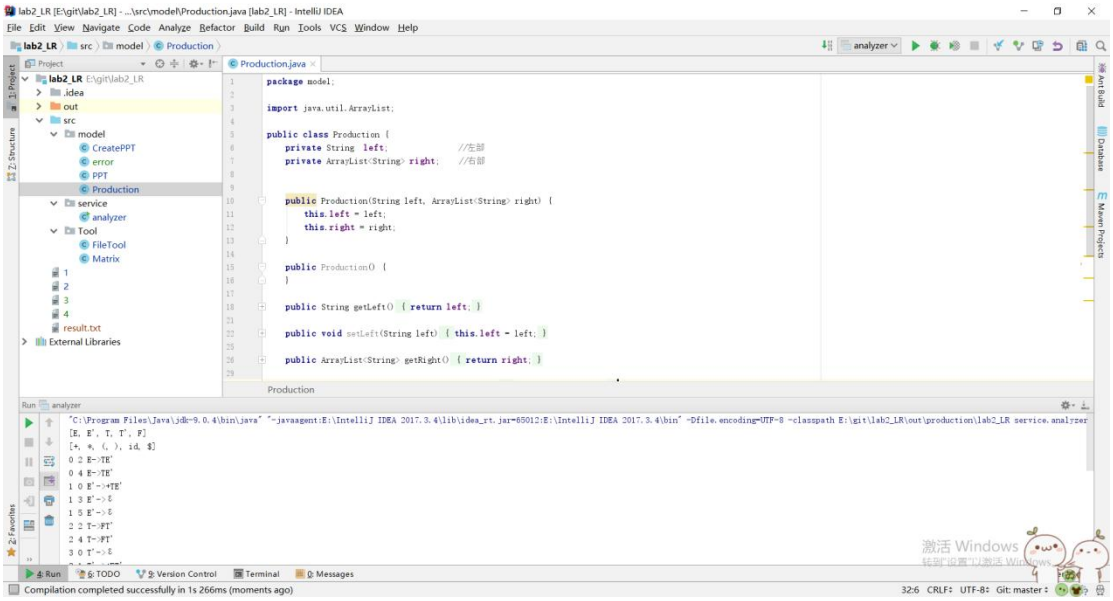
getColumn：获取对应终结符在预测分析表中的列。



Production 类：

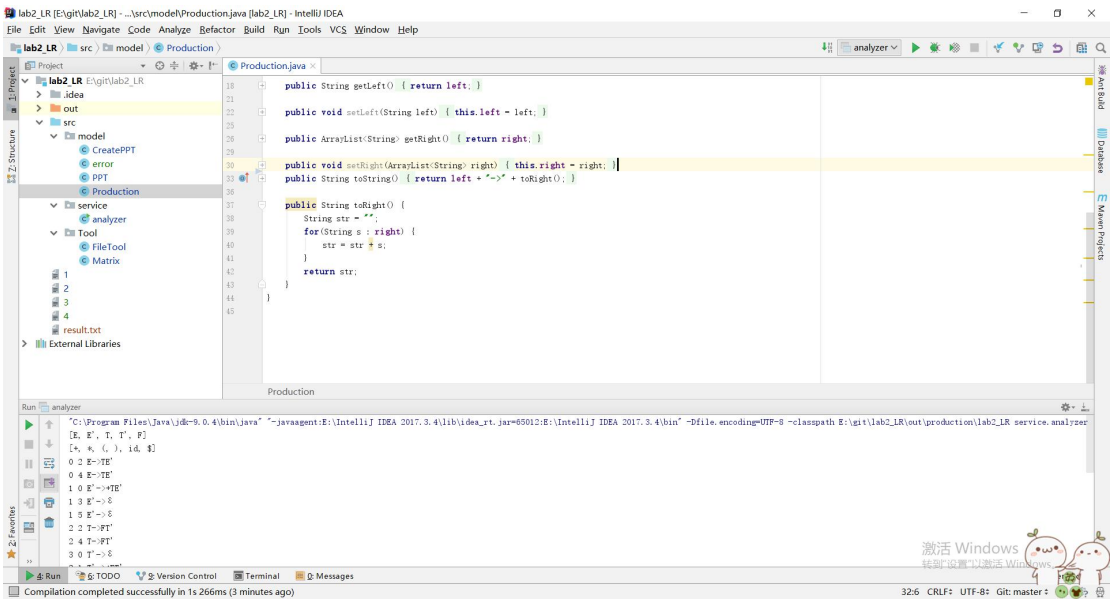
类属性：left 左部。

Right 右部。



主要方法：

toString 重写 toString 方法，将 Production 转为 String 类型。内部实现调用了 toRight 方法（将右部转为 String）。



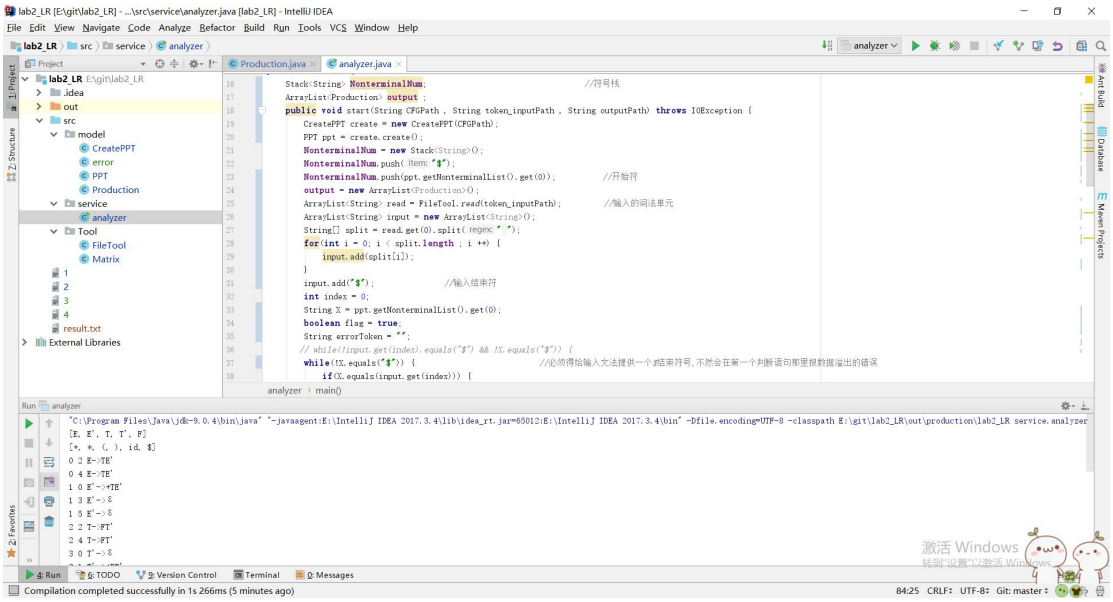
analyzer 类 （语法分析类）

类属性：NonterminalNum 符号栈。

Output 输出的推导式。

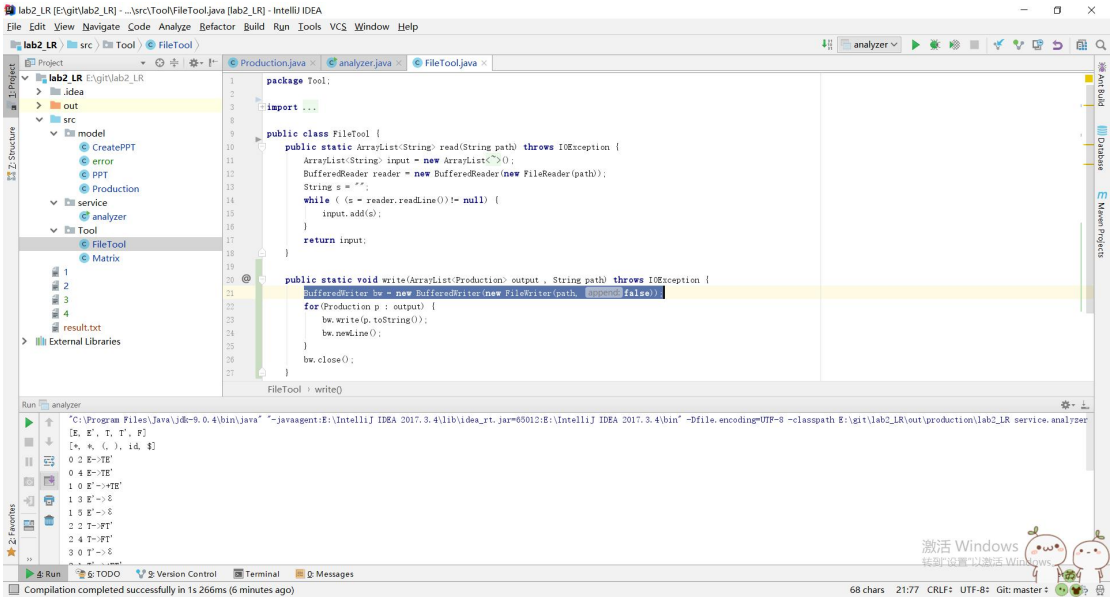
主要方法：start 函数

参数分别为 CFG 的文件路径，输入的 token 序列文件路径，以及输出文件的路径。



FileTool 类

主要功能是文件的读和写

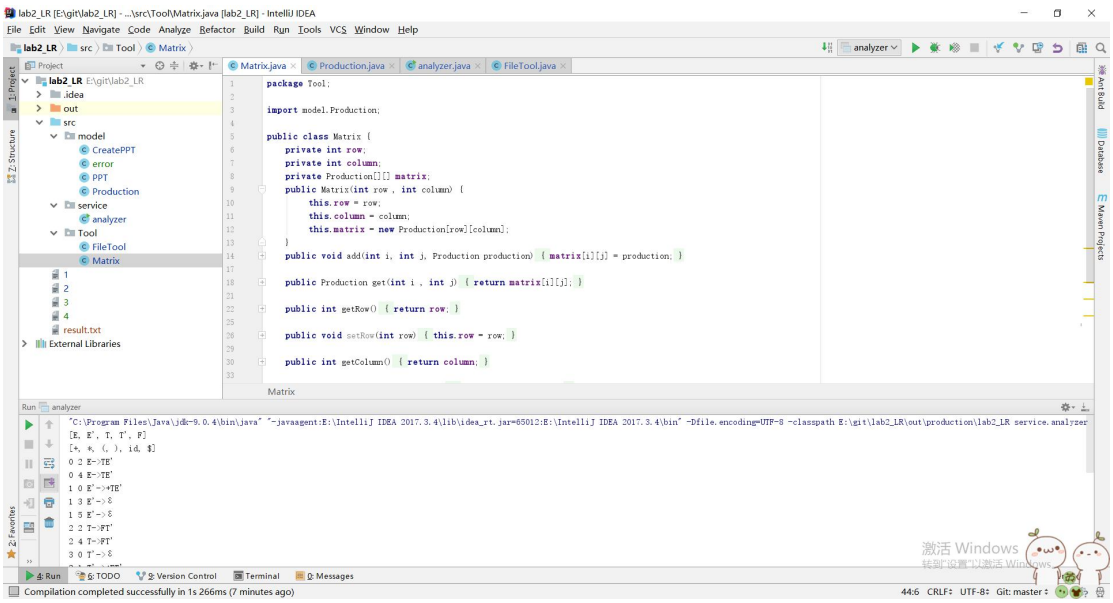


Matrix 类：

矩阵类：辅助存储预测分析表的工具类

类属性分别是行，列和二维数组

方法就是常见的矩阵读写的方法。

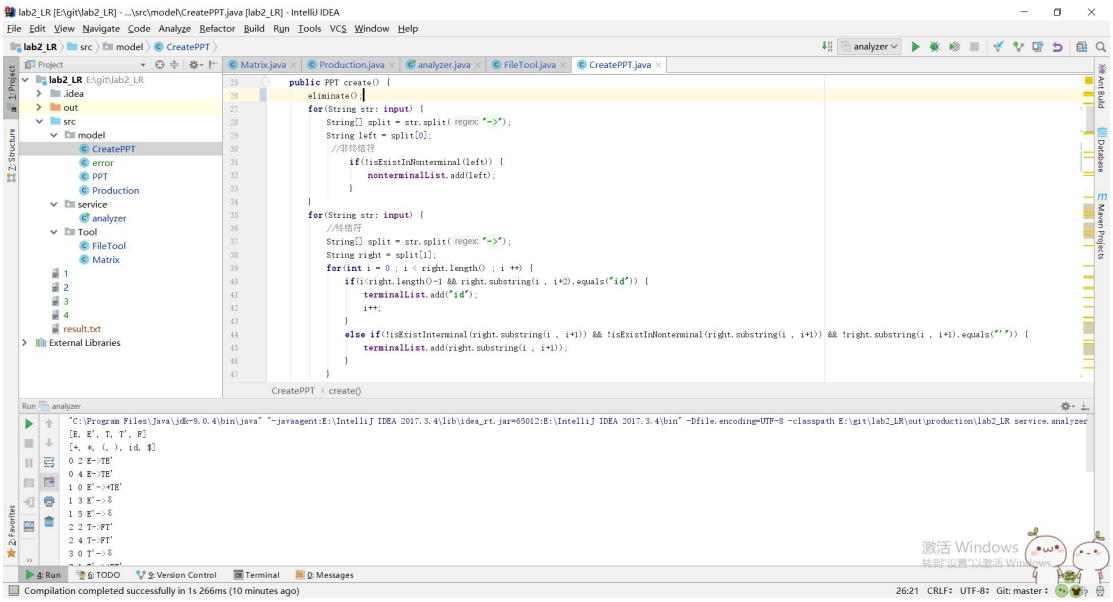


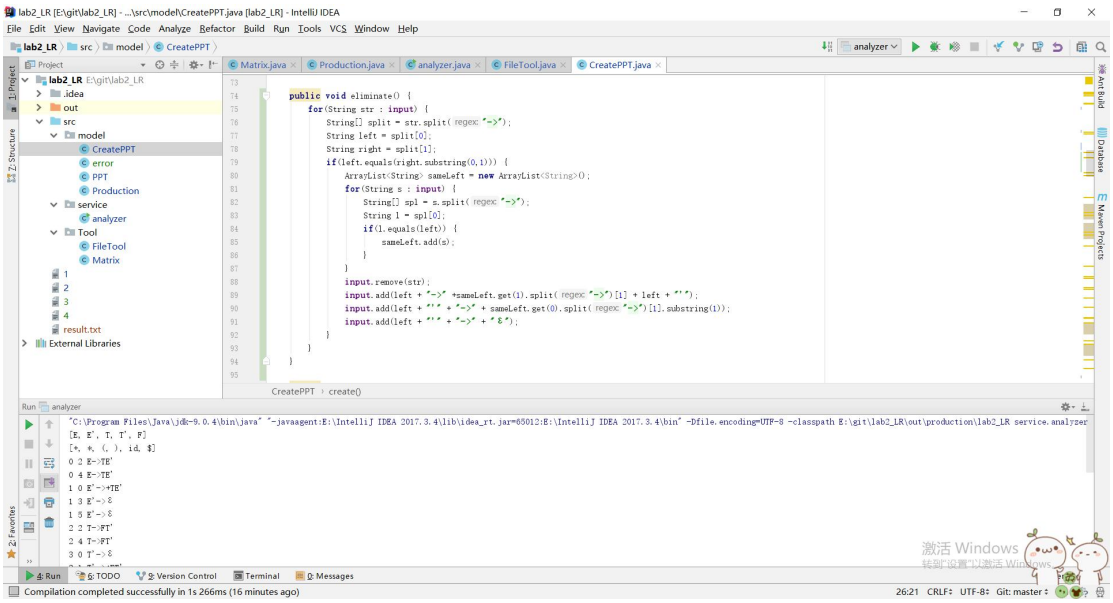
核心算法

主要分为两块。

1. 生成预测分析表 PPT；　（CreatePPT 类里实现）

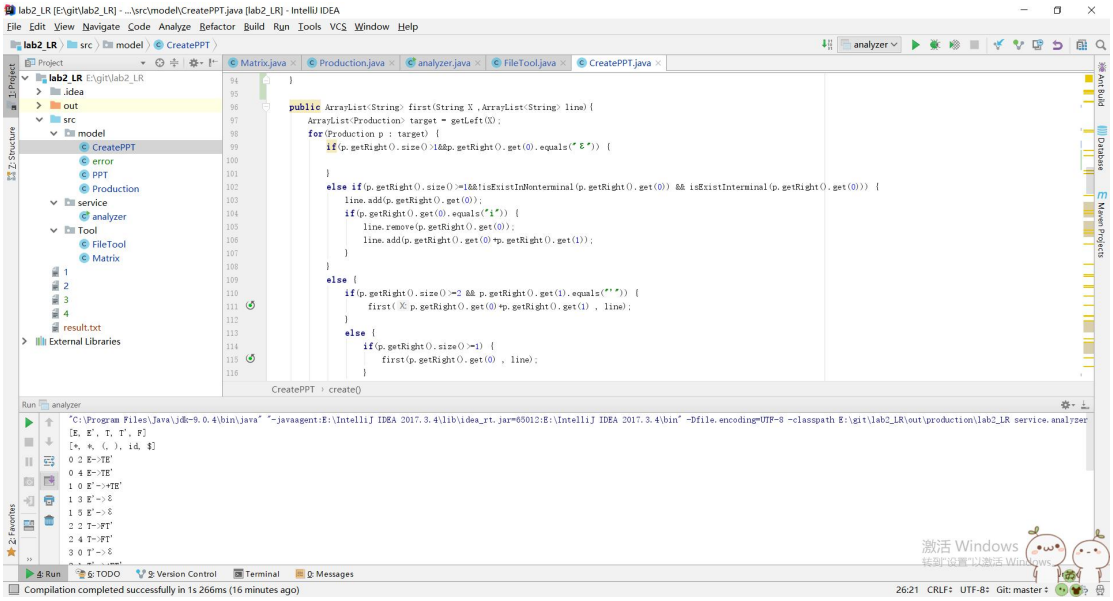
　　按照龙书上讲的步骤，首先我们得对这些产生式进行预处理；于是我先调用了 eliminate 方法来对得到的 input 进行直接左递归的消除处理。大致思想就是先遍历全部 input，对于左部和右部首个符号相同的产生式，即为直接左递归。找到直接左递归的产生式后，得到与之左部相同的所有产生式。通过添加新的左部，将直接左递归拆解。





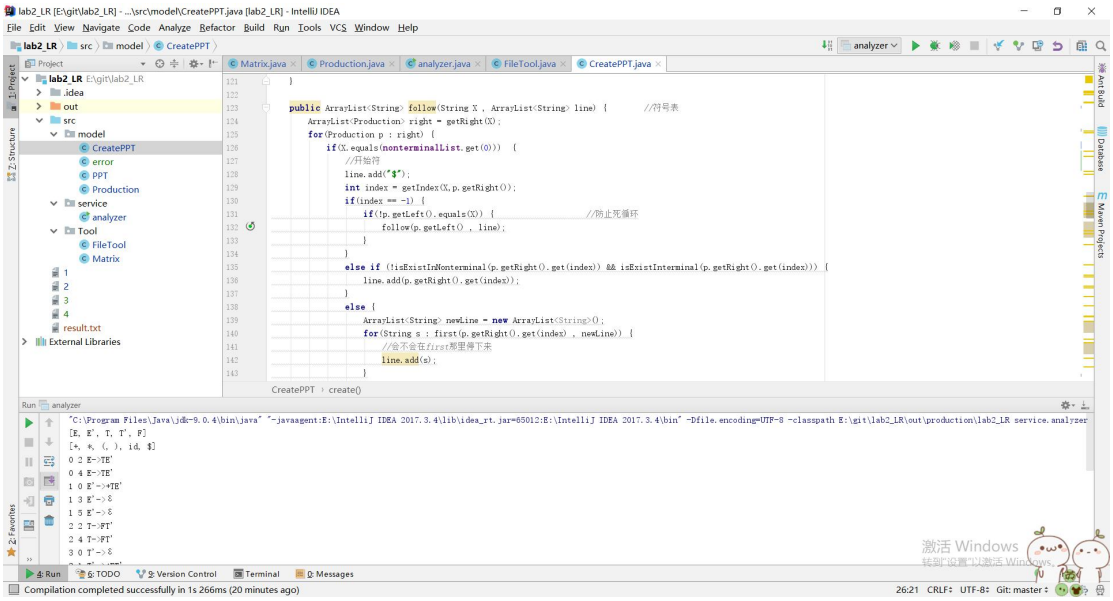
然后我们就是需要求解每一个非终结符的 **first** 和 **follow** 了，对于 **first** 函数，只需要用到递归，就可轻松实现。循环里是该非终结符在产生式左部的所有产生式。

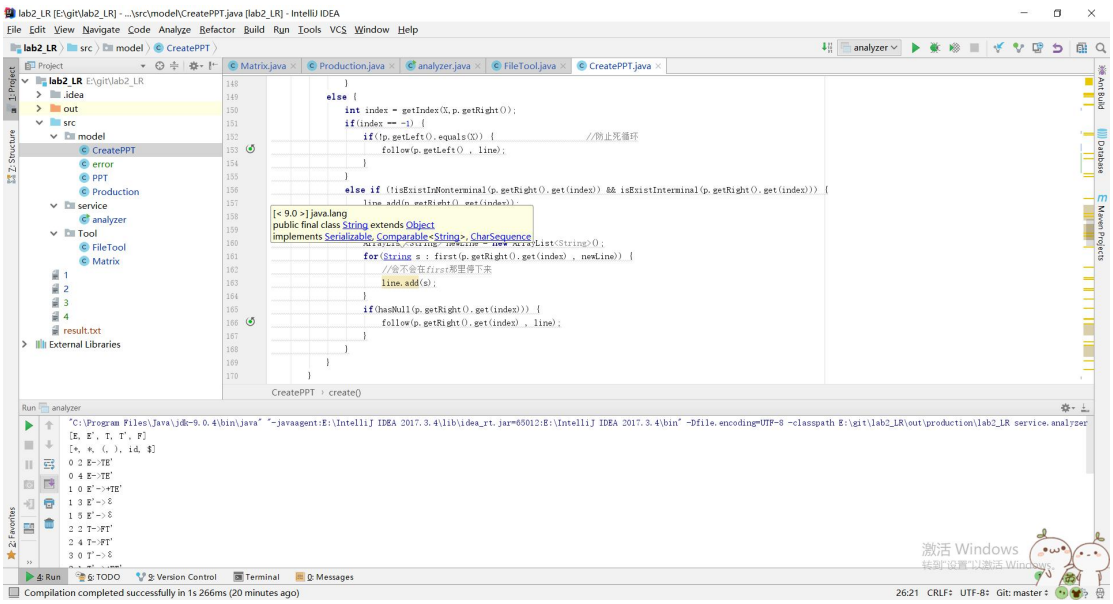
如果右部是空串，跳过；如果右部的首位是终结符，就直接将它添加进去。如果右部是非终结符，就用 **first** 递归。



对于 **follow** 函数，用到递归以及调用 **first** 函数。循环里是该非终结符在产生式右部的产生式。如果是开始符，加入**$**;如果该非终结符

是最后一位，并且左部和它不相同（会死循环），递归 follow。如果紧跟着的是终结符，加入它。如果紧跟着的是非终结符，首先调用 first 函数，将紧跟非终结符的 first 加进去，再检查该非终结符是否会推出空串，如果会的话，则递归调用 follow。





2. 得到预测分析表后进行语法分析；　（analyzer 类里实现）

相比较于生成预测分析表而言，进行语法分析并不是特别复杂。

首先获得输入的 token 序列，并在末尾加上“$”的结束符号。在符号栈里先压入“$”和开始符。这样初始状态就已经好了；

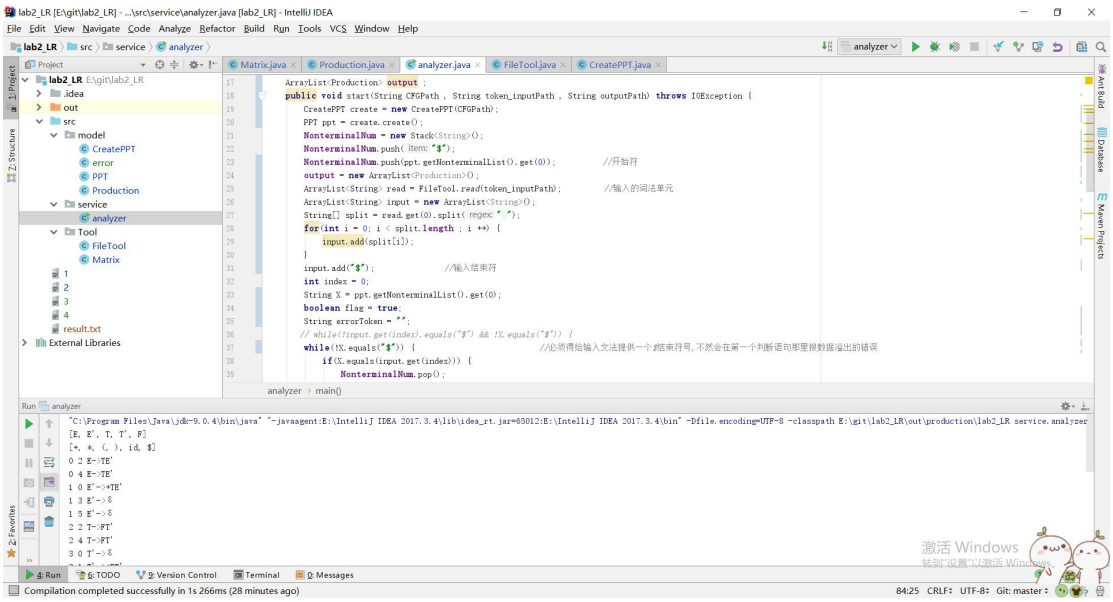
然后当栈非空，如果栈顶符号等于 token 序列所指符号；出栈，token 序列指针后移。

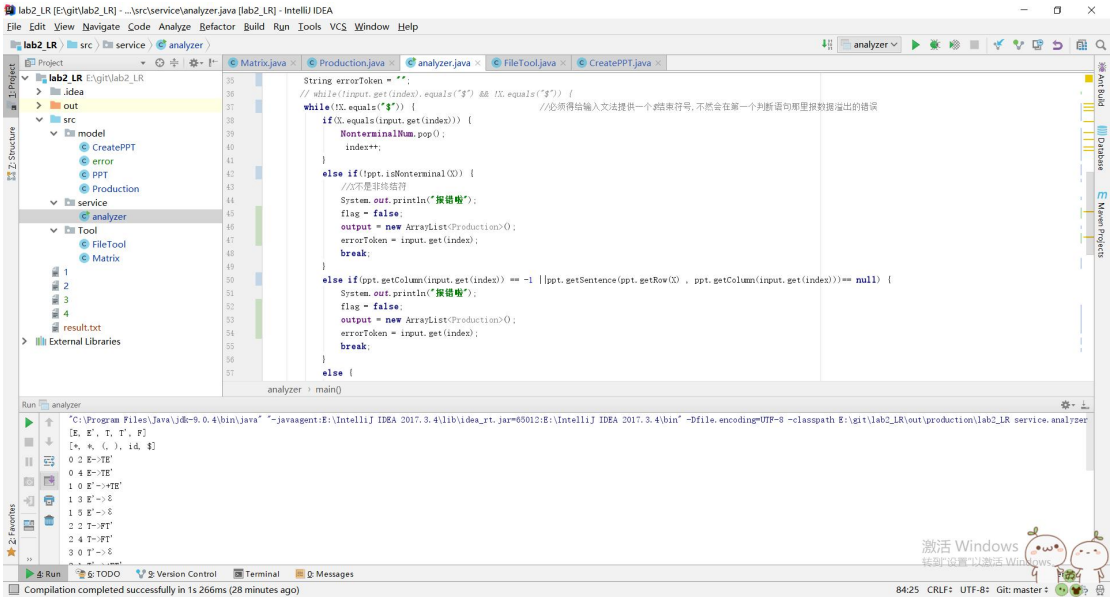
如果栈顶符号是一个终结符（即 token 序列不匹配），报错。

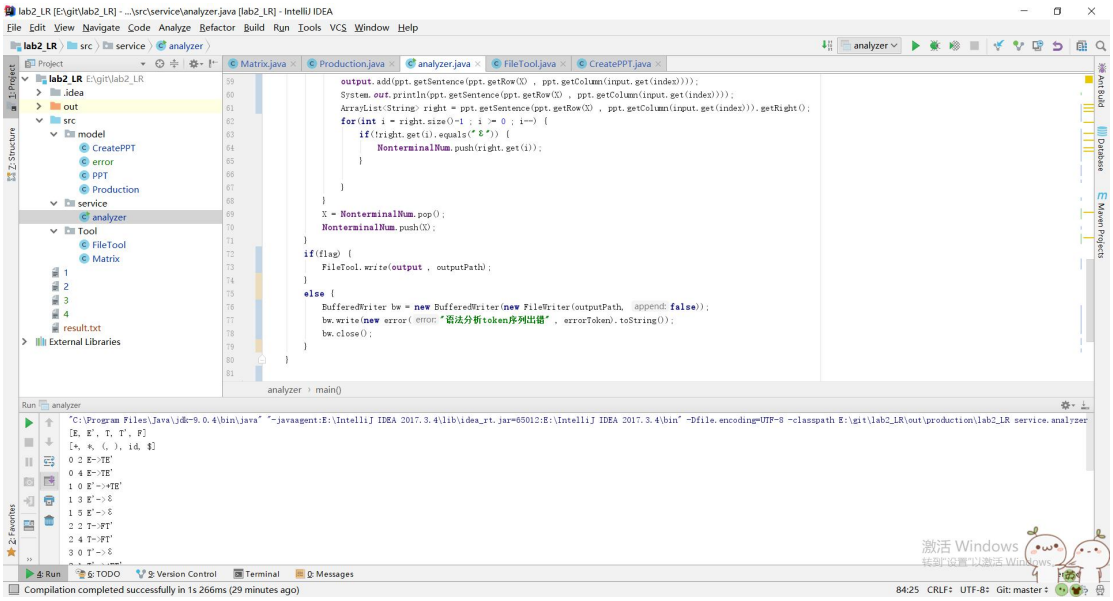
如果预测分析表内对应的产生式为空 ， 报错

如果预测分析表内对应的产生式不为空 ， 弹出栈顶符号，并将推导产生式的右部按照倒序地顺序压栈。

依次循环，直到栈为空。

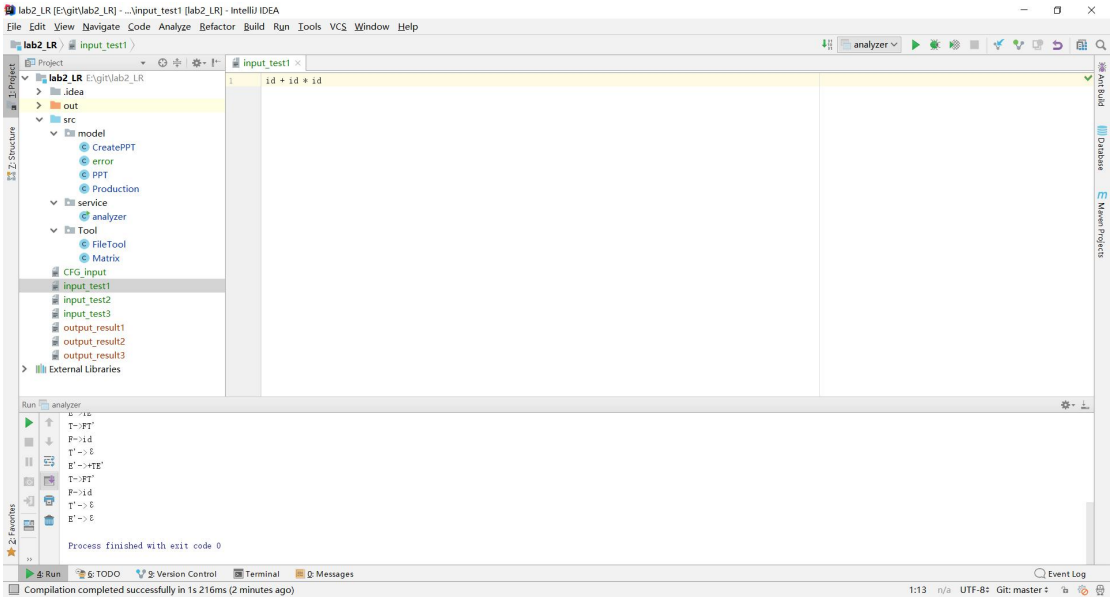



lab2_LR [E:\git\lab2_LR] - ...\src\service\analyzer.java [lab2_LR] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
lab2_LR src service analyzer
Project
lab2_LR E:\git\lab2_LR
.idea
out
src
model
CreatePPT
error
PPT
Production
service
analyzer
Tool
FileTool
Matrix
1
2
3
4
result.txt
External Libraries
Matrix.java
Production.java
analyzer.java
FileTool.java
CreatePPT.java
String errorToken = "";
// while(!input.get(index).equals("$") && !X.equals("$")) {
while(!X.equals("$")) {
//必须得给输入文法提供一个#结束符号,不然会在第一个判断语句那里报数据溢出的错误
if(X.equals(input.get(index))) {
NonterminalNum.pop();
index++;
}
else if(!ppt.isNonterminal(X)) {
//X不是非终结符
System.out.println("报错啦");
flag = false;
output = new ArrayList<Production>();
errorToken = input.get(index);
break;
}
else if(ppt.getColumn(input.get(index)) == -1 ||ppt.getSentence(ppt.getRow(X) , ppt.getColumn(input.get(index)))== null) {
System.out.println("报错啦");
flag = false;
output = new ArrayList<Production>();
errorToken = input.get(index);
break;
}
else {
analyzer main()
Run analyzer
"C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=65012:E:\IntelliJ IDEA 2017.3.4\bin" -Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer
[E, E', T, T', F]
[+, *, (,), id, $]
0 2 E->TE'
0 4 E->TE'
1 0 E'->+TE'
1 3 E'->ε
1 5 E'->ε
2 2 T->FT'
2 4 T->FT'
3 0 T'->ε
4: Run 6: TODO 9: Version Control Terminal 0: Messages
Compilation completed successfully in 1s 266ms (28 minutes ago)
84:25 CRLF: UTF-8: Git: master
激活 Windows
转到"设置"以激活 Windows。

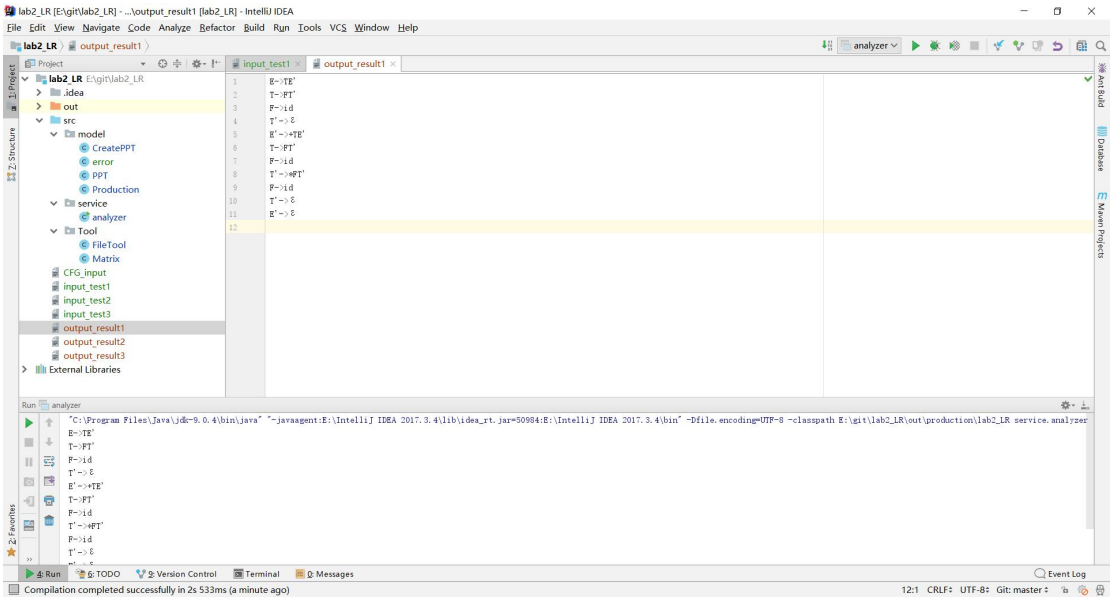

lab2_LR [E:\git\lab2_LR] - ...\src\service\analyzer.java [lab2_LR] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
lab2_LR src service analyzer
Project
lab2_LR E:\git\lab2_LR
.idea
out
src
model
CreatePPT
error
PPT
Production
service
analyzer
Tool
FileTool
Matrix
1
2
3
4
result.txt
External Libraries
Matrix.java
Production.java
analyzer.java
FileTool.java
CreatePPT.java
output.add(ppt.getSentence(ppt.getRow(X) , ppt.getColumn(input.get(index))));
System.out.println(ppt.getSentence(ppt.getRow(X) , ppt.getColumn(input.get(index))));
ArrayList<String> right = ppt.getSentence(ppt.getRow(X) , ppt.getColumn(input.get(index))).getRight();
for(int i = right.size()-1 ; i >= 0 ; i--) {
if(!right.get(i).equals("ε")) {
NonterminalNum.push(right.get(i));
}
}
}
X = NonterminalNum.pop();
NonterminalNum.push(X);
}
if(flag) {
FileTool.write(output , outputPath);
}
else {
BufferedWriter bw = new BufferedWriter(new FileWriter(outputPath, append: false));
bw.write(new error(error: "语法分析token序列出错" , errorToken).toString());
bw.close();
}
}
analyzer main()
Run analyzer
"C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=65012:E:\IntelliJ IDEA 2017.3.4\bin" -Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer
[E, E', T, T', F]
[+, *, (,), id, $]
0 2 E->TE'
0 4 E->TE'
1 0 E'->+TE'
1 3 E'->ε
1 5 E'->ε
2 2 T->FT'
2 4 T->FT'
3 0 T'->ε
4: Run 6: TODO 9: Version Control Terminal 0: Messages
Compilation completed successfully in 1s 266ms (29 minutes ago)
84:25 CRLF: UTF-8: Git: master
激活 Windows
转到"设置"以激活 Windows。

运行截图

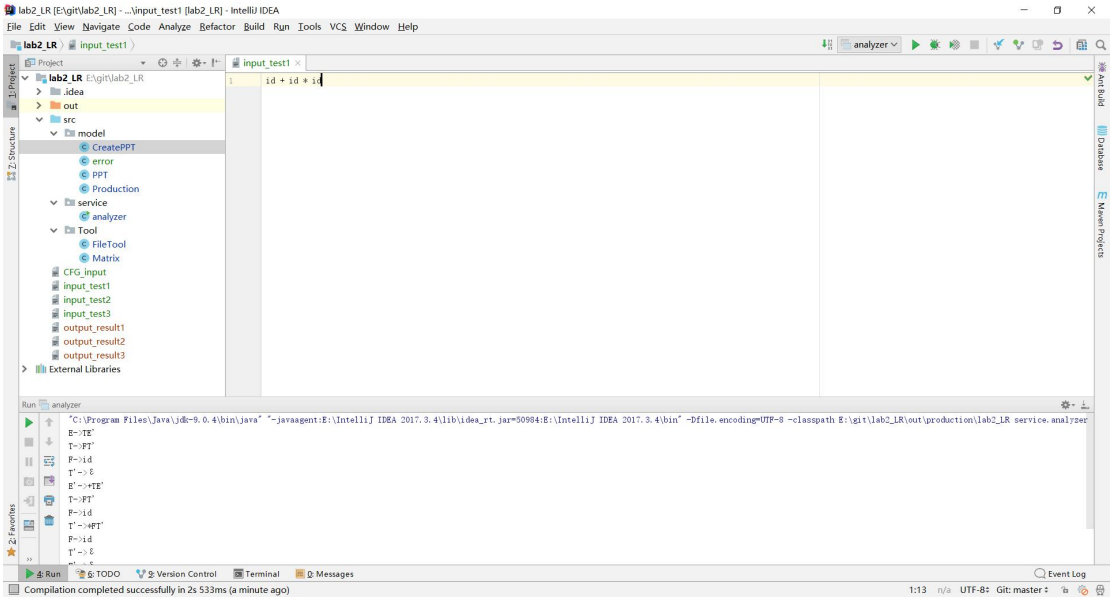
Input_test1 输入文件：



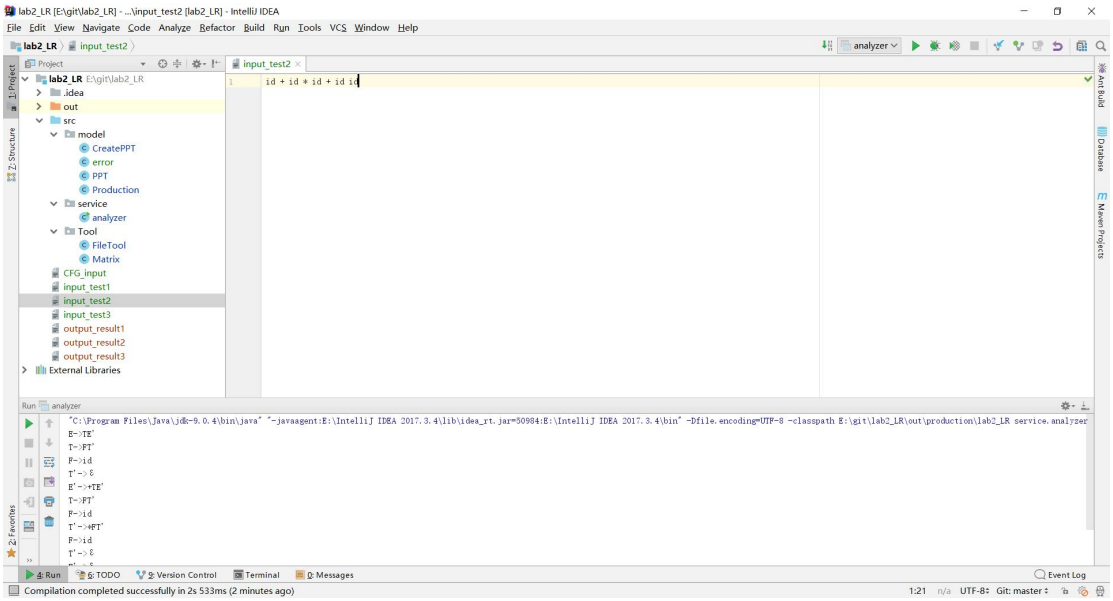
Output_result1 输出文件：



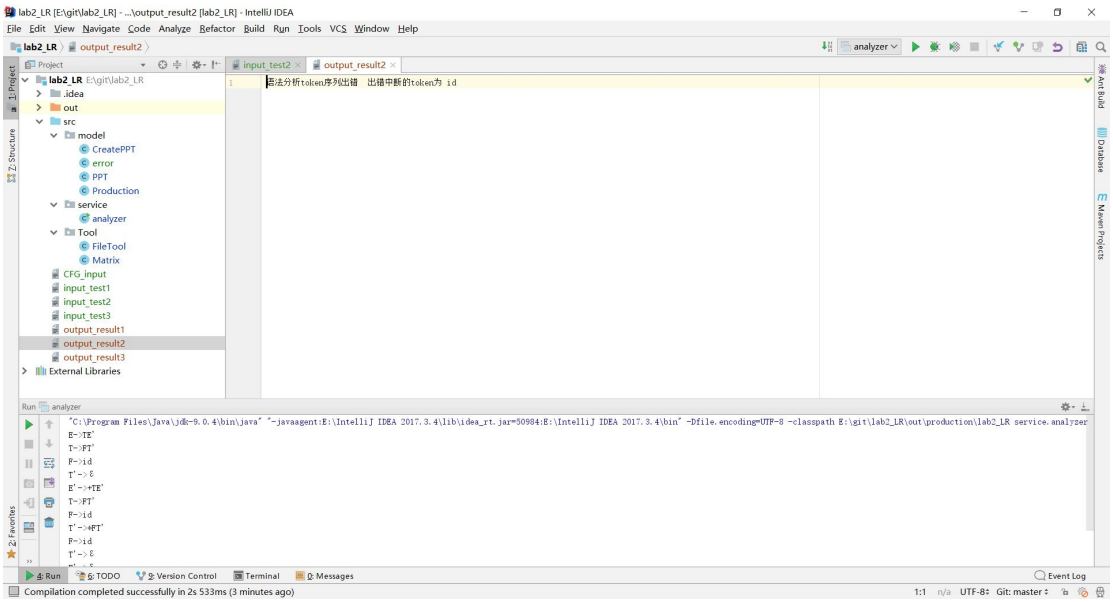
控制台输出：



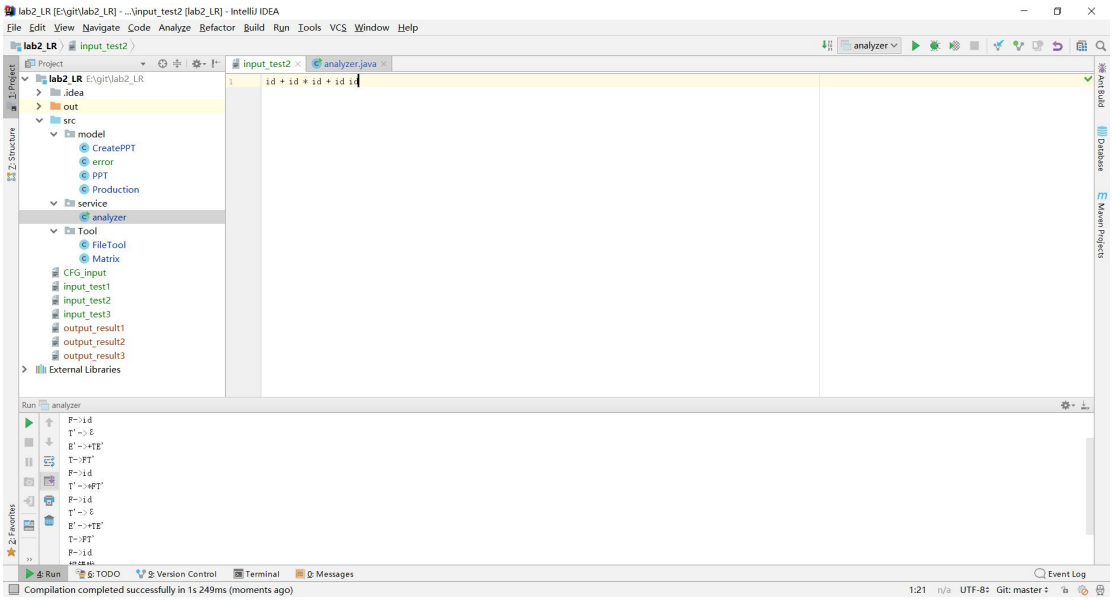
Input_test2 输入文件：



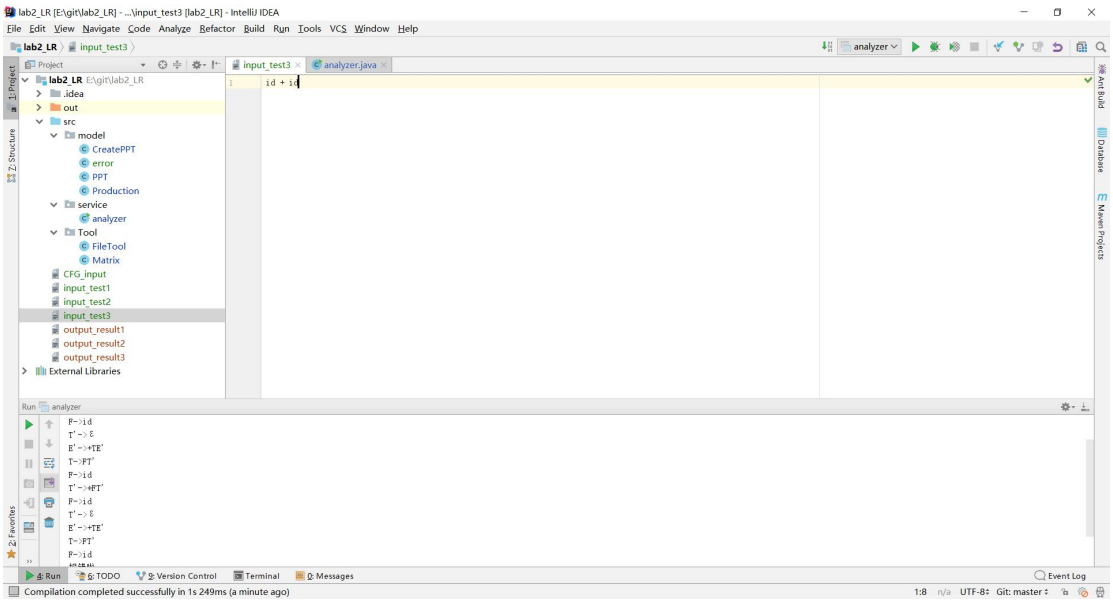
Outoput_result2 输出文件：



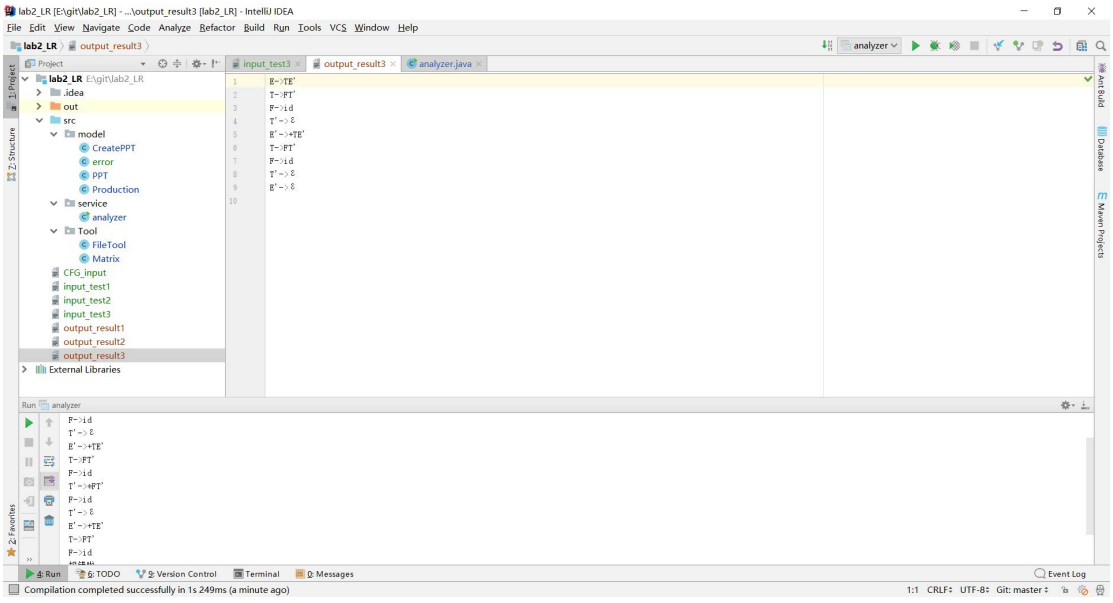
控制台输出：



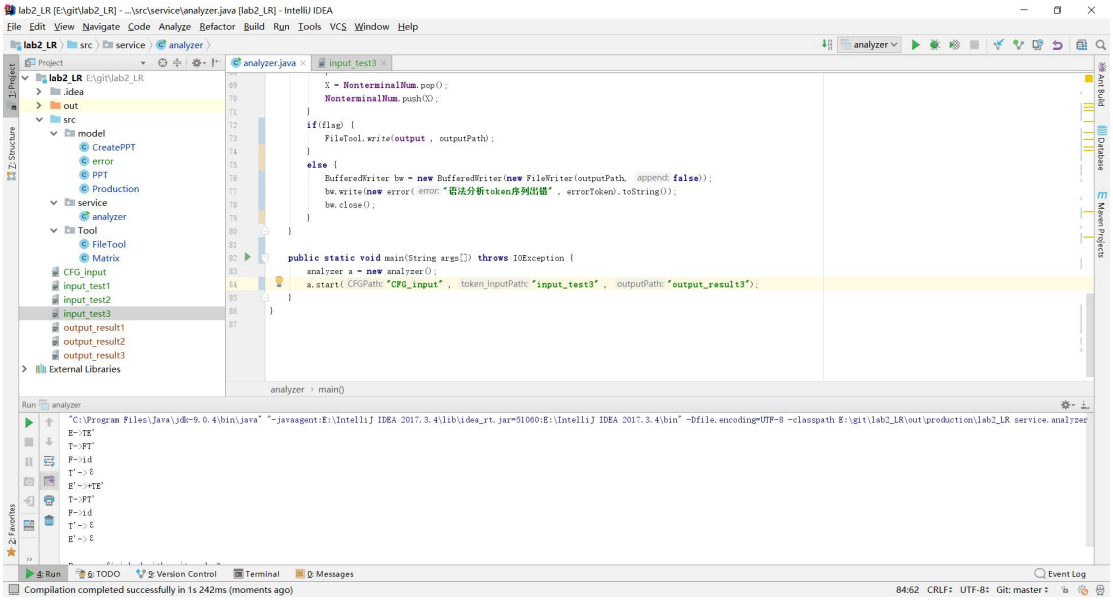
Input_test3 输入文件：



Outoput_result3 输出文件：

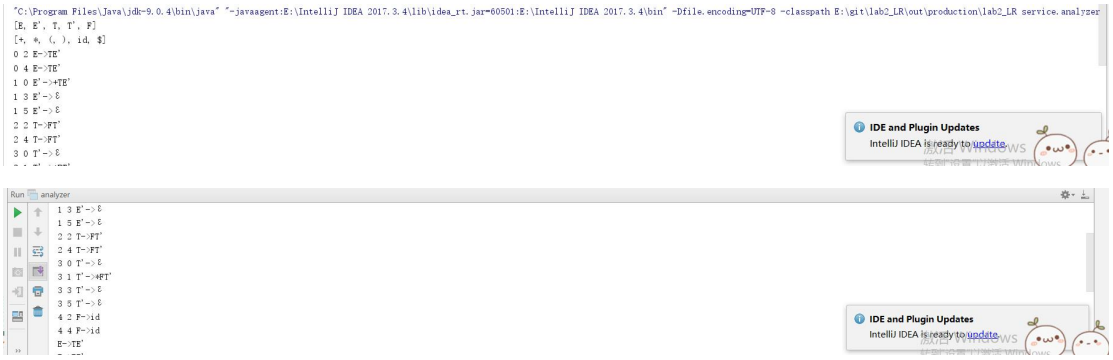


控制台输出：



程序入口就在这个 main 函数里，如果助教想测试的话，只需要把路径换一下就好了！

预测分析表输出检验：



问题与解决

一开始忘记在 token 序列后面加上”$”结束符号，导致在执行第一个条件判断语句的时候，会报字符串指针越界的错误。因为有一种情况（碰巧被我测试的时候碰上了）当把最后一个 token 弹出的时候，如果没有”$”，循环可能还会继续走下去，因为栈这时未必为空，而不添加”$”的 token 序列已经到头了。所以当进入第一个条件时，指针会加一，也就会报越界错误了。因此，我后来的解决方案就是在得到的 token 序列后加上”$”。

感受与总结

实验发现，感觉最难实现的还是生成语法分析表，大部分的代码实现都是花在了写生成预测分析表 PPT 上了。得到语法分析表之后，进行语法分析器就容易得多。所谓“工欲善其事必先利其器”，差不多就是这个道理吧！