

# 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。

```

"C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=60501: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'->E
1 5 E'->E
2 2 T->FT'
2 4 T->FT'
3 0 T'->E
3 1 T'->FT'
3 3 T'->E
3 5 T'->E
4 2 P->id
4 4 P->id
E->TE'

```

The screenshot shows the IntelliJ IDEA interface with the 'analyzer' tool open. The left pane displays a tree structure of grammar rules and their derivations. The right pane shows a 'IDE and Plugin Updates' notification from 'IntelliJ IDEA'.

## 重要数据结构描述

总共有 7 个类。

The screenshot shows the IntelliJ IDEA interface with the 'CreatePPT.java' file open in the code editor. The left pane shows the project structure with packages like 'lab2\_LR' and 'model'. The bottom status bar indicates 'Compilation completed successfully in 1s 357ms (3 minutes ago)'.

```

lab2_LR [E:\git\lab2_LR] - ...\\src\\model\\CreatePPT.java [lab2_LR] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
lab2_LR > src > model > CreatePPT.java
Project Z-Space External Libraries
1 2 3 4 result.txt
public class CreatePPT {
    ArrayList<String> input;
    ArrayList<String> nonterminalList;
    ArrayList<String> terminalList;
}

public CreatePPT(ArrayList<String> input, ArrayList<String> nonterminalList, ArrayList<String> terminalList) {
    this.input = input;
    this.nonterminalList = nonterminalList;
    this.terminalList = terminalList;
}

public CreatePPT(String path) throws IOException {
    this.input = FileTool.readPath();
    this.nonterminalList = new ArrayList<String>();
    this.terminalList = new ArrayList<String>();
}

public PPT create() {
    System.out.println("input");
    CreatePPT > create()
}

```

## CreatePPT 类:

The screenshot shows the IntelliJ IDEA interface with the project 'lab2\_LR' open. The 'CreatePPT.java' file is selected in the editor. The code defines a class 'CreatePPT' with methods for creating a PPT from input and reading from a file. Below the editor is a 'Run' tool window showing a tree of grammar rules like T->E, E->TE, etc., and a status message 'Process finished with exit code 0'. The bottom status bar indicates 'Compilation completed successfully in 1s 357ms (5 minutes ago)'.

```
package model;
import ...

public class CreatePPT {
    ArrayList<String> input;
    ArrayList<String> nonterminalList;
    ArrayList<String> terminalList;
}

public CreatePPT(ArrayList<String> input, ArrayList<String> nonterminalList, ArrayList<String> terminalList) {
    this.input = input;
    this.nonterminalList = nonterminalList;
    this.terminalList = terminalList;
}

public CreatePPT(String path) throws IOException {
    this.input = FileTool.readPath();
    this.nonterminalList = new ArrayList<String>();
    this.terminalList = new ArrayList<String>();
}

public PPT create() {
    System.out.println("input");
    CreatePPT.create();
}
```

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。

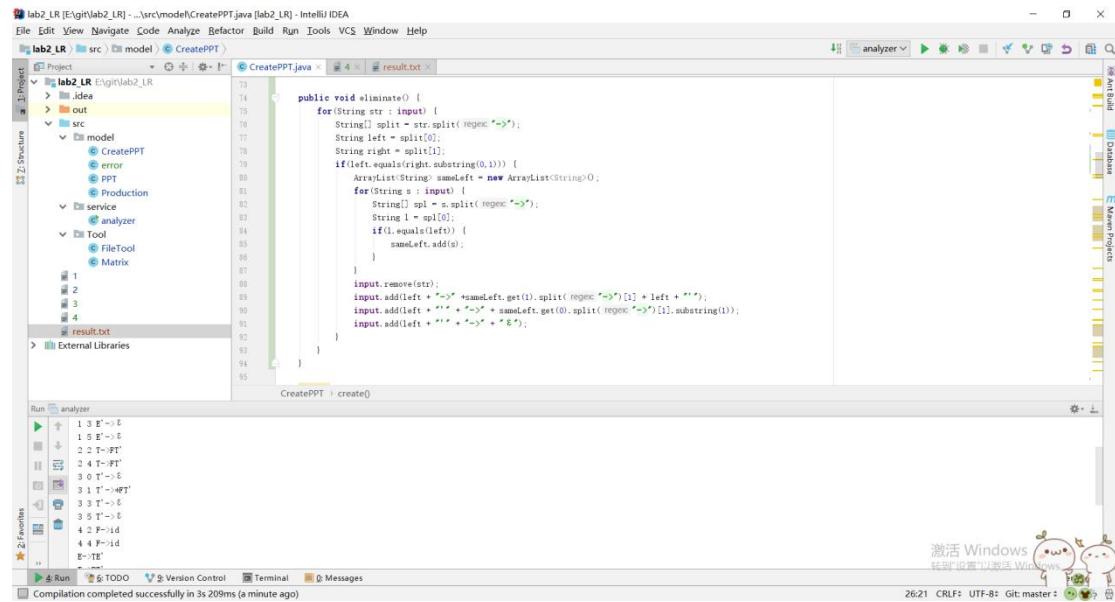
主要方法说明：

`create` 方法：是产生 ppt 预测分析表。

The screenshot shows the IntelliJ IDEA interface with the project 'lab2\_LR' open. The 'CreatePPT.java' file is selected in the editor. The code implements the `create()` method by splitting input strings into left and right parts, then adding them to the respective lists based on their type (non-terminal or terminal). Below the editor is a 'Run' tool window showing a tree of grammar rules and a status message 'Process finished with exit code 0'. The bottom status bar indicates 'Compilation completed successfully in 1s 357ms (9 minutes ago)'.

```
public PPT create() {
    for(String str: input) {
        String[] split = str.split("->");
        String left = split[0];
        //非终结符
        if(!isExistInTerminal(left)) {
            nonterminalList.add(left);
        }
    }
    for(String str: input) {
        //终结符
        String[] split = str.split("->");
        String right = split[1];
        for(int i = 0 ; i < right.length() ; i++) {
            if((right.length()-1 == right.substring(i, i+1)) && right.substring(i, i+2).equals("id")) {
                i++;
            } else if((!isExistInTerminal(right.substring(i, i+1)) && !isExistInNonterminal(right.substring(i, i+1))) && right.substring(i, i+1).equals("*")) {
                terminalList.add(right.substring(i, i+1));
            }
        }
    }
}
```

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。



The screenshot shows the IntelliJ IDEA interface with the project 'lab2\_LR' open. The 'CreatePPT.java' file is the active editor. The code implements the 'eliminate' method to remove left recursion from grammar rules. It uses regular expressions to split strings and an ArrayList to store non-terminal symbols. The 'Run' tool window at the bottom shows the results of the analysis, which include various grammar rules like '1 3 E-> E', '2 2 T-> PT', etc.

```
public void eliminate() {
    for(String str : input) {
        String[] split = str.split(regex:"->*");
        String left = split[0];
        String right = split[1];
        if(left.equals(right.substring(0,1))) {
            ArrayList<String> sameLeft = new ArrayList<String>();
            for(String s : input) {
                String[] spl = s.split(regex:"->*");
                String l = spl[0];
                if(l.equals(left)) {
                    sameLeft.add(s);
                }
            }
            input.remove(str);
            input.add(left + "-" + sameLeft.get(0).split(regex:"->*")[1] + left + "-");
            input.add(left + "-" + sameLeft.get(0).split(regex:"->*")[1] + "-" + sameLeft.get(0).split(regex:"->*")[1].substring(1));
            input.add(left + "-" + sameLeft.get(0).split(regex:"->*")[1] + "-" + sameLeft.get(0).split(regex:"->*")[1].substring(1));
        }
    }
}
```

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。

```

public ArrayList<String> first(String X, ArrayList<String> line) {
    ArrayList<Production> target = getLeft(X);
    for (Production p : target) {
        if (p.getRight().size() == 1 && p.getRight().get(0).equals("ε")) {
            line.add(p.getRight().get(0));
        } else if (p.getRight().size() == 1 && !isExistInNonterminal(p.getRight().get(0)) && isExistInTerminal(p.getRight().get(0))) {
            line.add(p.getRight().get(0));
        } else if (p.getRight().get(0).equals("i")) {
            line.remove(p.getRight().get(0));
            line.add(p.getRight().get(0).get(0));
        }
    }
    if (p.getRight().size() == 2 && p.getRight().get(0).equals("ε")) {
        first(X, p.getRight().get(0), line);
    } else {
        if (p.getRight().size() == 1) {
            first(p.getRight().get(0), line);
        }
    }
}

```

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。

```

public ArrayList<String> follow(String X, ArrayList<String> line) { //对表示
    ArrayList<Production> right = getRight(X);
    for (Production p : right) {
        if (X.equals(nonterminalList.get(0))) {
            //开始符
            line.add("$");
            int index = getIndex(X, p.getRight());
            if (index == -1) {
                if (!p.getLeft().equals("S")) { //防止死循环
                    follow(p.getLeft(), line);
                }
            }
            else if (!isExistInNonterminal(p.getRight().get(index)) && isExistInTerminal(p.getRight().get(index))) {
                line.add(p.getRight().get(index));
            }
            else {
                ArrayList<String> newline = new ArrayList<String>();
                for (String s : first(p.getRight().get(index), newline)) {
                    //会在first那里停下来
                    line.add(s);
                }
                if (hasNull(p.getRight().get(index))) {
                    follow(p.getRight().get(index), line);
                }
            }
        }
    }
}

```

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

```
package model;
public class error {
    private String error;
    private String Token;
    public error(String error, String token) {
        this.error = error;
        Token = token;
    }
    public String getError() {
        return error;
    }
    public void setError(String error) {
        this.error = error;
    }
    public String getToken() {
        return Token;
    }
}
```

主要方法分析:

重写 `toString` 的方法, 将 `error` 转为字符串。

```
return error;
}
public void setError(String error) {
    this.error = error;
}
public String getToken() {
    return Token;
}
public void setToken(String token) {
    Token = token;
}
public String toString() {
    return error + '出错中断的token为' + Token;
}
```

## 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

```
package model;
import ...;

public class PPT {
    private Matrix matrix; //PPT预测分析表
    private ArrayList<String> nonterminalList;
    private ArrayList<String> terminalList;

    public PPT(int row, int column) {
        this.matrix = new Matrix(row, column);
        this.nonterminalList = new ArrayList<String>();
        this.terminalList = new ArrayList<String>();
    }

    public void addProduction(int i, int j, Production pro) {
        this.matrix.add(i, j, pro);
    }

    public Production getSentence(int i, int j) {
        return this.matrix.get(i, j);
    }

    public Matrix getMatrix() {
        return this.matrix;
    }

    PPT > print()
}
```

The screenshot shows the IntelliJ IDEA interface with the PPT.java file open. The code defines a PPT class with a matrix field and two ArrayLists for non-terminal and terminal symbols. It includes methods for adding productions and getting sentences from the matrix. The bottom status bar indicates "Compilation completed successfully in 3s 209ms (5 minutes ago)".

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

```
package model;
import ...;

public class PPT {
    private Matrix matrix; //PPT预测分析表
    private ArrayList<String> nonterminalList;
    private ArrayList<String> terminalList;

    public PPT(int row, int column) {
        this.matrix = new Matrix(row, column);
        this.nonterminalList = new ArrayList<String>();
        this.terminalList = new ArrayList<String>();
    }

    public void addProduction(int i, int j, Production pro) {
        this.matrix.add(i, j, pro);
    }

    public Production getSentence(int i, int j) {
        return this.matrix.get(i, j);
    }

    public Matrix getMatrix() {
        return this.matrix;
    }

    PPT > print()
}
```

The screenshot shows the IntelliJ IDEA interface with the PPT.java file open. The code is identical to the previous screenshot, defining the PPT class with its methods and fields. The bottom status bar indicates "Compilation completed successfully in 3s 209ms (8 minutes ago)".

`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。

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "lab2\_LR". The "src" directory contains "model" and "Tool" packages, with "PPT" selected.
- Code Editor:** Displays the `PPT.java` file. The code implements methods `getRow` and `getColumn` to find indices of tokens in a terminal list.
- Run Tab:** Contains a list of recent runs, including various grammar rules like `E->E`, `E->E`, `F->F`, etc.
- Bottom Status Bar:** Shows "Compilation completed successfully in 3s 209ms (11 minutes ago)" and the current time "82:88".

```
public int getRow(String x) {
    int index = -1;
    for(int i = 0 ; i < nonterminalList.size() ; i++) {
        if(x.equals(nonterminalList.get(i))) {
            index = i;
            break;
        }
    }
    return index;
}

public int getColumn(String x) {
    int index = -1;
    for(int i = 0 ; i < terminalList.size() ; i++) {
        if(x.equals(terminalList.get(i))) {
            index = i;
            break;
        }
    }
    return index;
}
```

## Production 类:

类属性: left 左部。

Right 右部。

The screenshot shows the IntelliJ IDEA interface with the Production.java file open. The code defines a Production class with private attributes left and right, both of type ArrayList<String>. It includes a constructor, a getLeft() method, a setLeft() method, and a getRight() method. The code is annotated with comments //左部 and //右部. The IDE's status bar at the bottom indicates a successful compilation.

```
package model;
import java.util.ArrayList;

public class Production {
    private String left; //左部
    private ArrayList<String> right; //右部

    public Production(String left, ArrayList<String> right) {
        this.left = left;
        this.right = right;
    }

    public Production() {
    }

    public String getLeft() { return left; }

    public void setLeft(String left) { this.left = left; }

    public ArrayList<String> getRight() { return right; }
}
```

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）。

The screenshot shows the IntelliJ IDEA interface with the Production.java file open, focusing on the toString() method. The method concatenates the left attribute and the result of the toRight() method. The toRight() method iterates through the right ArrayList and concatenates each string with a separator. The IDE's status bar at the bottom indicates a successful compilation.

```
public String getLeft() { return left; }

public void setLeft(String left) { this.left = left; }

public ArrayList<String> getRight() { return right; }

public void setRight(ArrayList<String> right) { this.right = right; }

public String toString() { return left + "→" + toRight(); }

public String toRight() {
    String str = "";
    for (String s : right) {
        str += str + s;
    }
    return str;
}
```

## 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

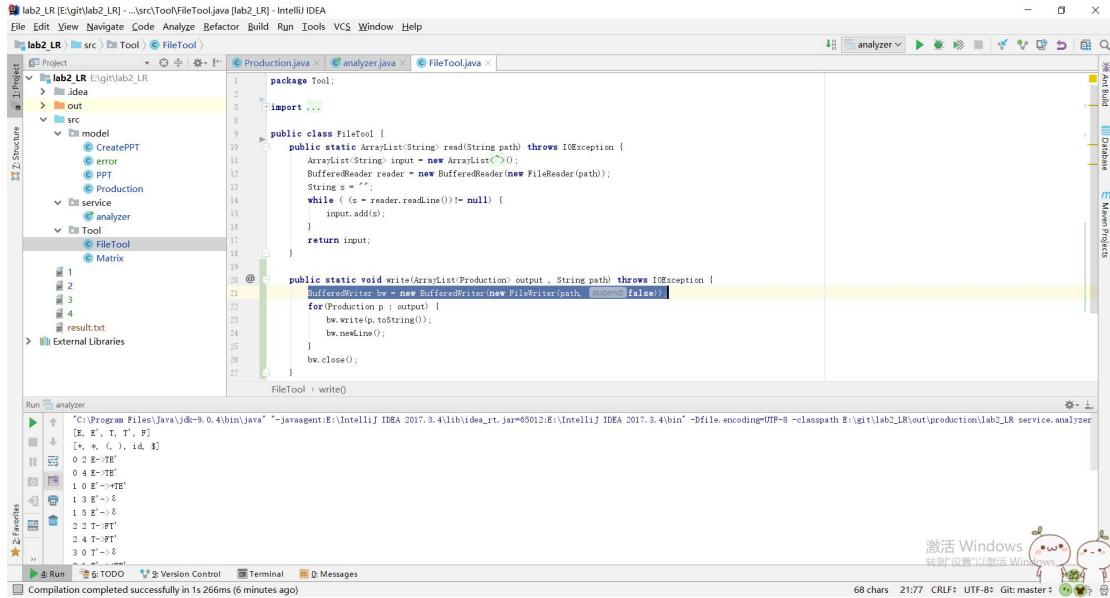
参数分别为 CFG 的文件路径，输入的 token 序列文件路径，以及输出文件的路径。

The screenshot shows the IntelliJ IDEA interface with the 'analyzer' class open in the editor. The code implements a parser for a grammar defined in a CFG file. It reads tokens from an input file, processes them using a stack of non-terminal symbols, and outputs the resulting derivation trees. The 'Run' tool window at the bottom displays the generated trees for each input token.

```
public class analyzer {
    public void start(String CFGPath, String token_inputPath, String outputPath) throws IOException {
        Stack<String> NonterminalNum = new Stack<String>();
        ArrayList<Production> output = new ArrayList<Production>();
        CreatePPT create = new CreatePPT(CFGPath);
        PPT ppt = create.create();
        NonterminalNum.push("S");
        NonterminalNum.push(ppt.getNonterminalList().get(0));
        output.add(ppt.getProduction(0));
        ArrayList<String> read = FileTool.read(token_inputPath);
        ArrayList<String> input = new ArrayList<String>();
        String[] split = read.get(0).split(" ");
        for(int i = 0; i < split.length; i++) {
            input.add(split[i]);
        }
        input.add("$");
        int index = 0;
        String X = ppt.getNonterminalList().get(0);
        boolean flag = true;
        String errorToken = "";
        // 必须得给输入文法提供一个终结符符号,不然会在第一个判断语句里报数据溢出的错误
        while(!input.get(index).equals("$") && !X.equals("$")) {
            if(X.equals(input.get(index))) {
                if(X.equals(input.get(index))) {
```

## FileTool 类

主要功能是文件的读和写



```
lab2_LR (E:\git\lab2_LR - ...\\src\Tool\FileTool.java [lab2_LR] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
lab2_LR src Tool FileTool
1-Project Z-Structure 1-Tool analyzer
lab2_LR E:\git\lab2_LR
> idea
> out
> src
  > model
    < CreatePPT
    < error
    < PPT
    < Production
  > service
    < analyzer
  > Tool
    < FileTool
      < Matrix
      < result.txt
> External Libraries
FileTool.java
1 package Tool;
2
3 import ...
4
5
6 public class FileTool {
7     public static ArrayList<String> read(String path) throws IOException {
8         ArrayList<String> input = new ArrayList<String>();
9         BufferedReader reader = new BufferedReader(new FileReader(path));
10        String s = "";
11        while ((s = reader.readLine()) != null) {
12            input.add(s);
13        }
14        return input;
15    }
16
17    public static void write(ArrayList<Production> output, String path) throws IOException {
18        BufferedWriter bw = new BufferedWriter(new FileWriter(path, false));
19        for (Production p : output) {
20            bw.write(p.toString());
21            bw.newLine();
22        }
23        bw.close();
24    }
25
26
27 }
FileTool.java: write()
```

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, 0 E'->F'  
1, 0 E'->TE'  
1, 1 E'->S  
1, 2 E'->T'  
2, 2 T->FT'  
2, 4 T->FT'  
3, 0 T'->S

Run TODO Version Control Terminal Messages

Compilation completed successfully in 1s 266ms (6 minutes ago)

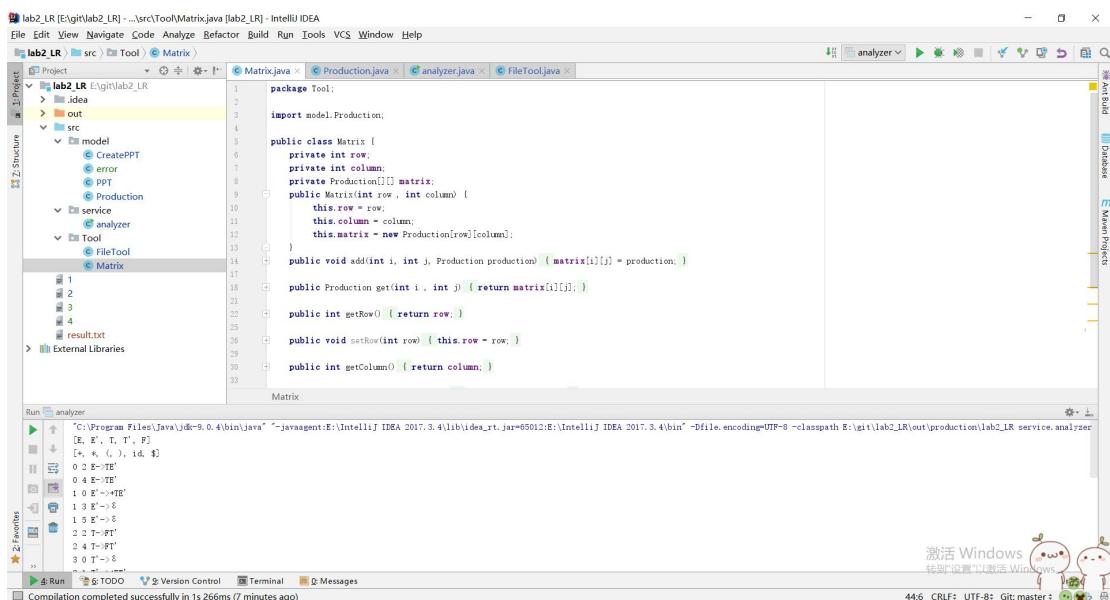
68 chars 21:77 CRLF: UTF-8 Git: master: 446

## 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。



```
lab2_LR (E:\git\lab2_LR - ...\\src\Tool\Matrix.java [lab2_LR] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
lab2_LR src Tool Matrix
1-Project Z-Structure 1-Tool analyzer
lab2_LR E:\git\lab2_LR
> idea
> out
> src
  > model
    < CreatePPT
    < error
    < PPT
    < Production
  > service
    < analyzer
  > Tool
    < FileTool
      < Matrix
      < result.txt
> External Libraries
Matrix.java
1 package Tool;
2
3 import model.Production;
4
5
6 public class Matrix {
7     private int row;
8     private int column;
9     private Production[][] matrix;
10    public Matrix(int row, int column) {
11        this.row = row;
12        this.column = column;
13        this.matrix = new Production[row][column];
14    }
15    public void add(int i, int j, Production production) { matrix[i][j] = production; }
16
17    public Production get(int i, int j) { return matrix[i][j]; }
18
19    public int getRow() { return row; }
20
21    public void setRow(int row) { this.row = row; }
22
23    public int getColumn() { return column; }
24
25
26
27 }
Matrix.java: getRow()
```

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, 0 E'->F'  
1, 0 E'->TE'  
1, 1 E'->S  
1, 2 E'->T'  
2, 2 T->FT'  
2, 4 T->FT'  
3, 0 T'->S

Run TODO Version Control Terminal Messages

Compilation completed successfully in 1s 266ms (7 minutes ago)

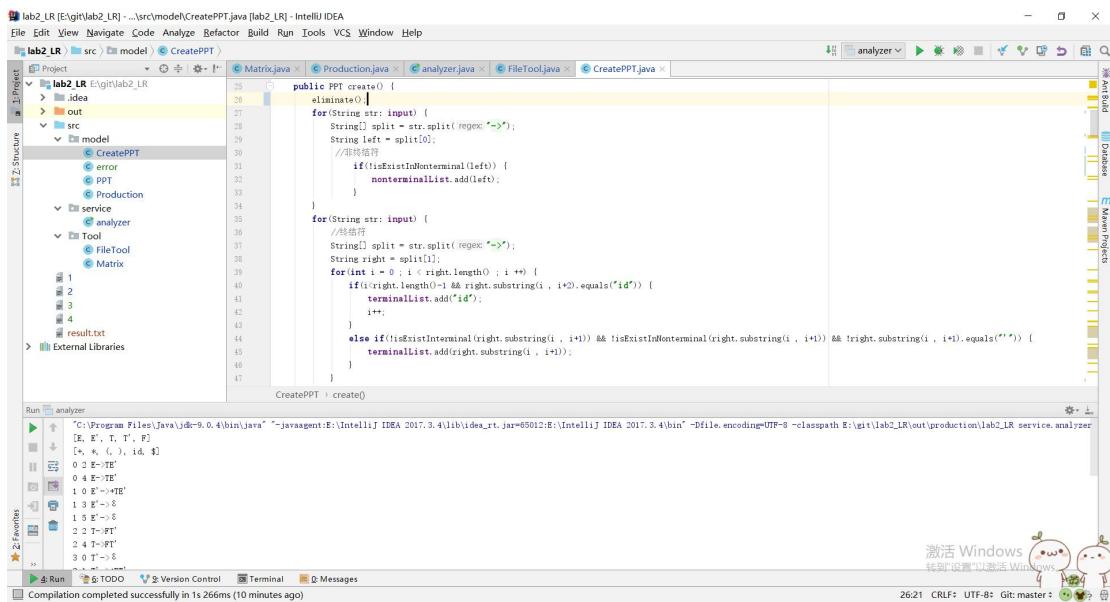
446 CRLF: UTF-8 Git: master: 446

# 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。



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** Shows the project structure for "lab2\_LR".
- CreatePPT.java:** The code editor displays the `CreatePPT` class with its `create()` method. The code implements the `eliminate` method to handle direct left-recursion by adding new non-terminals.
- Run Tool Window:** The "analyzer" configuration is selected, showing the command: `C:\Program Files\Java\jdk-9.0.4\bin\java -javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=E:\IntelliJ IDEA 2017.3.4\bin -Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer`.
- Status Bar:** Shows the message "激活 Windows 转到设置以激活 Windows" and the time "26:21 CRLF: UTF-8 Git: master".

```

public void create() {
    for(String str : input) {
        String[] split = str.split(">\"");
        String left = split[0];
        String right = split[1];
        if(left.equals(right.substring(0,1))) {
            ArrayList<String> sameLeft = new ArrayList<String>();
            for(String s : input) {
                String[] spl = s.split(">\"");
                String l = spl[0];
                if(l.equals(left)) {
                    sameLeft.add(s);
                }
            }
            input.remove(str);
            input.add(left + ">\"" + sameLeft.get(0).split(" >\"")[1] + left + "\"");
            input.add(left + ">\"" + sameLeft.get(0).split(" >\"")[1].substring(1) + "\"");
            input.add(left + ">\"" + right + "\"");
        }
    }
}

```

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。

```

public ArrayList<String> first(String X, ArrayList<String> line) {
    ArrayList<Production> target = getLeft(X);
    for(Production p : target) {
        if(p.getRight().size() == 1 & !p.getRight().get(0).equals("ε")) {
            line.add(p.getRight().get(0));
        } else if(p.getRight().size() == 1 & !isExistInNonterminal(p.getRight().get(0)) & isExistInTerminal(p.getRight().get(0))) {
            line.add(p.getRight().get(0));
        } else if(p.getRight().size() == 1 & p.getRight().get(0).equals("i")) {
            line.remove(p.getRight().get(0));
            line.add(p.getRight().get(0) + p.getRight().get(1));
        } else {
            if(p.getRight().size() == 2 & p.getRight().get(0).equals("ε")) {
                first(X, p.getRight().get(1), line);
            } else {
                if(p.getRight().size() == 1) {
                    first(p.getRight().get(0), line);
                }
            }
        }
    }
}

```

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**。

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "lab2\_LR" and contains packages like "Egit\lab2\_LR", "src", and "model".
- Code Editor:** The file "CreatePPT.java" is open, showing Java code for creating a PPT. The code includes methods for handling different types of nodes (Production, NonTerminal, Terminal) and building the final output.
- Run Configuration:** A configuration named "analyzer" is selected, pointing to "C:\Program Files\Java\jdc-9.0.4\bin\javac" as the command.
- Terminal:** The terminal window shows the command being run: "javac -d E:\IntelliJ IDEA 2017.3\lib\idea\_rt.jar -bootclasspath E:\IntelliJ IDEA 2017.3\lib\idea\_rt.jar -encoding UTF-8 -classpath E:\git\lab2\_LR\out\production\lab2\_LR service.analyzer".

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** Shows the project structure under "Lab2\_LR".
- Code Editor:** Displays the `CreatePPT.java` file. The code implements a `CharSequence` and `Comparable<String>`. It includes logic for handling terminal symbols and non-terminals, with annotations like `[< 9.0 >] java.lang` and `implements Serializable`.
- Toolbars:** Standard IntelliJ toolbars for Run, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Bottom Status Bar:** Shows the message "Compilation completed successfully in 1526ms (20 minutes ago)".
- Right Sidebar:** Shows a vertical list of icons representing different tools and features.

## 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；

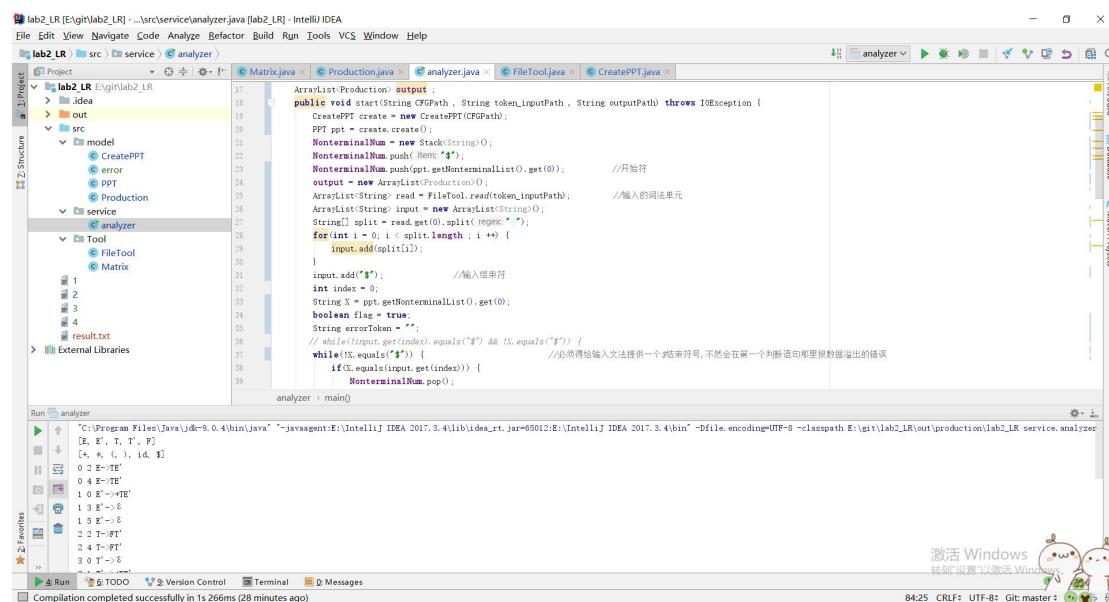
然后当栈非空，如果栈顶符号等于 token 序列所指符号；出栈，token 序列指针后移。

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。



The screenshot shows the IntelliJ IDEA interface with the project 'lab2\_LR' open. The 'analyzer.java' file is the active editor. The code implements a Predictive Parser (LR(0) parser) for a grammar defined in 'Matrix.java'. It uses a stack of terminals to parse input tokens from 'FileTool.java'. The code includes logic to handle lookahead tokens and to push non-terminal symbols onto the stack. A stack trace at the bottom shows the sequence of tokens being processed.

```
ArrayList<Production> output;
public void start(String CFGPath, String token_inputPath, String outputPath) throws IOException {
    CreatePPT create = new CreatePPT(CFGPath);
    PPT ppt = create.create();
    NonterminalNum = new Stack<String>();
    NonterminalNum.push("Start");
    NonterminalNum.push(ppt.getNonterminalList().get(0)); //开始符
    output = new FileOutputStream(outputPath);
    output.write(token_inputPath.getBytes());
    ArrayList<String> read = FileTool.read(token_inputPath);
    ArrayList<String> input = new ArrayList<String>();
    String[] split = read.get(0).split(" ");
    String errorToken = "";
    for(int i = 0; i < split.length; i++) {
        input.add(split[i]);
    }
    input.add("$"); //输入结束符
    int index = 0;
    String X = ppt.getNonterminalList().get(0);
    boolean flag = true;
    String errorToken = "";
    // while((input.get(index).equals("$") || !X.equals("$")) && !flag) {
    while((input.get(index).equals("$") || !X.equals("$")) && !flag) { //必须得给输入文法提供一个结束字符,不然会在最后一个判断语句里报数据溢出的错误
        if(X.equals("$")) {
            if(input.get(index).equals("$")) {
                NonterminalNum.pop();
            }
        }
    }
}
```

lab2\_LR [E:\git\lab2\_LR] - ...\\src\\service\\analyzer.java [lab2\_LR] - IntelliJ IDEA

```
String errorToken = "";
while(!input.get(index).equals("$") && !X.equals("$")) {
    if(X.equals(input.get(index))) {
        NonterminalNum.pop();
        index++;
    }
    else if(ppt.isNonterminal(X)) {
        //不是终结符
        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=60512:E:\IntelliJ IDEA 2017.3.4\bin"-Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer
```

2 Favorites

Run TODO Version Control Terminal Messages

Compilation completed successfully in 1s 266ms (28 minutes ago)

84:25 CRLF: UTF-8 Git: master

lab2\_LR [E:\git\lab2\_LR] - ...\\src\\service\\analyzer.java [lab2\_LR] - IntelliJ IDEA

```
String output = 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("$")) {
        NonterminalNum.push(right.get(i));
    }
}
X = NonterminalNum.pop();
NonterminalNum.push(X);
}
if(flag) {
    FileCol.write(output , outputPath);
}
else {
    BufferedWriter bw = new BufferedWriter(new FileWriter(outputPath, append: false));
    bw.write(error( error: "语法分析token序列出错" , errorToken).toString());
    bw.close();
}
```

Run analyzer

```
C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=60512:E:\IntelliJ IDEA 2017.3.4\bin"-Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer
```

2 Favorites

Run TODO Version Control Terminal Messages

Compilation completed successfully in 1s 266ms (29 minutes ago)

84:25 CRLF: UTF-8 Git: master

# 运行截图

Input\_test1 输入文件:

The screenshot shows the IntelliJ IDEA interface with the project 'lab2\_LR' open. The 'input\_test1' file is selected in the left navigation bar. The code content is as follows:

```
1 id + id * id
```

The 'Run' tool window at the bottom shows the command used to run the analyzer:

```
C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=50994:E:\IntelliJ IDEA 2017.3.4\bin" -Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer
```

The status bar at the bottom right indicates the compilation was successful in 1s 216ms (2 minutes ago).

Output\_result1 输出文件:

The screenshot shows the IntelliJ IDEA interface with the project 'lab2\_LR' open. The 'output\_result1' file is selected in the left navigation bar. The code content is as follows:

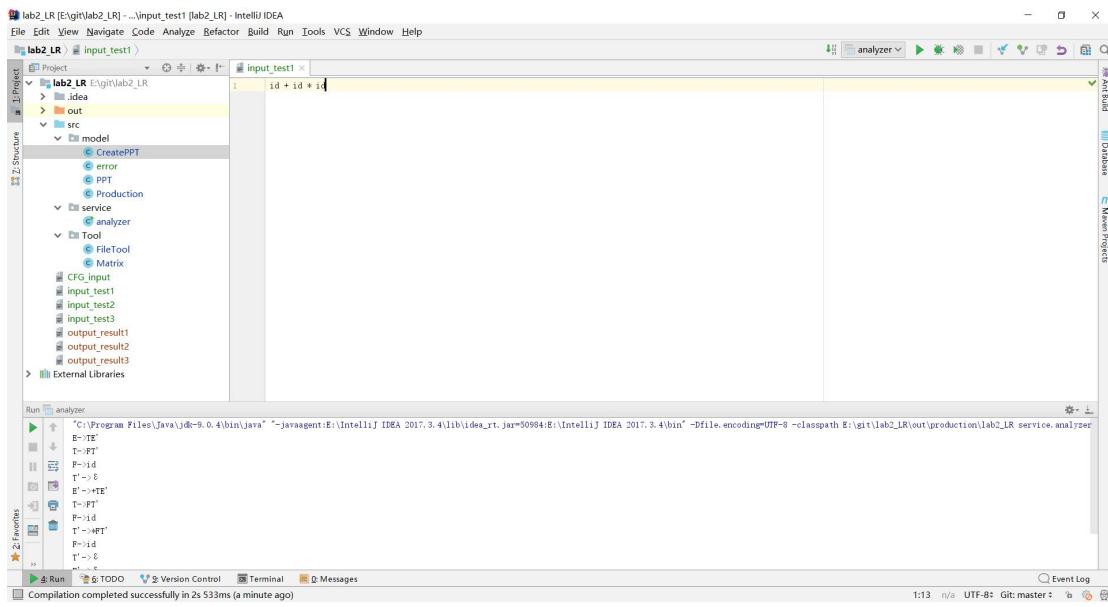
```
1 E->TE'
2 T->T'
3 F->id
4 T'->S
5 E'->TE'
6 T'->T'
7 F->id
8 T'->FT'
9 F->id
10 T'->S
11 E'->S
12
```

The 'Run' tool window at the bottom shows the command used to run the analyzer:

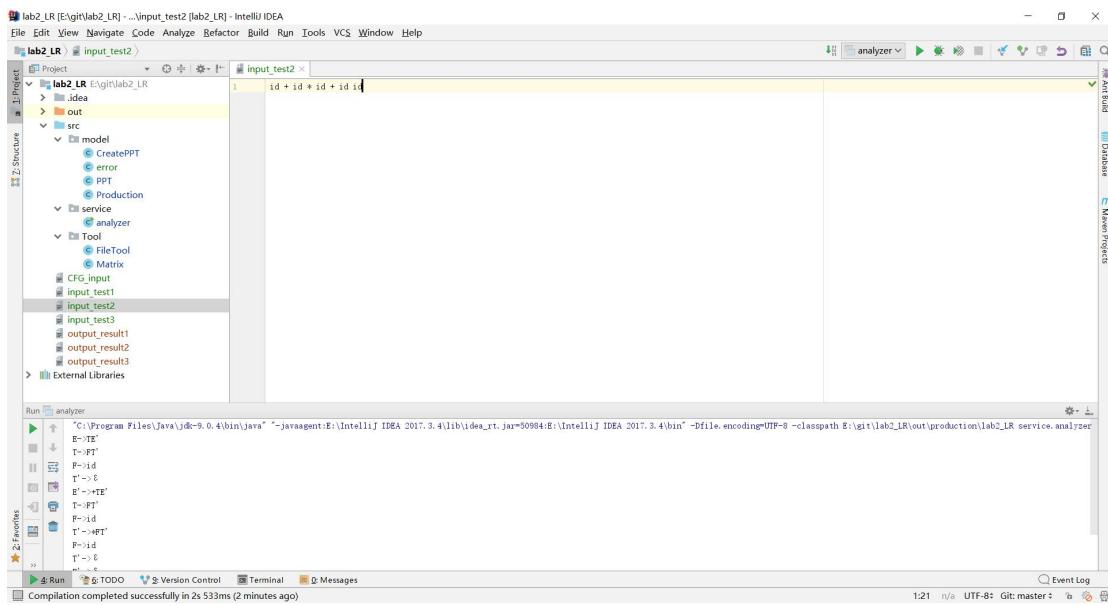
```
C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=50994:E:\IntelliJ IDEA 2017.3.4\bin" -Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer
```

The status bar at the bottom right indicates the compilation was successful in 2s 533ms (a minute ago).

## 控制台输出：



## Input\_test2 输入文件:



## Output\_result2 输出文件:

The screenshot shows the IntelliJ IDEA interface with the project 'lab2\_LR' open. The 'input\_test2' file is selected in the editor. The code contains a single line: `id + id * id + id id`. The status bar at the bottom indicates 'Compilation completed successfully in 2s 533ms (3 minutes ago)'. The 'Run' tool window shows the command used to run the analyzer: `C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=50984:E:\IntelliJ IDEA 2017.3.4\bin" -Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer`.

## 控制台输出:

The screenshot shows the IntelliJ IDEA interface with the project 'lab2\_LR' open. The 'input\_test2' file is selected in the editor. The code contains a single line: `id + id * id + id id`. The status bar at the bottom indicates 'Compilation completed successfully in 1s 249ms (moments ago)'. The 'Run' tool window shows the command used to run the analyzer: `C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=50984:E:\IntelliJ IDEA 2017.3.4\bin" -Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer`.

## Input\_test3 输入文件:

```
lab2_LR [E:\git\lab2_LR - ...\\input_test3 [lab2_LR] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
lab2_LR > input_test3 > analyzer.java
```

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor contains the following Java code:

```
1 id + id
```

The code editor has a yellow background, indicating syntax highlighting for the identifier 'id'.

## Output\_result3 输出文件:

```
lab2_LR [E:\git\lab2_LR - ...\\output_result3 [lab2_LR] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
lab2_LR > output_result3 > analyzer.java
```

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor contains the following Java code:

```
1 E->TE'
2 T->FT'
3 F->id
4 T->S
5 E'->TE'
6 T->FT'
7 F->id
8 T->S
9 E'->S
10
```

The code editor has a white background, indicating syntax highlighting for the identifiers 'E', 'T', 'F', 'id', 'S', and the symbols '>' and '-'.

## 控制台输出:

```
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 Z-Structure I-Project
lab2_LR E:\git\lab2_LR
  > idea
  > out
  > src
    > model
      > CreatePPT
      > error
      > PPT
      > production
    > service
      > analyzer
    > Tool
      > FileTool
      > Matrix
    CFG_input
    input_1
    input_2
    input_3
    output_result1
    output_result2
    output_result3
  > External Libraries
analyzer.java [input_test3]
  89 X = NonterminalNum.pop();
  70   X = NonterminalNum.push(X);
  71 }
  72 if(flag) {
  73   FileTool.write(output, outputPath);
  74 } else {
  75   BufferedWriter bw = new BufferedWriter(new FileWriter(outputPath, append: false));
  76   bw.write(new Error(error: "错误分析token序列出错", errorToken: toString()));
  77   bw.close();
  78 }
  80 }
  81
public static void main(String args[]) throws IOException {
  82   analyzer a = new analyzer();
  83   a.start(CFGPath: "CFG_input", token_inputPath: "input_test3", outputPath: "output_result3");
  84 }
  85
  86 }

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=51000:E:\IntelliJ IDEA 2017.3.4\bin"-Dfile.encoding=UTF-8 -classpath E:\git\lab2_LR\out\production\lab2_LR service.analyzer
E->TE
E->T
T->T
T->E
E->TE
E->T
T->T
T->id
T->S
E->S
Run TODO Version Control Terminal Messages Event Log
Compilation completed successfully in 1s 242ms (moments ago)
84:62 CRLF: UTF-8 Git: master: 8 8
```

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:

```
C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:E:\IntelliJ IDEA 2017.3.4\lib\idea_rt.jar=60501: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'->E
1 5 E'->S
2 2 T->FT
2 4 T->FT'
3 0 T'->E
1 3 E'->E
1 5 E'->S
2 2 T->FT
2 4 T->FT'
3 0 T'->E
1 3 E'->E
1 5 E'->S
2 2 T->FT
2 4 T->FT'
3 0 T'->E
3 3 T'->E
4 2 F->id
4 4 F->id
5 2 E->TE
5 4 E->TE'

Run analyzer
1 3 E'->E
1 5 E'->S
2 2 T->FT
2 4 T->FT'
3 0 T'->E
3 3 T'->E
3 5 T'->S
4 2 F->id
4 4 F->id
5 2 E->TE
5 4 E->TE

IDE and Plugin Updates
IntelliJ IDEA is ready to update
IDEA 2017.3.4 V 2017.3.4 Windows
4.2.0 (173.455.17) 17.3.4.20170328-1734
```

## 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！