

《编译原理》研究性学习专题实验报告

实验3： LL(1)语法分析设计原理与实现

学 院：计算机与信息技术学院

专 业： 计算机科学与技术

学生姓名： 杜永坤

学 号： 16281002

班 级： 计算机1601班

指导教师： 于双元

北京交通大学

2019年05月09日

目录

[实验要求： 4](#_Toc8320233)

[1.[实验项目] 4](#_Toc8320234)

[2.[设计说明] 4](#_Toc8320235)

[3.[设计要求] 4](#_Toc8320236)

[实验过程： 4](#_Toc8320237)

[1、文法扩充修改 4](#_Toc8320238)

[1.1、修改文法： 4](#_Toc8320239)

[1.2、扩充后E表示的表达式可以识别逻辑表达式 5](#_Toc8320240)

[1.3、if语句文法设计说明 5](#_Toc8320241)

[2、LL1分析程序设计说明 6](#_Toc8320242)

[2.1设计要求： 6](#_Toc8320243)

[2.2设计说明： 6](#_Toc8320244)

[3、程序功能描述 7](#_Toc8320245)

[4、主要的数据结构描述 7](#_Toc8320246)

[4.1主要使用的java数据结构类型 7](#_Toc8320247)

[4.1.1 List 7](#_Toc8320248)

[list中添加，获取，删除元素 7](#_Toc8320249)

[list中是否包含某个元素 8](#_Toc8320250)

[list中根据索引将元素数值改变(替换) 8](#_Toc8320251)

[list中查看（判断）元素的索引 8](#_Toc8320252)

[根据元素索引位置进行的判断 9](#_Toc8320253)

[利用list中索引位置重新生成一个新的list（截取集合） 9](#_Toc8320254)

[对比两个list中的所有元素 9](#_Toc8320255)

[判断list是否为空 10](#_Toc8320256)

[4.1.2 Map 10](#_Toc8320257)

[HashMap 10](#_Toc8320258)

[常用API 10](#_Toc8320259)

[4.2 二元式文件结构 11](#_Toc8320260)

[4.3文法 12](#_Toc8320261)

[4.4 非终极符号与终结符号 13](#_Toc8320262)

[4.5 FIRST 集 14](#_Toc8320263)

[4.6 FOLLOW 集 15](#_Toc8320264)

[4.7 LL(1)分析表 17](#_Toc8320265)

[5、程序结构描述 18](#_Toc8320266)

[5.1Java 主类：LL(1)语法分析主类LL1PredictionAnalysisMain 18](#_Toc8320267)

[5.2 Java 副类：LL(1)语法分析副类类LL1PredictionAnalysis 18](#_Toc8320268)

[5.2.1私有变量： 19](#_Toc8320269)

[5.2.2副类的构造函数 19](#_Toc8320270)

[5.2.3副类的方法 20](#_Toc8320271)

[5.3 Java 副类 构造文法的First集和Follow集类FirstAndFollow 21](#_Toc8320272)

[5.3.1 私有变量 21](#_Toc8320273)

[5.3.2构造函数FirstAndFollow() 21](#_Toc8320274)

[5.3.3 FirstAndFollow副类的方法 24](#_Toc8320275)

[5.4 JAVA类之间关系 25](#_Toc8320276)

[5.4.1 类之间的关系 25](#_Toc8320277)

[6、程序测试 25](#_Toc8320278)

[6.1 正确用例 25](#_Toc8320279)

[6.2 错误用例 29](#_Toc8320280)

[6.2.1缺少 ‘；’符号 29](#_Toc8320281)

[6.2.1缺少 ‘）’符号 29](#_Toc8320282)

[6.2.3缺少if语句程序体: 30](#_Toc8320283)

[实验过程总结 30](#_Toc8320284)

[1、LL(1)分析器分析步骤 30](#_Toc8320285)

[1.1 FIRST集 30](#_Toc8320286)

[1.2 FOLLOW集 30](#_Toc8320287)

[1.3 LL(1)分析表构建 31](#_Toc8320288)

[1.4 LL(1)分析过程 31](#_Toc8320289)

[2、 实验输出分析过程的完善，print\_indexp(int i) 32](#_Toc8320290)

### 实验要求：

1.[实验项目]

实现LL(1)分析中控制程序（表驱动程序）；完成以下描述赋值语句的LL(1)文法的LL(1)分析过程。

G[A]:A→V=E

E→TE′

E′→ATE′|ε

T→FT′

T′→MFT′|ε

F→ (E)|i

A→+|-

M→\*|/

V→i

2.[设计说明]

终结符号i 为用户定义的简单变量,即标识符的定义。

3.[设计要求]

（1）输入串应是词法分析的输出二元式序列，即某算术表达式“专题1”的输出结果。输出为输入串是否为该文法定义的算术表达式的判断结果；

（2）LL(1)分析过程应能发现输入串出错；

（3）设计两个测试用例（尽可能完备，正确和出错），并给出测试结果；

（4）考虑根据LL(1)文法编写程序构造LL（1）分析表，并添加到你的LL（1）分析程序中。

### 实验过程：

#### 1、文法扩充修改

##### 1.1、修改文法：

文法与专题2修改相同，基本可以识别c语言if语句、判断语句和赋值语句：

G[P]:

P→S|Q|;

S→V=E;

V→i

E→TR

R→ATR|*$*

T→FY

Y→MFY|*$*

F→CZ

Z→OCZ|*$*

C→BI

I→XBI|*$*

B→(E)|i

A→+|-

M→\*|/

X→a|o //a表示逻辑符号&&，o表示逻辑符号||

O→t|d|g|l|u|e //t表示>=，d表示<=，g表示>,l表示<,e表示==，u表示!=

Q→8JKH //8表示if在符号表中序号

H→fJKH|9K|*$* //f 表示 else if符号的组合、9表示else在符号表中的序号

J→(E) //逻辑语句

K→S|{U}|; //if语句程序体

U→PU|{U}U|*$*

##### 1.2、扩充后E表示的表达式可以识别逻辑表达式

规则：

F→CF′

F′→OCF′|ε

C→BC′

C′→XBC′|ε

X→a|o//a表示逻辑符号&&，o表示逻辑符号||

O→t|d|g|l//t表示>=，d表示<=，g表示>,l表示<

将逻辑与（&&）和逻辑或（||）加入到表达式中，将关系运算符（<=、>=、==、!=、<、>）加入到表达式中。

##### 1.3、if语句文法设计说明

Q→8JKH//8表示if在符号表中的序号

H→89JKH|9K|ε//9表示else在符号表中的序号

J→(E) //逻辑语句

K→S|{U}|; //if语句程序体

U→PU|{U}U|$

这部分规则，表示的是if语句的识别规则：

Q→if<判断语句><IF语句程序体><ELSE IF 或者 ELSE语句程序体>

<ELSE IF 或者 ELSE语句程序体>→else if<判断语句><IF语句程序体><ELSE IF 或者 ELSE语句程序体>|else<IF语句程序体>|*$*

<判断语句>→(E) //逻辑语句

<IF语句程序体>→S|{U}|; //if语句程序体

U→PU|{U}U|*$* //U这个部分保证了if语句程序体中能够出现赋值语句和if语句的闭包，并且能识别普通的程序体

3、

其中 S→V=E; 规则我在最后加入了 ; 终结符号，使所有的赋值表达式都比较符合c语言规则。

#### 2、**LL1分析程序设计说明**

##### 2.1设计要求：

（1）输入串应是词法分析的输出二元式序列，即某算术表达式“专题1”的输出结果。输出为输入串是否为该文法定义的算术表达式的判断结果；

（2）LL(1)分析过程应能发现输入串出错；

（3）设计两个测试用例（尽可能完备，正确和出错），并给出测试结果；

（4）考虑根据LL(1)文法编写程序构造LL（1）分析表，并添加到你的LL（1）分析程序中。

##### 2.2设计说明：

（1）该语言大小写不敏感；

（2）字母为a-zA-Z，数字为0-9；

（3）对文法进行扩充和改造；

（4）扩充后E表示的表达式可以识别逻辑表达式

规则：

F→CF′

F′→OCF′|ε

C→BC′

C′→XBC′|ε

X→a|o//a表示逻辑符号&&，o表示逻辑符号||

O→t|d|g|l//t表示>=，d表示<=，g表示>,l表示<

将逻辑与（&&）和逻辑或（||）加入到表达式中，将关系运算符（<=、>=、==、!=、<、>）加入到表达式中。

Q→8JKH//8表示if在符号表中的序号

H→89JKH|9K|ε//9表示else在符号表中的序号

J→(E) //逻辑语句

K→S|{U}|; //if语句程序体

U→PU|{U}U|$

这部分规则，表示的是if语句的识别规则（将字母替换成文字意义的非终结符号）：

Q→if<判断语句><IF语句程序体><ELSE IF 或者 ELSE语句程序体>

<ELSE IF 或者 ELSE语句程序体>→else if<判断语句><IF语句程序体><ELSE IF 或者 ELSE语句程序体>|else<IF语句程序体>|*$*

<判断语句>→(E) //逻辑语句

<IF语句程序体>→S|{U}|; //if语句程序体

U→PU|{U}U|*$* //U这个部分保证了if语句程序体中能够出现赋值语句和if语句的闭包，并且能识别普通的程序体

（5）根据FISRT集和FOLLOW集的构造规则，构造出文法的FIRST集和FOLLOW集

（6）根据FISRT集和FOLLOW集构造LL1分析表；

（7）根据分析规则，构造LL1分析器。

#### 3、**程序功能描述**

（1）、能够录入一个.tys文件中的二元式内容；

（2）、根据.tys文件内容进行语法分析，可识别

①赋值表达式

②逻辑表达式

③c语言if语句；

（3）、根据输入的二元式内容进行分析语法分析，并打印结果；

（4）、打印分析过程（分析栈和保留串）和错误提示；

（5）、根据文法构造FIRST集和FOLLOW集；

（6）、根据构造FIRST集和FOLLOW集，构造出LL1分析表；

（7）、LL1分析器。

#### 4、**主要的数据结构描述**

##### 4.1主要使用的java数据结构类型

###### 4.1.1 List

list中添加，获取，删除元素

List<String> person=new ArrayList<>();

person.add("jackie"); //索引为0 //.add(e)

person.add("peter"); //索引为1

person.add("annie"); //索引为2

person.add("martin"); //索引为3

person.add("marry"); //索引为4

person.remove(3); //.remove(index)

person.remove("marry"); //.remove(Object o)

String per="";

per=person.get(1);

System.out.println(per); ////.get(index)

for (int i = 0; i < person.size(); i++) {

System.out.println(person.get(i)); //.get(index)

}

list中是否包含某个元素

方法：.contains（Object o）； 返回true或者false

List<String> fruits=new ArrayList<>();

fruits.add("苹果");

fruits.add("香蕉");

fruits.add("桃子");

//for循环遍历list

for (int i = 0; i < fruits.size(); i++) {

System.out.println(fruits.get(i));

}

String appleString="苹果";

//true or false

System.out.println("fruits中是否包含苹果："+fruits.contains(appleString));

if (fruits.contains(appleString)) {

System.out.println("我喜欢吃苹果");

}else {

System.out.println("我不开心");

}

list中根据索引将元素数值改变(替换)

.set(index, element); 和 .add(index, element); 的不同

String a="白龙马", b="沙和尚", c="八戒", d="唐僧", e="悟空";

List<String> people=new ArrayList<>();

people.add(a);

people.add(b);

people.add(c);

people.set(0, d); //.set(index, element); //将d唐僧放到list中索引为0的位置，替换a白龙马

people.add(1, e); //.add(index, element); //将e悟空放到list中索引为1的位置,原来位置的b沙和尚后移一位

//增强for循环遍历list

for(String str:people){

System.out.println(str);

}

list中查看（判断）元素的索引

　　注意：.indexOf（）； 和 lastIndexOf（）的不同；

List<String> names=new ArrayList<>();

names.add("刘备"); //索引为0

names.add("关羽"); //索引为1

names.add("张飞"); //索引为2

names.add("刘备"); //索引为3

names.add("张飞"); //索引为4

System.out.println(names.indexOf("刘备"));

System.out.println(names.lastIndexOf("刘备"));

System.out.println(names.indexOf("张飞"));

System.out.println(names.lastIndexOf("张飞"));

根据元素索引位置进行的判断

if (names.indexOf("刘备")==0) {

System.out.println("刘备在这里");

}else if (names.lastIndexOf("刘备")==3) {

System.out.println("刘备在那里");

}else {

System.out.println("刘备到底在哪里？");

}

利用list中索引位置重新生成一个新的list（截取集合）

方法： .subList(fromIndex, toIndex)；　　.size() ； 该方法得到list中的元素数的和

List<String> phone=new ArrayList<>();

phone.add("三星"); //索引为0

phone.add("苹果"); //索引为1

phone.add("锤子"); //索引为2

phone.add("华为"); //索引为3

phone.add("小米"); //索引为4

//原list进行遍历

for(String pho:phone){

System.out.println(pho);

}

//生成新list

phone=phone.subList(1, 4); //.subList(fromIndex, toIndex) //利用索引1-4的对象重新生成一个list，但是不包含索引为4的元素，4-1=3

for (int i = 0; i < phone.size(); i++) { // phone.size() 该方法得到list中的元素数的和

System.out.println("新的list包含的元素是"+phone.get(i));

}

对比两个list中的所有元素

　　//两个相等对象的equals方法一定为true, 但两个hashcode相等的对象不一定是相等的对象

//1.<br>if (person.equals(fruits)) {

System.out.println("两个list中的所有元素相同");

}else {

System.out.println("两个list中的所有元素不一样");

}

//2.

if (person.hashCode()==fruits.hashCode()) {

System.out.println("我们相同");

}else {

System.out.println("我们不一样");

}

判断list是否为空

　　//空则返回true，非空则返回false

if (person.isEmpty()) {

System.out.println("空的");

}else {

System.out.println("不是空的");

}

###### 4.1.2 Map

HashMap

最常用的Map,它根据键的HashCode 值存储数据,根据键可以直接获取它的值，具有很快的访问速度。HashMap最多只允许一条记录的键为Null(多条会覆盖);允许多条记录的值为 Null。非同步的。

常用API

|  |  |
| --- | --- |
| clear() | 从 Map 中删除所有映射 |
| remove(Object key) | 从 Map 中删除键和关联的值 |
| put(Object key, Object value) | 将指定值与指定键相关联 |
| putAll(Map t) | 将指定 Map 中的所有映射复制到此 map |
| entrySet() | 返回 Map 中所包含映射的 Set 视图。Set 中的每个元素都是一个 Map.Entry 对象，可以使用 getKey() 和 getValue() 方法（还有一个 setValue() 方法）访问后者的键元素和值元素 |
| keySet() | 返回 Map 中所包含键的 Set 视图。删除 Set 中的元素还将删除 Map 中相应的映射（键和值） |
| values() | 返回 map 中所包含值的 Collection 视图。删除 Collection 中的元素还将删除 Map 中相应的映射（键和值） |
| get(Object key) | 返回与指定键关联的值 |
| containsKey(Object key) | 如果 Map 包含指定键的映射，则返回 true |
| containsValue(Object value) | 如果此 Map 将一个或多个键映射到指定值，则返回 true |
| isEmpty() | 如果 Map 不包含键-值映射，则返回 true |
| size() | 返回 Map 中的键-值映射的数目 |

##### 4.2 二元式文件结构

二元式文件通过专题1的词法分析程序得到：

其中一个测试用例为：

(8,if)

(3,()

(1,num1)

(24,>=)

(1,num2)

(3,))

(3,{)

(1,a)

(23,=)

(1,num1)

(14,+)

(1,num2)

(3,;)

(3,})

(9,else)

(3,{)

(3,;)

(3,})

其源程序为：

if (num1>=num2)

{

a = num1+num2;

}

else

{

;

}

二元式文件内容被录入到

**private** **static** List<String> *InputStream* = **new** ArrayList<String>(); //从二元式文件中拆解的符号穿输入流

二元式文件的录入和专题2一样，InputStream是一个Java List列表的一个对象，list列表是一系列的String类型的字符串，具体的操作：

br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(fp.getName())));

String erYuanShi = "";

**while**((erYuanShi=br.readLine())!=**null**) {

//截取符号串

*InputStream*.add(erYuanShi.substring(erYuanShi.indexOf(",") + 1, erYuanShi.lastIndexOf(")")));

}

*InputStream*.add("#"); //末尾添加#号

br 为一个文件的读入流，通过使用br.readLine()方法读入二元式文件当前行内容并返回给String类型的变量erYuanShi，然后每一行的内容类似为(1,num1)的形式，但是我们需要就是num1，所以通过erYuanShi.substring(erYuanShi.indexOf(",") + 1, erYuanShi.lastIndexOf(")"))方法将num1截取下来，放入List列表对象*InputStream*中，继续读文件，直到读取结束。这样就将二元式文件的内容读取到了List列表对象*InputStream中。*

然后每次分析当前符号的时候，都需要到列表对象*InputStream*中读取一个字符串进行分析，但是读取列表内容的时候，不能直接使用，因为，源程序和我们的文法有转换的地方，也就if被8替换，a表示&&等等，所以读取list内容之后，要通过一个函数进行转换：

**public** String get\_indexP(**int** index)

该函数要传入当前扫描指针位置，也就是indexP，根据indexP的位置，得到当前字符，然后该函数根据当前字符进行转换，返回转换后的字符。具体细节可以查看源程序或者该函数的批注。

##### 4.3文法

**private** Map<Character, String> grammar = **new** HashMap<Character, String>();

采用HashMap创建了grammar对象，保存文法，key为规则左补，value为规则右部。

需要手动录入：

//输入文法

grammar.put('P', "S|Q|;");

grammar.put('S', "V=E;");

grammar.put('E', "TR");

grammar.put('R', "ATR|"+*epsn*);

grammar.put('T', "FY");

grammar.put('Y', "MFY|"+*epsn*);

grammar.put('F', "CZ");

grammar.put('Z', "OCZ|"+*epsn*);

grammar.put('C', "BI");

grammar.put('I', "XBI|"+*epsn*);

grammar.put('B', "(E)|i");

grammar.put('A', "+|-");

grammar.put('M', "\*|/");

grammar.put('V', "i");

grammar.put('Q', "8JKH");

grammar.put('H', "fJKH|9K|"+*epsn*);

grammar.put('J', "(E)");

grammar.put('K', "S|{U}|;");

grammar.put('U', "PU|{U}U|"+*epsn*);

grammar.put('X', "a|o");

grammar.put('O', "t|d|g|l|u|e");

##### 4.4 非终极符号与终结符号

**private** List<Character> Vt = **new** ArrayList<Character>();

//非终结符号集

**private** List<Character> Vn = **new** ArrayList<Character>();

采用，List创建了Vn，Vt对象，Vn保存非终结符号，Vt保存终结符号：

需手动录入：

//添加非终结符号

Vn.add('P');

Vn.add('Q');

Vn.add('J');

Vn.add('K');

Vn.add('H');

Vn.add('S');

Vn.add('V');

Vn.add('E');

Vn.add('R');

Vn.add('T');

Vn.add('Y');

Vn.add('F');

Vn.add('A');

Vn.add('M');

Vn.add('Z');

Vn.add('C');

Vn.add('I');

Vn.add('B');

Vn.add('X');

Vn.add('O');

Vn.add('U');

//添加终结符号

Vt.add('8');

Vt.add('9');

Vt.add('f');

Vt.add('=');

Vt.add('i');

Vt.add('+');

Vt.add('-');

Vt.add('\*');

Vt.add('/');

Vt.add('(');

Vt.add(')');

Vt.add('{');

Vt.add('}');

Vt.add(';');

Vt.add('a');

Vt.add('o');

Vt.add('t');

Vt.add('d');

Vt.add('g');

Vt.add('l');

Vt.add('e');

Vt.add('u');

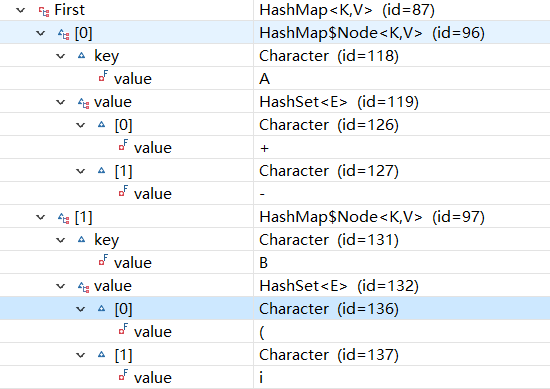
Vt.add(*end*);

Vt.add(*epsn*);

##### 4.5 FIRST 集

**private** Map<Character,Set<Character>> First = **new** HashMap<Character,Set<Character>>();

FIRST集采用HashMap的数据结构：



KEY为每个非终结符号，然后value是每个非终极符号FIRST集，

文法的FIRST集如下：

P:8 i ;

Q:8

J:(

K:i { ;

H:$ f 9

S:i

V:i

E:( i

R:$ + -

T:( i

Y:$ \* /

F:( i

A:+ -

M:\* /

Z:$ t d u e g l

C:( i

I:a $ o

B:( i

X:a o

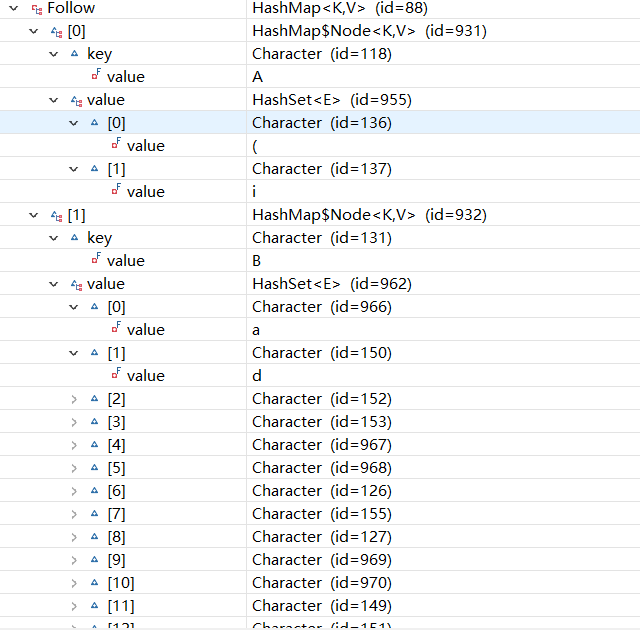
O:t d u e g l

U:$ 8 i ; {

##### 4.6 FOLLOW 集

**private** Map<Character,Set<Character>> Follow = **new** HashMap<Character,Set<Character>>();

FOLLOW集采用java HashMap的数据结构：



Key为非终结符号，value为对应非终结符号的FOLLOW集

非终结符号 FOLLOW集

P # 8 i ; { }

Q # 8 i ; { }

J i { ;

K # f 8 9 i ; { }

H # 8 i ; { }

S # f 8 9 i ; { }

V =

E ) ;

R ) ;

T ) + ; -

Y ) + ; -

F ) \* + ; - /

A ( i

M ( i

Z ) \* + ; - /

C t d u e g ) \* + ; l - /

I t d u e g ) \* + ; l - /

B a d e g ) \* + l - o / t u ;

X ( i

O ( i

U }

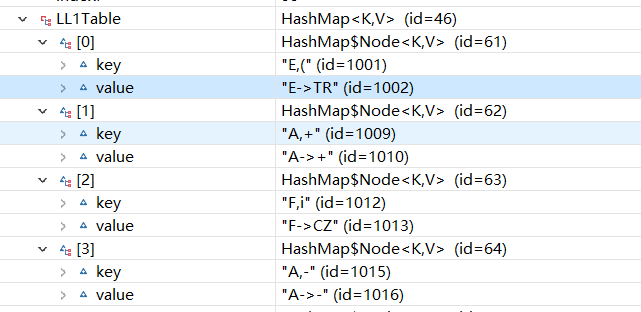
4.7 LL(1)分析表

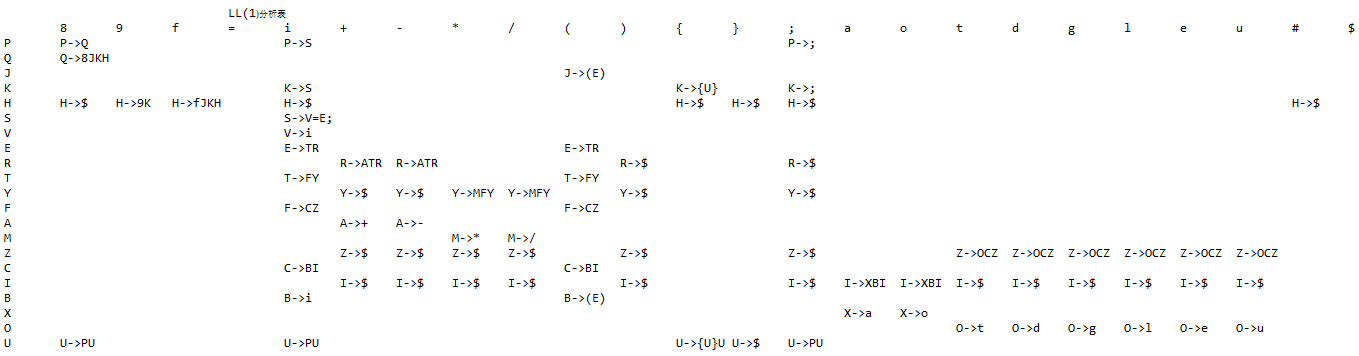
**private** Map<String,String> LL1Table = **new** HashMap<String,String>();

创建了key为string类型，value为string类型的HashMap的对象LL1Table

//LLR1Table的形式：key "E,("， value "E->TR"

如果"E,("在LL1Table中存在值，则表示LL(1)分析表中存在产生式：



最终的LL1分析表：

#### 5、程序结构描述

##### 5.1Java 主类：LL(1)语法分析主类LL1PredictionAnalysisMain

**public** **class** LL1PredictionAnalysisMain {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//FirstAndFollow faf = new FirstAndFollow();

//faf.displayGrammar();

//1、构造预测分析表

LL1PredictionAnalysis pa = **new** LL1PredictionAnalysis("zhuanti3\_2.tys");

//2、进行预测分析

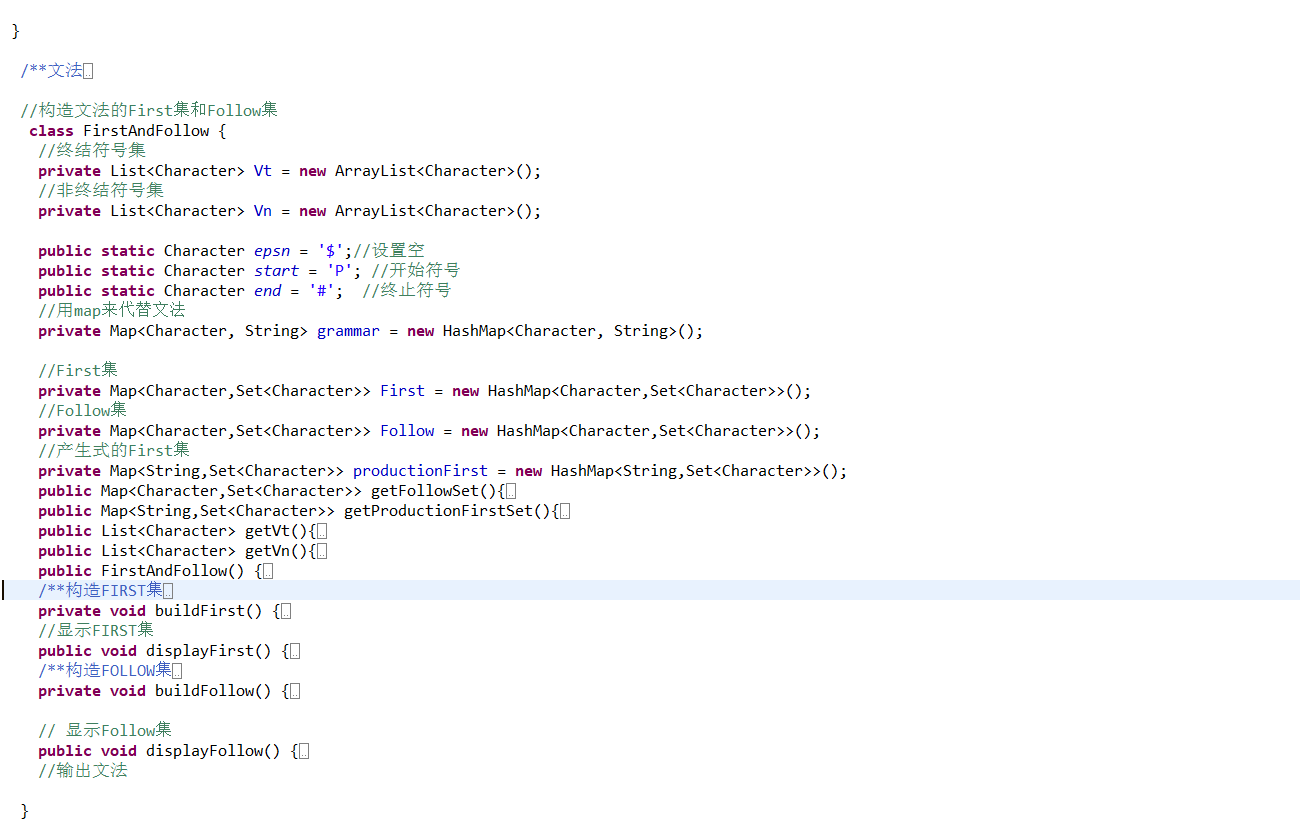
pa.analysisProcessing();

}

}

该类只有一个main函数，该函数创建LL(1)语法分析类LL1PredictionAnalysis的一个对象pa，通过pa对象调用语法分析入口函数pa.analysisProcessing();

##### 5.2 Java 副类：LL(1)语法分析副类类LL1PredictionAnalysis



###### 5.2.1私有变量：

|  |  |  |
| --- | --- | --- |
| **变量类型** | 变量名称 | 变量作用 |
| **private** BufferedReader | br | 输入的二元式文件流，作为读入二元式文件的读入流。 |
| **Private static** List<String> | *InputStream* | 保存二元式文件中的源程序。 |
| **private** **int** | indexP | *InputStream的*扫描指针，初始为0 |
| **private** Map<String,String> | LL1Table | LL(1)分析表，LLR1Table的形式：key "E,("， value "E->TR"  如果"E,("在LL1Table中存在值，则表示LL(1)分析表中存在产生式 |
| **private** FirstAndFollow | faf | FirstAndFollow类的一个对象，主要就是创建和保存文法G的FIRST集和FOLLOW集，通过创建一个FirstAndFollow类对象，会创建FIRST集和FOLLOW集 |
| **private** Stack<Character> | analysisStack | LL(1)分析器的分析栈 |
| **private** **int** | tab | 制符表的数量，初始为1 |

###### 5.2.2副类的构造函数

**public** LL1PredictionAnalysis(String fileName) {

File fp = **new** File(fileName);

**if**(!fp.getName().endsWith(".tys")) {

System.***out***.println("文件格式不正确...");

**return**;

}

//构造文件扫描

**try** {

br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(fp.getName())));

String erYuanShi = "";

**while**((erYuanShi=br.readLine())!=**null**) {

//截取符号串

*InputStream*.add(erYuanShi.substring(erYuanShi.indexOf(",") + 1, erYuanShi.lastIndexOf(")")));

}

*InputStream*.add("#"); //末尾添加#号

//输出一下序列

System.***out***.print("输入符号串为：");

**int** l=*InputStream*.size();

**for**(**int** i = 0;i<l-1;++i) {

print\_indexp(i);

}

System.***out***.println();

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

System.***out***.println(fileName+"文件不存在...");

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

//构造分析表

buildLLR1Table();

//显示分析表

displayLLR1Table();

//分析栈中移入终结符号和文法的开始符号

analysisStack.push(FirstAndFollow.*end*);

analysisStack.push(FirstAndFollow.*start*);

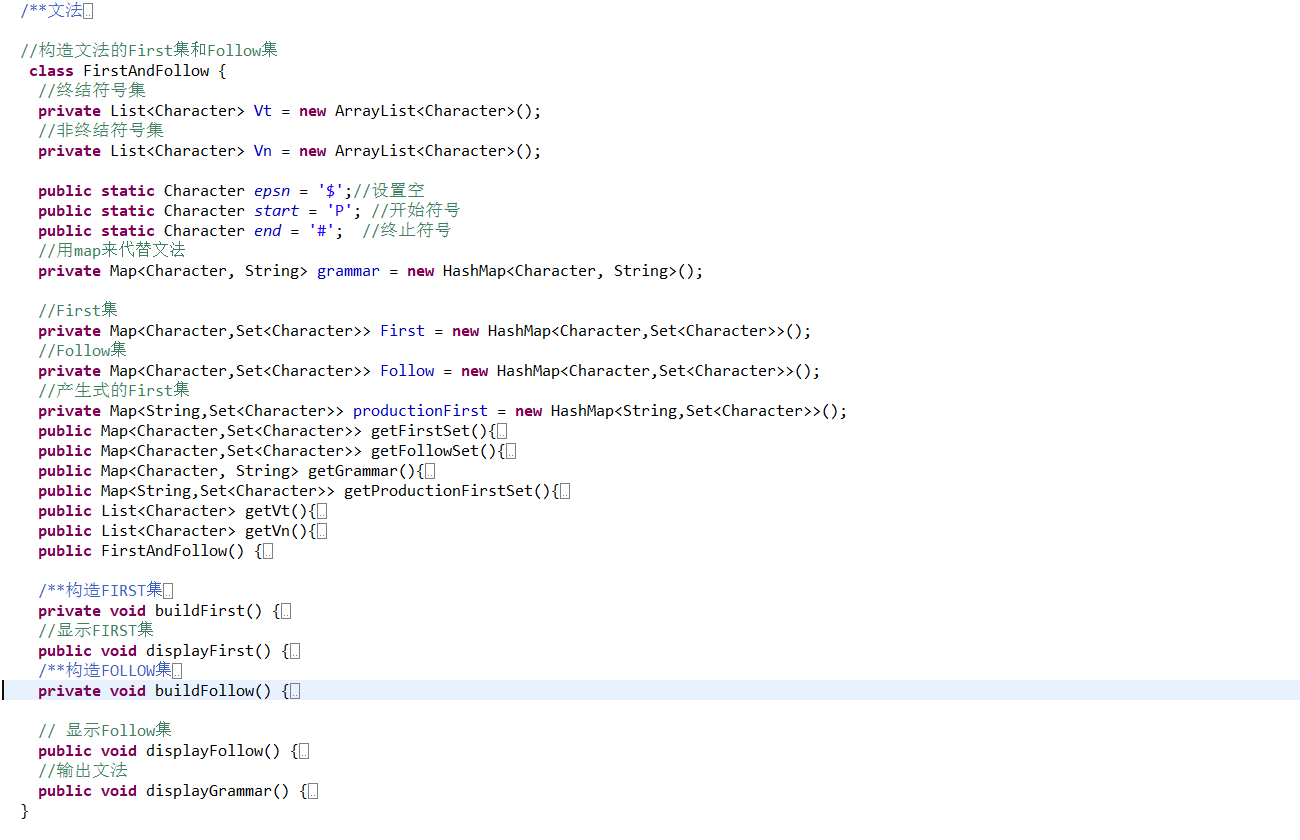
}

构造函数主要完成的就是录入二元式文件内容到*InputStream*，然后构造分析表，将分析表展示出来，构造函数就是为LL(1)分析器做准备工作。

###### 5.2.3副类的方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 返回值 | 函数名 | 形式参数 | 功能 |
| **public** | 无 | LL1PredictionAnalysis | 无 | 构造函数 |
| **private** | 无 | buildLLR1Table | 无 | 构造LL(1)分析表 |
| **private** | 无 | displayLLR1Table | 无 | 展示LL(1)分析表的矩阵形式 |
| **public** | 无 | analysisProcessing | 无 | LL(1)分析器的入口，根据分析栈，当前字符进行分析 |
| **private** | 无 | displayProcessing | String 传入当前产生式 | 输出LL(1)分析过程. |
| **public** | 无 | print\_indexp | Int 传入当前需要打印字符的扫描指针位置 | 打印带有缩进的当前字符 |
| **public** | String | get\_indexP | Int传入当前需要转换字符的扫描指针位置 | 根据当前字符进行转换得到文法能够识别的字符 |

##### 5.3 Java 副类 构造文法的First集和Follow集类FirstAndFollow



###### 5.3.1 私有变量

|  |  |  |
| --- | --- | --- |
| **变量类型** | 变量名称 | 变量作用 |
| **private** List<Character> | Vt | 保存终结符号集，需要手动设置 |
| **Private** List<Character> | Vn | 保存非终结符号集。需要手动设置 |
| **public** **static** Character | *epsn* | 终结符号ε |
| **public** **static** Character | *start* | 文法开始符号 |
| **public** **static** Character | *end* | 终结符号# |
| **private** Map<Character, String> | grammar | 保存文法，需要手动设置 |
| **private** Map<Character,Set<Character>> | First | 保存非终结符号的FIRST集，由函数生成内容 |
| **private** Map<Character,Set<Character>> | Follow | 保存非终结符号的FOLLOW集，有相应的生成函数生成其内容 |
| **private** Map<String,Set<Character>> | productionFirst | 保存产生式的FIRST集 |

###### 5.3.2构造函数FirstAndFollow()

**public** FirstAndFollow() {

//添加非终结符号

Vn.add('P');

Vn.add('Q');

Vn.add('J');

Vn.add('K');

Vn.add('H');

Vn.add('S');

Vn.add('V');

Vn.add('E');

Vn.add('R');

Vn.add('T');

Vn.add('Y');

Vn.add('F');

Vn.add('A');

Vn.add('M');

Vn.add('Z');

Vn.add('C');

Vn.add('I');

Vn.add('B');

Vn.add('X');

Vn.add('O');

Vn.add('U');

//添加终结符号

Vt.add('8');

Vt.add('9');

Vt.add('f');

Vt.add('=');

Vt.add('i');

Vt.add('+');

Vt.add('-');

Vt.add('\*');

Vt.add('/');

Vt.add('(');

Vt.add(')');

Vt.add('{');

Vt.add('}');

Vt.add(';');

Vt.add('a');

Vt.add('o');

Vt.add('t');

Vt.add('d');

Vt.add('g');

Vt.add('l');

Vt.add('e');

Vt.add('u');

Vt.add(*end*);

Vt.add(*epsn*);

//输入文法

grammar.put('P', "S|Q|;");

grammar.put('S', "V=E;");

grammar.put('E', "TR");

grammar.put('R', "ATR|"+*epsn*);

grammar.put('T', "FY");

grammar.put('Y', "MFY|"+*epsn*);

grammar.put('F', "CZ");

grammar.put('Z', "OCZ|"+*epsn*);

grammar.put('C', "BI");

grammar.put('I', "XBI|"+*epsn*);

grammar.put('B', "(E)|i");

grammar.put('A', "+|-");

grammar.put('M', "\*|/");

grammar.put('V', "i");

grammar.put('Q', "8JKH");

grammar.put('H', "fJKH|9K|"+*epsn*);

grammar.put('J', "(E)");

grammar.put('K', "S|{U}|;");

grammar.put('U', "PU|{U}U|"+*epsn*);

grammar.put('X', "a|o");

grammar.put('O', "t|d|g|l|u|e");

//非终结符号的First集和Follow集初始化

**for**(**int** i = 0;i<Vn.size();++i) {

First.put(Vn.get(i), **new** HashSet<Character>());

Follow.put(Vn.get(i), **new** HashSet<Character>());

}

//根据文法初始化productionFirst集

**for**(**int** i = 0;i<Vn.size();++i) {

String str = grammar.get(Vn.get(i));

String []nArryStr = str.split("\\|");

**for**(**int** j = 0;j<nArryStr.length;++j) {

productionFirst.put(Vn.get(i)+"->"+nArryStr[j], **new** HashSet<Character>());

}

}

//构造first集

**this**.buildFirst();

//显示first集

//this.displayFirst();

displayFirst();

//构造Follow集

**this**.buildFollow();

//显示Follow集

**this**.displayFollow();

}

构造函数主要完成的就是，Vn，Vt，grammar，初始化，以及FIRST,FOLLOW集初始，然后调用构造FIRST，FOLLOW集的函数，构造first，follow集，并把FIRST，FOLLOW集输出。

###### 5.3.3 FirstAndFollow副类的方法

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 返回值 | 函数名 | 功能 |
| **public** | 无 | FirstAndFollow | 构造函数 |
| **public** | Map<Character,Set<Character>> | getFollowSet | 返回私有变量grammar集 |
| **public** | Map<String,Set<Character>> | getProductionFirstSet() | 返回私有变量的productionFirst集 |
| **public** | List<Character> | getVt | 返回私有变量Vt集 |
| **public** | List<Character> | getVn | 返回私有变量Vn集 |
| **public** | **void** | buildFirst | 构造FIRST集 |
| **private** | **void** | displayFirst | 输出FIRST集 |
| **public** | **void** | buildFollow | 构造FOLLOW集 |
| **private** | **void** | displayFollow | 输出FOLLOW集 |

##### 5.4 JAVA类之间关系

###### 5.4.1 类之间的关系



FirstAndFollow类构造出FIRST集和FOLLOW集后，LL1PredictionAnalysis类通过**private** FirstAndFollow faf = **new** FirstAndFollow(); //调用文法G的FIRST集FOLLOW集，然后通过FIRST集和FOLLOW集进行LL(1)分析表，然后LL1PredictionAnalysisMain类通过

LL1PredictionAnalysis pa = **new** LL1PredictionAnalysis("zhuanti3\_2.tys");

//2、进行预测分析

pa.analysisProcessing();

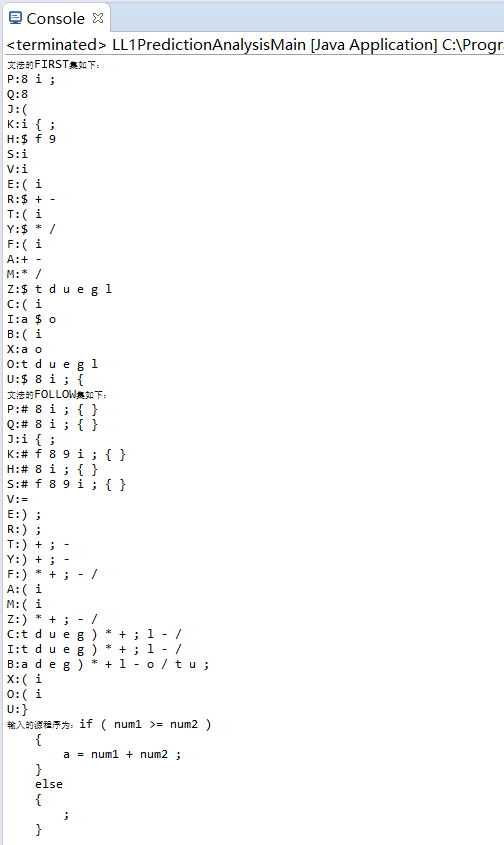
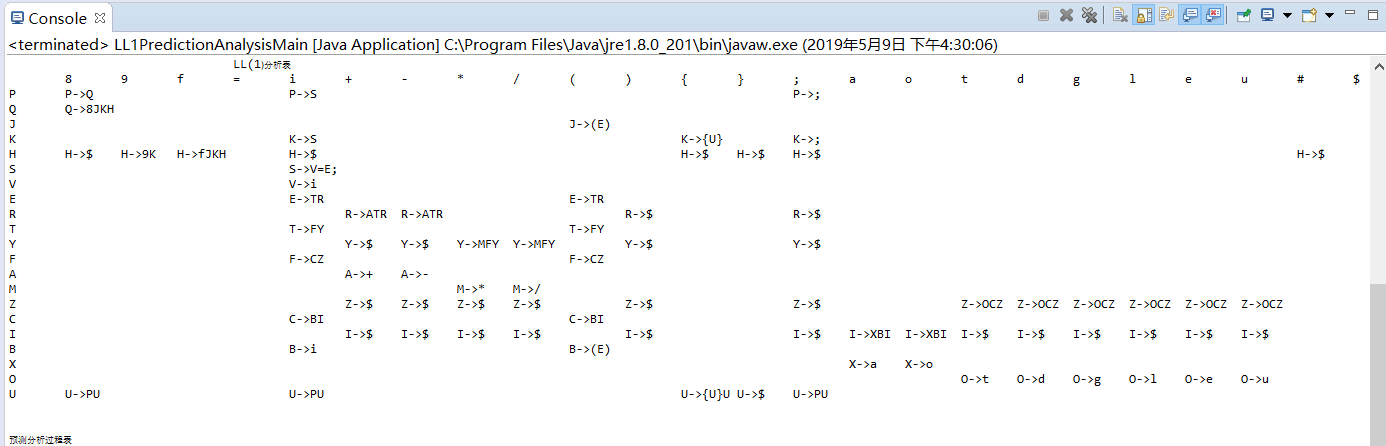
调用LL1PredictionAnalysis类创建一个对象pa，通过对象pa.analysisProcessing()调用LL(1)分析器函数，进行分析，得到结果。

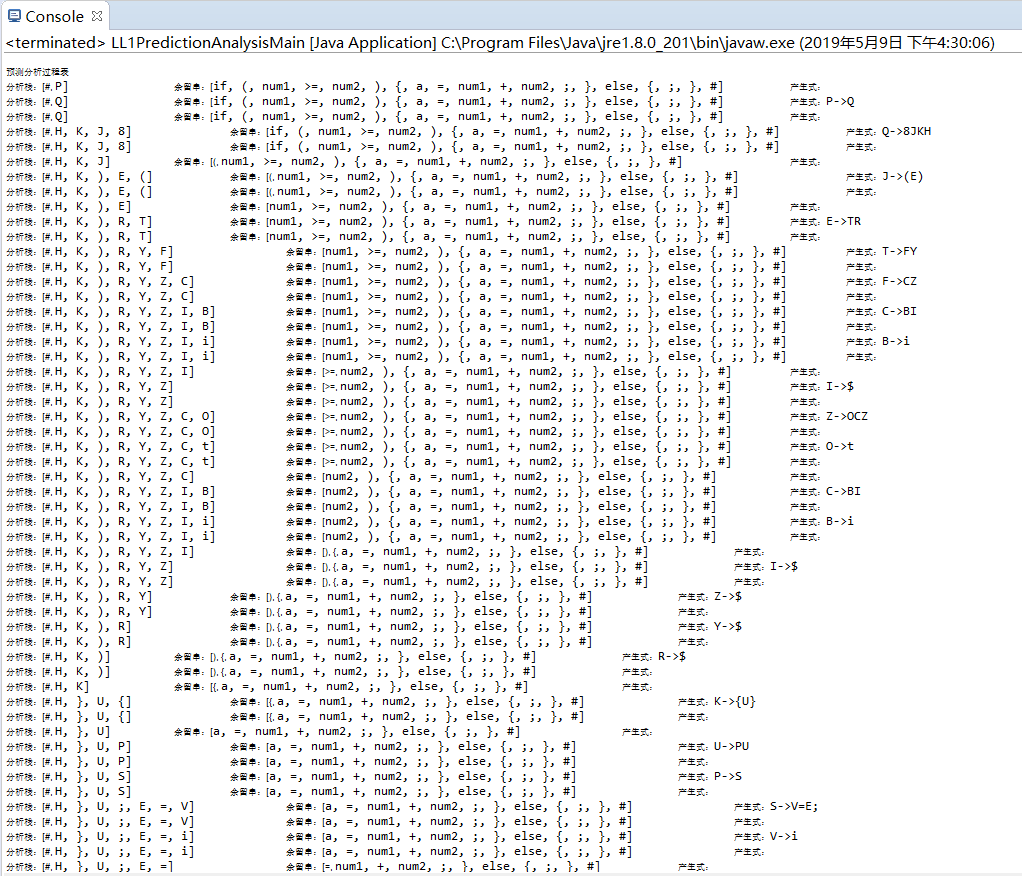
#### 6、程序测试

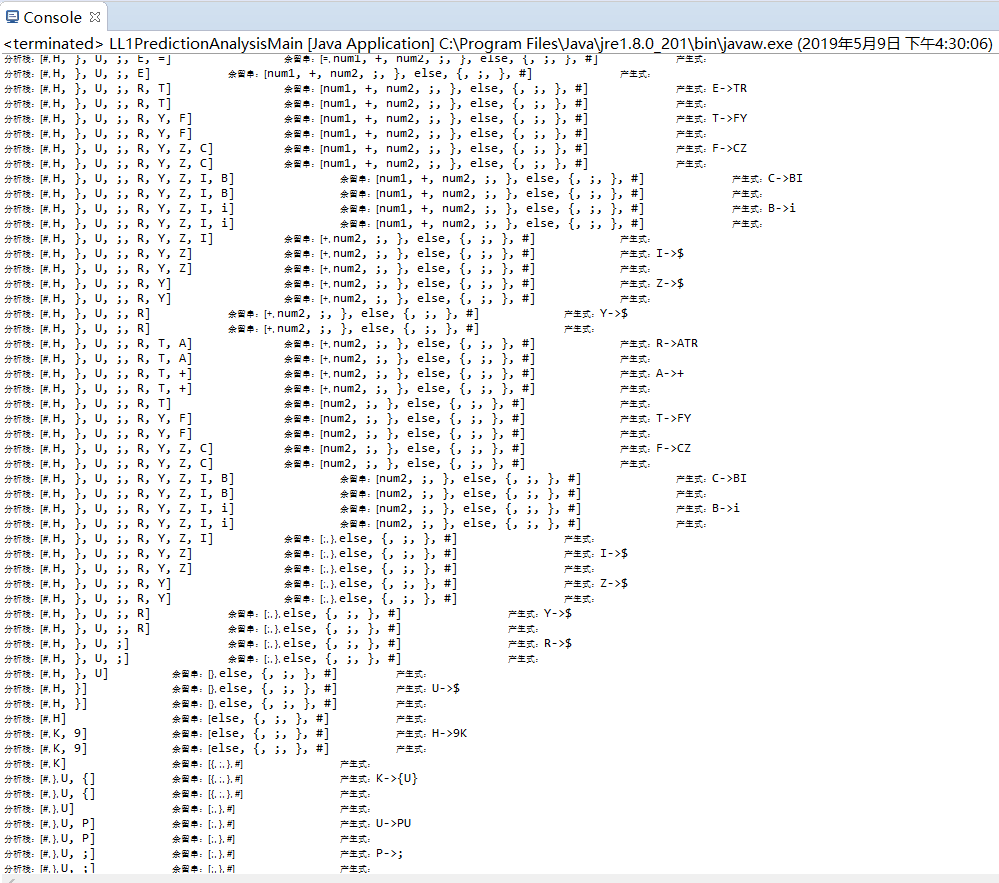
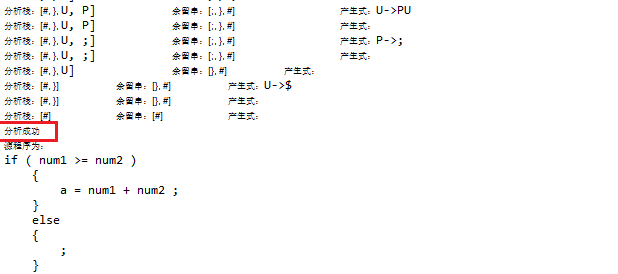
##### 6.1 正确用例

**测试用例为**[**二元式文件结构部分的用例**](#_4.2_二元式文件结构)

**结果为：**



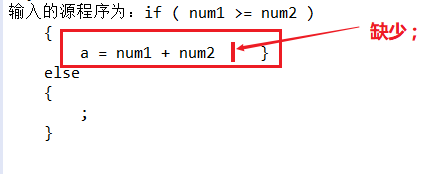




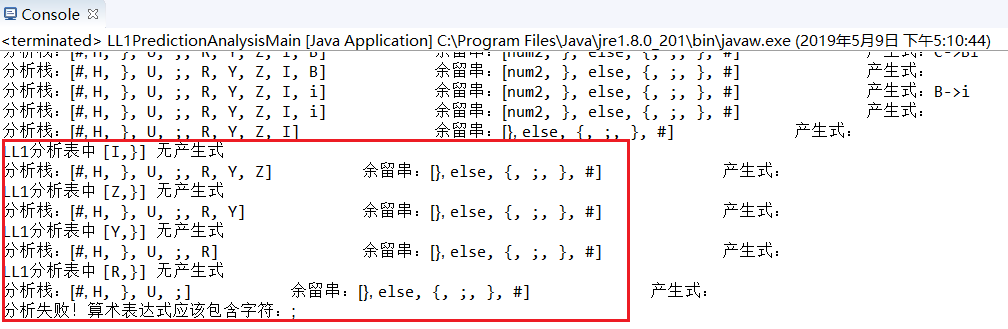
##### 6.2 错误用例

通过修改测试用例1使其不符合if语句语法：

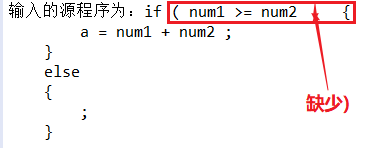
###### 6.2.1缺少 ‘；’符号



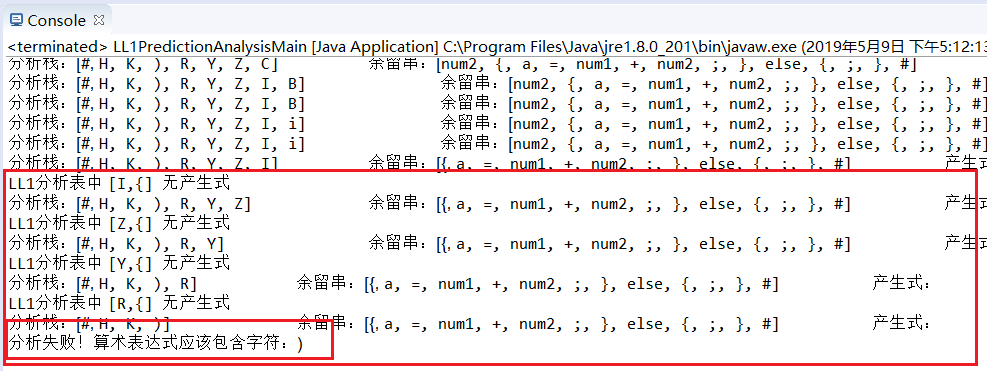
错误提示：



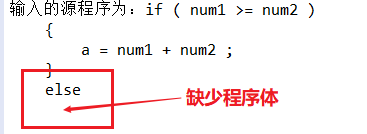
###### 6.2.1缺少 ‘）’符号



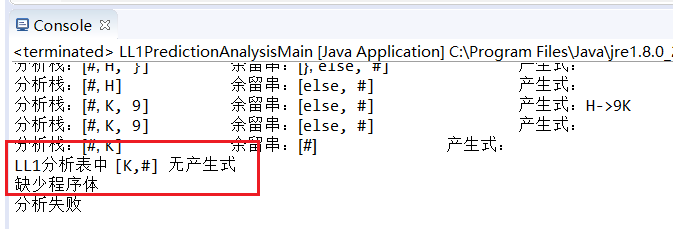
错误提示：



###### 6.2.3缺少if语句程序体:



错误提示：



### 实验过程总结

#### 1、LL(1)分析器分析步骤

##### 1.1 FIRST集

构造FIRST集

反复利用如下规则，直至FIRST集不再增大

（1）若X属于Vt,则FIRST(X)={X};

（2）若X属于Vn,且有X->aN(a属于Vt),则令a属于FIRST(X);若有X->$,则$属于FIRST(X);

（3）若X->Y1Y2...Yk,

ⅰ 将FIRST(Y1)中的一切非$的终结符加进FIRST(X);

ⅱ 若$属于FIRST(Y1),则将FIRST(Y2)中的一切非$的终结符加进FIRST(X);

ⅲ 若$属于FIRST(Y1),并且$属于FIRST(Y2),则将FIRST(Y3)中的一切非$终结符加进FIRST(X);一次类推

ⅲi 若$都属于FIRST(Y1...YN),则将$加进FIRST(X)

##### 1.2 FOLLOW集

构造FOLLOW集

1. 令# ∈FOLLOW(S) S为文法开始符号

②对A→ αBβ, 且β ≠ ε则

将 FIRST(β) -{ε}加入FOLLOW(B)中

1. 反复, 直至每一个FOLLOW(A)不再增大

对A→ αB或A→ αBβ(且ε ∈ FIRST(β)) 则FOLLOW(A)中的全部元素加入FOLLOW(B)

##### 1.3 LL(1)分析表构建

由每一个产生式A → α1︱ α2︱……. ︱ αn

确定M[A,a]矩阵 a ∈ Vt

1. 任何a ∈ FIRST(α) , 置M[A, a]=“pop,push(α′)”

α′为 α倒置 或将A → α 规则填入M[A, a]

② 若ε ∈ FIRST(α) ,

则对于任一个b ∈FOLLOW(A) b ∈ Vt或#

置M[A, b]=“pop”或将A → ε规则填入M[A, b ]

 此时 b不属于FIRST(A)

1. 其它空白为出错

##### 1.4 LL(1)分析过程

预测分析程序的总控程序在任何时候都是按STACK栈顶符号X和当前的输入符号行事的，对于任何(X,a),总控程序

每次都执行下述三种可能的动作之一;

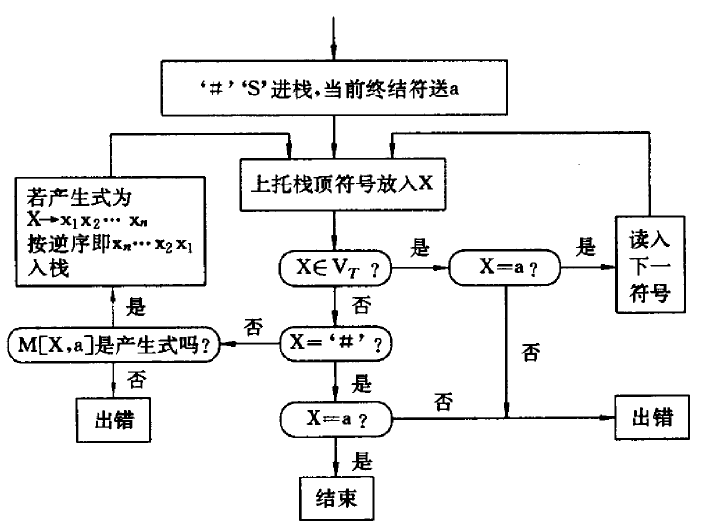
(1) 若X=a=”#”,则宣布分析成功,停止分析过程.

(2) 若X=a≠”#”,则把X从STACK栈顶逐出,让a指向下一个输入符号.

(3) 若X是一个非终结符,则查看分析表M,若M[A,a]中存放着关于X的一个产生式,那么,首先把X逐出STACK栈顶,然后

把产生式的右部符号串按反序一一推进STACK栈(若右部符号为ε,则意味着不推什么东西进栈).在把产生式的右部符号推进栈的同时应做这个产生式相应得语义动作,若M[A,a]中如果不存在,则调用出错诊察程序ERROR.

流程图：



#### ****2、 实验输出分析过程的完善，print\_indexp(int i)****

该函数进行了对输出分析过程的格式完善，也就是怎么将二元式文件输出为源程序，输出的时候不仅仅只对源程序考虑为一个字符串进行输出，还要注意到了缩进，也就是程序的对齐，保证其可读性。大致结果就是如下：

***if ( ( ( num1 >= num2 ) && ( num1 + num2 != 23 ) ) || a >= num1 )***

***{***

***a = num1 + num2 ;***

***if ( a >= num1 )***

***{***

***;***

***}***

***else***

***{***

***a = num1 + num2 ;***

***}***

***a = num1 + num2 ;***

***}***

***else if ( num1 >= num2 )***

***{***

***if ( a >= num1 )***

***{***

***a = num1 + num2 ;***

***}***

***else***

***{***

***a = num1 + num2 ;***

***}***

***}***

***else if ( num1 >= num2 ) ;***

***else***

***{***

***{***

***if ( a >= num1 ) a = num1 + num2 ;***

***else***

***{***

***a = num1 + num2 ;***

***}***

***}***

***}***

而不是输出一个字符串，可读性非常差，例如下面这样：

***if(((num1>=num2)&&(num1+num2!=23))||a>=num1) { a = num1+num2; if(a>=num1) { ; } else { a = num1+num2; } a = num1+num2; } else if (num1>=num2) { if(a>=num1) { a = num1+num2; } else { a = num1+num2; } } else if (num1>=num2) ; else { { if(a>=num1) a = num1+num2; else { a = num1+num2; } } }***

**public** **void** print\_indexp(**int** i) 根据当前字符的情况，进行分析，是否进行格式的输出，主要控制格式的就是tab和换行符，函数根据当前字符的情况，分析出应当先输出几个tab或是否应该输出\n换行符。**private** **int** tab = 1; //制符表的数量，初始为1的作用就是管理tab数量的一个私有变量。

举个简单的例子，遇到’{‘符号，我们要先考虑当前的tab数量应该输出多少，并且要输出一个\n换行符，然后要改变一下tab的数量，因为按照习惯，遇到一个’{’符号后，在输入语句的时候，我们会进行一个相对tab的输入，所以tab要++。同理 ‘}’要考虑当前的tab数量应该输出多少，并且要输出一个\n换行符，然后要改变一下tab的数量，但是tab要--。还有很多具体的情况要具体对待，具体的细节可以查看批注或者源程序。