

《编译原理》研究性学习专题实验报告

实验4： 算符优先语法分析设计原理与实现技术

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06月02日

目录

[实验要求： 3](#_Toc10407893)

[1.[实验项目] 3](#_Toc10407894)

[2.[设计说明] 4](#_Toc10407895)

[3.[设计要求] 4](#_Toc10407896)

[实验过程： 4](#_Toc10407897)

[1、算符优先分析程序设计说明 4](#_Toc10407898)

[1.1设计要求： 4](#_Toc10407899)

[1.2设计说明： 5](#_Toc10407900)

[3、程序功能描述 5](#_Toc10407901)

[4、主要的数据结构描述 6](#_Toc10407902)

[4.1主要使用的java数据结构类型 6](#_Toc10407903)

[4.1.1 List 6](#_Toc10407904)

[List添加元素 6](#_Toc10407905)

[list中是否包含某个元素 6](#_Toc10407906)

[list获取长度： 7](#_Toc10407907)

[list中查看（判断）元素的索引 7](#_Toc10407908)

[根据元素索引位置进行的判断 7](#_Toc10407909)

[判断list是否为空 8](#_Toc10407910)

[4.1.2 Map 8](#_Toc10407911)

[HashMap 8](#_Toc10407912)

[常用API 8](#_Toc10407913)

[Map添加元素 9](#_Toc10407914)

[Map的遍历 9](#_Toc10407915)

[Map获取元素 10](#_Toc10407916)

[4.2 二元式文件结构 11](#_Toc10407917)

[4.5 FIRSTVT 集 12](#_Toc10407918)

[4.6 LASTVT 集 15](#_Toc10407919)

[4.7 算符优先矩阵 17](#_Toc10407920)

[5、程序结构描述 20](#_Toc10407921)

[5.1Java 主类：OPGMain 20](#_Toc10407922)

[5.1.1全局变量 20](#_Toc10407923)

[5.1.2函数 21](#_Toc10407924)

[5.1.3函数调用关系图 22](#_Toc10407925)

[6、程序测试 22](#_Toc10407926)

[6.1 正确用例 22](#_Toc10407927)

[6.2 错误用例 24](#_Toc10407928)

[6.2.1缺少操作符 24](#_Toc10407929)

[6.2.2缺少 ‘）’符号 25](#_Toc10407930)

### 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[E]:

E →E+T∣E-T∣T

T→T\*F∣T/F∣F

F→(E)∣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）构造该算符优先文法的优先关系矩阵或优先函数；

（2）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。

（3）算符优先分析过程应能发现输入串出错。

（4）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；

（5）考虑编写程序根据算符优先文法构造算符优先关系矩阵，并添加到你的算符优先分析程序中。

### 实验过程：

#### 1、**算符优先分析程序设计说明**

##### 1.1设计要求：

（1）构造该算符优先文法的优先关系矩阵或优先函数；

（2）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。

（3）算符优先分析过程应能发现输入串出错。

（4）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；

（5）考虑编写程序根据算符优先文法构造算符优先关系矩阵，并添加到你的算符优先分析程序中。

##### 1.2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）根据FIRSTVT集和LASTVE集的构造规则，构造出文法的FIRSTVT集和LASTVT集

（5）根据FIRSTVT集和LASTVT集构造算符优先矩阵；

（6）根据分析规则，构造算符优先矩阵分析器。

#### 3、**程序功能描述**

（1）、能够录入一个.tys文件中的二元式内容；二元式内容为表达式

（2）、根据.tys文件内容进行算符优先分析，可识别

（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FIRSTVT集和LASTVT集；

（6）、根据构造FIRSTVT集和LASTVT集，构造出算符优先矩阵分析表；

（7）、算符优先矩阵分析器。

#### 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

//终结符号集

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

//非终结符号集

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

使用List保存文法的非终结符号和终结符号，一旦文法给定，非终结符号集合和终结符号集合也就确定，所以对Vt、Vn的操作一般是查找元素和遍历元素。

List添加元素

//设置非终结符号

Vn.add('S');

Vn.add('E');

Vn.add('T');

Vn.add('F');

//设置终结符号

Vt.add('#');

Vt.add('+');

Vt.add('-');

Vt.add('\*');

Vt.add('/');

Vt.add('(');

Vt.add(')');

Vt.add('i');

使用add()方法即可添加元素，上面的程序保存了Vt、Vn集合。

list中是否包含某个元素

方法：.contains（Object o）； 返回true或者false

例如下面在程序中使用到的：

Vn.contains(nArryStr[i].charAt(0))&&Vt.contains(nArryStr[i].charAt(1))

list获取长度：

Vt.size()

使用size()即可返回List的长度

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？");

}

在程序中使用OPGtable[Vt.indexOf(Vt1)][Vt.indexOf(Vt2)]

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 |

在程序中使用了MAP:

//文法

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

//FIRSTVT集

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

//LASTVT集

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

使用map保存文法，和FIRSTVT,LASTVT。

Map添加元素

grammar.put("S", "#E#");

grammar.put("E", "E+T|E-T|T");

grammar.put("T", "T\*F|T/F|F");

grammar.put("F", "(E)|i");

上面程序将题目要求的文法保存到map对象grammar中

Map的遍历

grammar.forEach((k,v)->{

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

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

{

//System.out.println(nArryStr[i]);

**if**(Vt.contains(nArryStr[i].charAt(0)))

{

**char** b= nArryStr[i].charAt(0);

FIRSTVT.get(k.charAt(0)).add(b);

}

**if**(nArryStr[i].length()>=2)

**if**(Vn.contains(nArryStr[i].charAt(0))&&Vt.contains(nArryStr[i].charAt(1)))

{

**char** b=nArryStr[i].charAt(1);

FIRSTVT.get(k.charAt(0)).add(b);

}

}

});

使用forEach方法，参数(k,v)

其中k为每个values的索引值，通过forEach的方法，遍历map的所有元素，就返回每个元素的key和values，并赋给参数（k,v）。

Map获取元素

Map获取元素一般根据key值获取对应的values：

例如：

LASTVT.get(k.charAt(0))

结果返回的是，key对应的values，而且values是声明map时设置的类型对应，

比如：

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

key的类型为Character

values的类型为Set<Character>

所以通过LASTVT.get(key)获得values时，得到类型为Set<Character>。

##### 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：

(1,a)

(4,+)

(1,b)

(10,\*)

(3,()

(1,c)

(5,-)

(2,34)

(3,))

(11,/)

(1,num5)

二元式文件内容被录入到

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

String erYuanShi = "";

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

//截取符号串

String substr=erYuanShi.substring(erYuanShi.indexOf("(") + 1, erYuanShi.lastIndexOf(","));

**if**(substr.equals("1")||substr.equals("2"))

{

s+="i";

}

**else**

{

s+=erYuanShi.substring(erYuanShi.indexOf(",") + 1, erYuanShi.lastIndexOf(")"));

}

}

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字符串s中。1 和2表示标识符和数字在符号表中的序号，也即1表示当前二元式的实际内容为标识符，2表示为数字，所以如果是标识符和数字时，就将内容转换为i保存到s中。

最后得到字符串s类似为: i+i\*(i-i)/i

但是实际算术表达式为：a+a\*(b-c)/2。

也就是将用户定义的标识符和数字都转为i。

##### 4.5 FIRSTVT 集

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

根据构造规则：

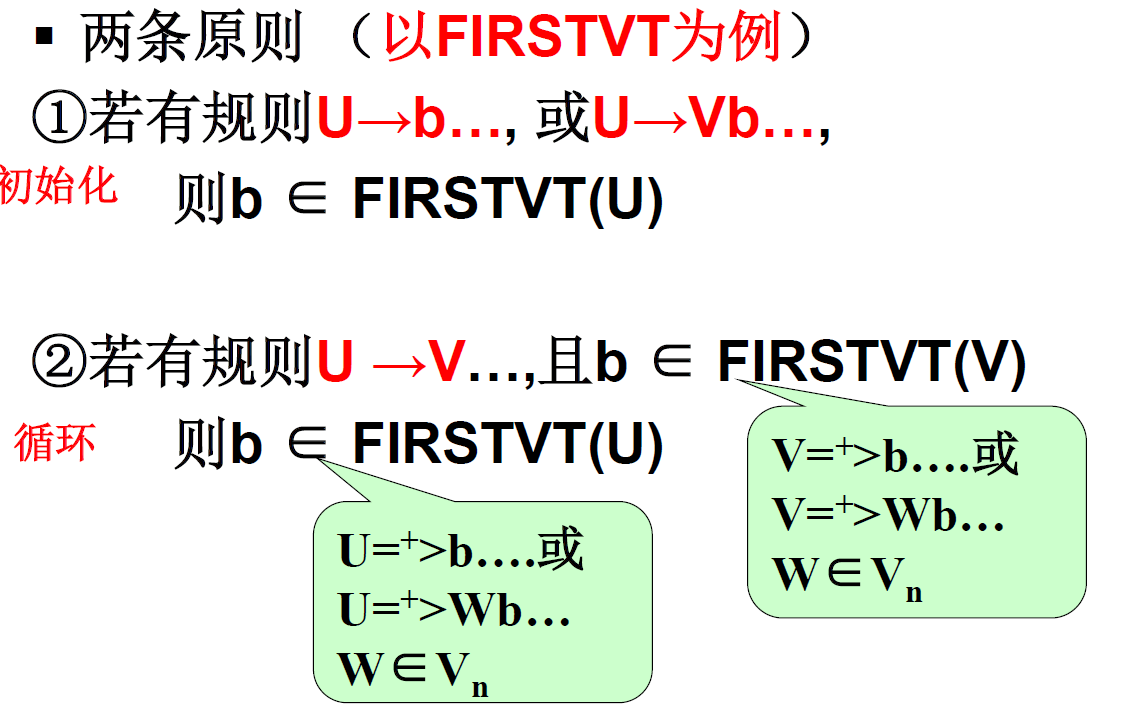
U∈Vn

FIRSTVT(U)=

{b∣U=+>b…, 或U =+> Vb…, b∈Vt,V∈Vn }

则形如W→…aU…的规则 a < b

b ∈ FIRSTVT(U)



编写代码：

//遍历文法。进行非终结符号的FIRSTVT的初始化

grammar.forEach((k,v)->{

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

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

{

//System.out.println(nArryStr[i]);

**if**(Vt.contains(nArryStr[i].charAt(0)))

{

**char** b= nArryStr[i].charAt(0);

FIRSTVT.get(k.charAt(0)).add(b);

}

**if**(nArryStr[i].length()>=2)

**if**(Vn.contains(nArryStr[i].charAt(0))&&Vt.contains(nArryStr[i].charAt(1)))

{

**char** b=nArryStr[i].charAt(1);

FIRSTVT.get(k.charAt(0)).add(b);

}

}

});

**do**

{

FIRSTVTAllNochange=1;

grammar.forEach((k,v)->{

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

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

{

**char** U=k.charAt(0);

**char** V=nArryStr[i].charAt(0);

**if**(Vn.contains(U)&&U!=V&&Vn.contains(V))

{

FIRSTVT.get(V).forEach(values->{

**if**(!FIRSTVT.get(U).contains(values))

{

// System.out.println(values);

FIRSTVT.get(U).add(values);

FIRSTVTAllNochange = 0;

}

});

}

}

});

**if**(FIRSTVTAllNochange==1)

**break**;

}**while**(**true**);

获得FIRSTVT集：

S:[#]

T:[(, i, \*, /]

E:[(, i, \*, +, -, /]

F:[(, i]

##### 4.6 LASTVT 集

//LASTVT集 **private** Map<Character,Set<Character>> LASTVT = **new** HashMap<Character,Set<Character>>();

LASTVT构造规则：

U∈Vn

LASTVT(U)=

{a∣U=+>…a, 或U =+>…aV, a∈Vt, V ∈Vn }

则形如W→…Ub…的规则 a > b

a ∈LASTVT(U)

编写代码：

//遍历文法。进行非终结符号的LASTVT的初始化

grammar.forEach((k,v)->{

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

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

{

//System.out.println(nArryStr[i]);

**int** len= nArryStr[i].length();

**if**(Vt.contains(nArryStr[i].charAt(len-1)))

{

**char** b= nArryStr[i].charAt(len-1);

LASTVT.get(k.charAt(0)).add(b);

}

**if**(nArryStr[i].length()>=2)

**if**(Vt.contains(nArryStr[i].charAt(len-2))&&Vn.contains(nArryStr[i].charAt(len-1)))

{

**char** b=nArryStr[i].charAt(len-2);

LASTVT.get(k.charAt(0)).add(b);

}

}

});

**do**

{

FIRSTVTAllNochange=1;

grammar.forEach((k,v)->{

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

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

{

**int** len =nArryStr[i].length();

**char** U=k.charAt(0);

**char** V=nArryStr[i].charAt(len-1);

**if**(Vn.contains(U)&&U!=V&&Vn.contains(V))

{

LASTVT.get(V).forEach(values->{

**if**(!LASTVT.get(U).contains(values))

{

//System.out.println(values);

LASTVT.get(U).add(values);

FIRSTVTAllNochange = 0;

}

});

}

}

});

**if**(FIRSTVTAllNochange==1)

**break**;

}**while**(**true**);

最后得到LASTVT集：

S:[#]

T:[), i, \*, /]

E:[), i, \*, +, -, /]

F:[), i]

##### 4.7 算符优先矩阵

/算符优先权关系表

/\*\*

\* 在OPGtable中，用-1，0,1,2表示优先权关系

\* 0 表示优先关系等于

\* 1 表示优先关系小于

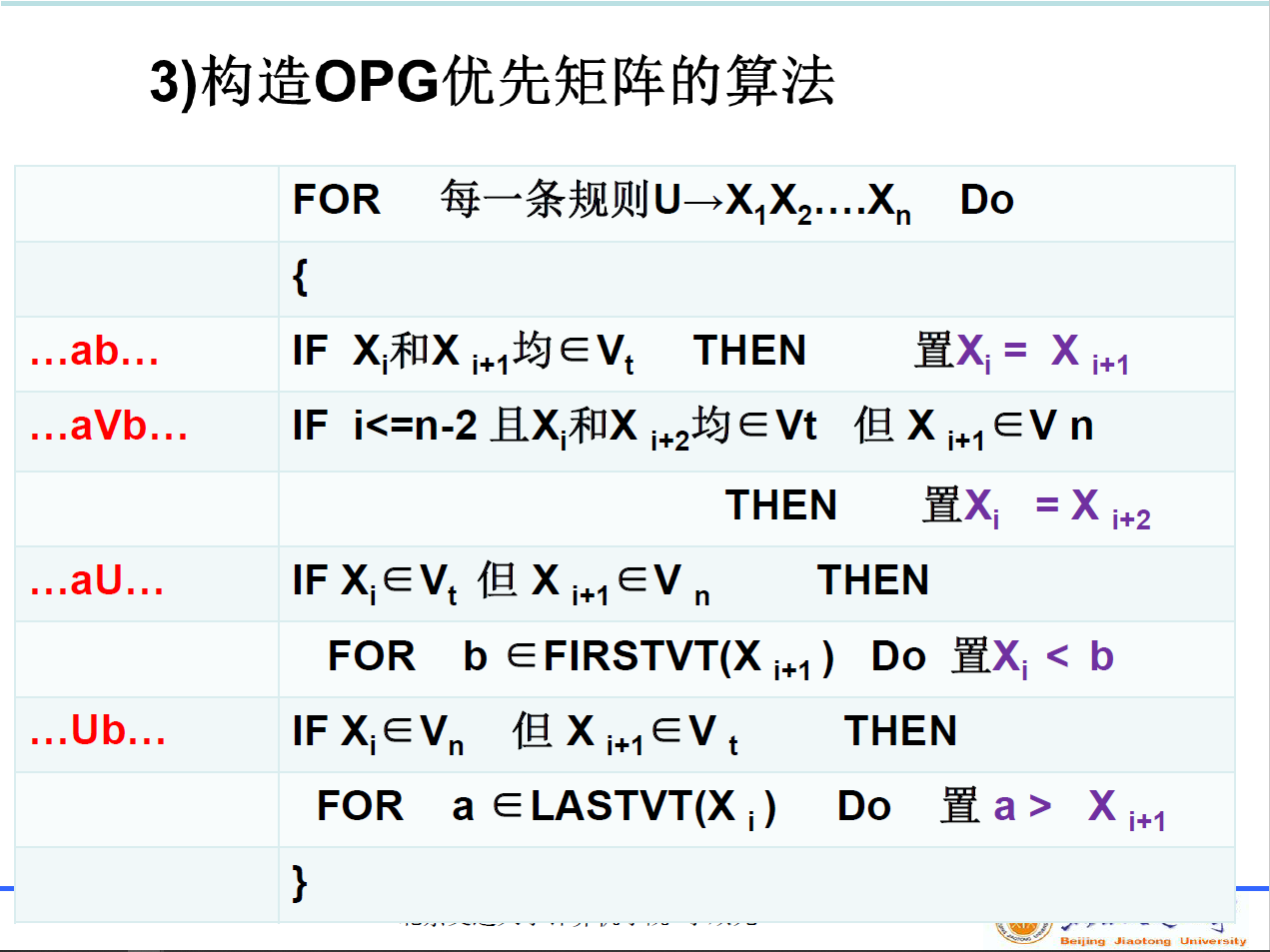
\* 2 表示优先关系大于

\* -1 表示不存在优先权关系

\* \*

\*/

**private** **int** [][]OPGtable = **new** **int** [N][N];

构造规则：

编写代码：  
 /\*\*

\* 在OPGtable中，用-1，0,1,2表示优先权关系

\* 0 表示优先关系等于

\* 1 表示优先关系小于

\* 2 表示优先关系大于

\* -1 表示不存在优先权关系

\* \*

\*/

grammar.forEach((k,v)->{

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

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

{

String ruleRight=nArryStr[i];

**int** len = ruleRight.length();

**if**(len>=2)

{

**for**(**int** i1= 0 ;i1<len-1 ;i1++)

{

**char** X1=ruleRight.charAt(i1);

**char** X2=ruleRight.charAt(i1+1);

**if**(Vt.contains(X1)&&Vt.contains(X2))

OPGtable[Vt.indexOf(X1)][Vt.indexOf(X2)]=0;//0 表示优先关系等于

**if**(Vt.contains(X1)&&Vn.contains(X2))

{

FIRSTVT.get(X2).forEach(values->{

OPGtable[Vt.indexOf(X1)][Vt.indexOf(values)]=1;//1 表示优先关系小于

});

}

**if**(Vn.contains(X1)&&Vt.contains(X2))

{

LASTVT.get(X1).forEach(values->{

OPGtable[Vt.indexOf(values)][Vt.indexOf(X2)]=2;//2 表示优先关系大于

});

}

**if**(len>=3&&i1<len-2)

{

**char** X3=ruleRight.charAt(i1+2);

**if**(Vt.contains(X1)&&Vn.contains(X2)&&Vt.contains(X3))

{

OPGtable[Vt.indexOf(X1)][Vt.indexOf(X3)]=0; //0 表示优先关系等于

}

}

}

}

}

});

最终的算符优先矩阵：

# + - \* / ( ) i

# =· <· <· <· <· <· <·

+ >· >· >· <· <· <· >· <·

- >· >· >· <· <· <· >· <·

\* >· >· >· >· >· <· >· <·

/ >· >· >· >· >· <· >· <·

( <· <· <· <· <· =· <·

) >· >· >· >· >· >·

i >· >· >· >· >· >·

#### 5、程序结构描述

##### 5.1Java 主类：OPGMain



###### 5.1.1全局变量

|  |  |  |
| --- | --- | --- |
| **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 |
| **private** Map<Character, String> | grammar | 保存文法，需要手动设置 |
| **private** Map<Character,Set<Character>> | FIRSTVT | 保存非终结符号的FIRSTVT集，由函数生成内容 |
| **private** Map<Character,Set<Character>> | LASTVT | 保存非终结符号的LASTVT集，有相应的生成函数生成其内容 |
| **private** Map<String,Set<Character>> | OPGtable | //算符优先权关系矩阵  /\*\*  \* 在OPGtable中，用-1，0,1,2表示优先权关系  \* 0 表示优先关系等于  \* 1 表示优先关系小于  \* 2 表示优先关系大于  \* -1 表示不存在优先权关系  \* \*  \*/ |

###### 5.1.2函数

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 返回值 | 函数名 | 功能 |
| **public** | 无 | OPGMain | 构造函数，初始化文法，Vt和Vn |
| **public** | **void** | Create\_FIRSTVT | 构造FIRSTVT集 |
| **public** | **void** | Create\_LASTVT | 输出LASTVT集 |
| **public** | **void** | Create\_OPGTable | 构造算符优先关系矩阵 |
| **public** | **void** | OPGanalysis | 算符优先文法分析器 |
| **public** | **int** | get\_PriorityRelationship | 获得两个终结符号的优先关系 |
| **public** | **static** **void** | main | 类的主函数，实例化主类，并传入中间文件。 |

###### 5.1.3函数调用关系图

绿色虚线为调用

#### 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)

(1,a)

(4,+)

(1,b)

(10,\*)

(3,()

(1,c)

(5,-)

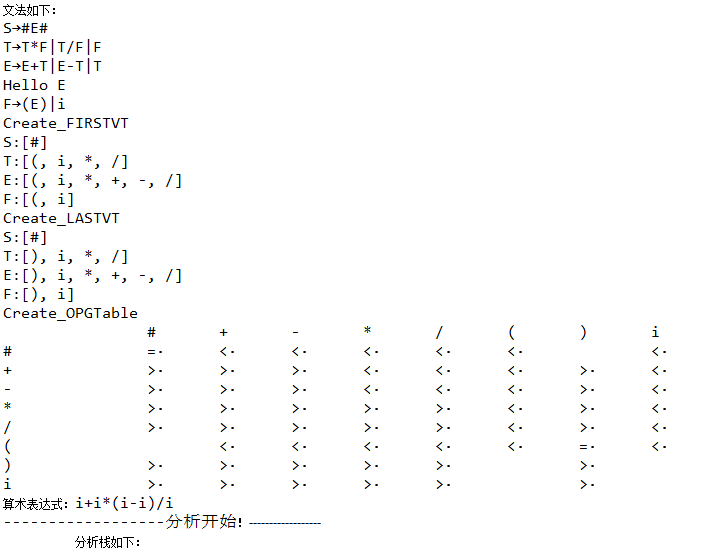
(2,34)

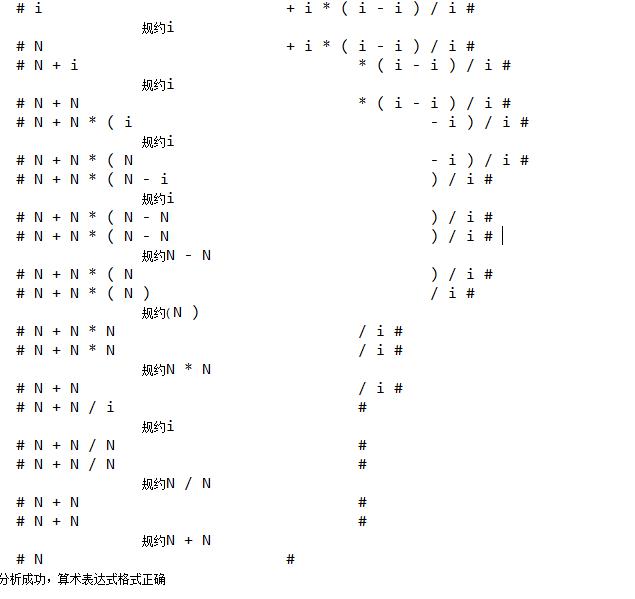
(3,))

(11,/)

(1,num5)

**结果为：**





##### 6.2 错误用例

通过修改测试用例1使其不符合赋值语句语法：

###### 6.2.1缺少操作符

###### 6.2.2缺少 ‘）’符号

