

**编译原理实验报告**

|  |  |
| --- | --- |
| 专业班级： | 计算机科学与技术 |
| 姓名学号： |  |
| 任课教师： | 李宏芒老师 |
| 指导教师： | 李宏芒老师 |
| 实验地点： | 综合实验楼 |
| 2020 - 2021 学年第二学期 | |

**目录**

[实验一 词法分析设计 3](#_Toc1729)

[1.1 实验目的 3](#_Toc8524)

[1.2 实验内容 3](#_Toc15028)

[1.3词法分析实验设计思想及算法 3](#_Toc7313)

[1.4 实验要求 6](#_Toc28967)

[1.5实验环境 7](#_Toc14222)

[1.6 实验结果与反思 7](#_Toc16334)

[1.7 核心代码 8](#_Toc23778)

[实验二 LL(1)分析法 12](#_Toc19357)

[2.1 实验目的 12](#_Toc14136)

[2.2 实验内容 12](#_Toc28660)

[2.3 LL（1）分析法实验设计思想及算法 12](#_Toc10826)

[2.4 实验要求 13](#_Toc2068)

[2.5 实验环境 13](#_Toc22784)

[2.6 实验结果与反思 14](#_Toc13644)

[2.7 核心代码 15](#_Toc18794)

[实验三 LR(1)分析法 19](#_Toc3967)

[3.1 实验目的 19](#_Toc32164)

[3.2 实验内容 19](#_Toc23285)

[3.3 LR（1）分析法实验设计思想及算法 20](#_Toc20724)

[3.4 实验要求 20](#_Toc30098)

[3.5 实验环境 21](#_Toc8852)

[3.6 实验结果与反思 21](#_Toc32490)

[3.7 核心代码 22](#_Toc25605)

# 实验一 词法分析设计

## 1.1 实验目的

通过本实验的编程实践，使学生了解词法分析的任务，掌握词法分析程序设

计的原理和构造方法，使学生对编译的基本概念、原理和方法有完整的和清楚的

理解，并能正确地、熟练地运用。

## 1.2 实验内容

用JAVA 语言实现对 C 语言子集的源程序进行词法分析。通过输入源程序从左到右对字符串进行扫描和分解，依次输出各个单词的内部编码及单词符号自身值；若遇到错误则显示“Error”，然后跳过错误部分继续显示；同时进行标识符登记符号表的管理。

### 1.2.1实现词法分析设计的主要工作

（1）从源程序文件中读入字符。

（2）统计行数和列数用于错误单词的定位。

（3）删除空格类字符，包括回车、制表符空格。

（4）按拼写单词，并用（内码，属性）二元式表示。

（5）如果发现错误则报告出错。

（6）根据需要是否填写标识符表供以后各阶段使用**。**

### 1.2.2单词的基本分类

* 关键字：由程序语言定义的具有固定意义的标识符。也称为保留字例如 if、 for、while、printf ； 单词种别码为 1。
* 标识符：用以表示各种名字，如变量名、数组名、函数名；
* 常数：任何数值常数。如 125, 1,0.5,3.1416；

* 运算符：+、-、\*、/；
* 关系运算符： <、<=、= 、>、>=、<>；

* 分界符： ；、，、（、）、[、]；

## 1.3词法分析实验设计思想及算法

1.3.1主程序设计考虑：

* 程序的说明部分为各种表格和变量安排空间。
* 在具体实现时，将各类单词设计成结构和长度均相同的形式，较短的关键字后面补空。
* k 数组------关键字表，每个数组元素存放一个关键字（事先构造好关键字表）。
* s 数组------存放分界符表（可事先构造好分界符表）。为了简单起见，分界符、
* 算术运算符和关系运算符都放在 s 表中（编程时，应建立算术运算符表和关系运算符表，并且各有类号），合并成一类。
* id 和 ci 数组分别存放标识符和常数。
* instring 数组为输入源程序的单词缓存。
* outtoken 记录为输出内部表示缓存。
* 还有一些为造表填表设置的变量。

1. 主程序开始后，先以人工方式输入关键字，造 k 表；再输入分界符等造 p

表。

1. 主程序的工作部分设计成便于调试的循环结构。每个循环处理一个单词；接收键盘上送来的一个单词；调用词法分析过程；输出每个单词的内部码。例如，把每一单词设计成如下形式: (type,pointer)其中 type 指明单词的种类，例如Pointer 指向本单词存放处的开始位置。还有一些为造表填表设置的变量。

1. 主程序开始后，先以人工方式输入关键字，造 k 表再输入分界符等造p表。
2. 主程序的工作部分设计成便于调试的循环结构。每个循环处理一个单词；接收键盘上送来的一个单词；调用词法分析过程；输出每个单词的内部码。例如，把每一单词设计成如下形式: (type,pointer)其中 type 指明单词的种类，例如Pointer 指向本单词存放处的开始位置。

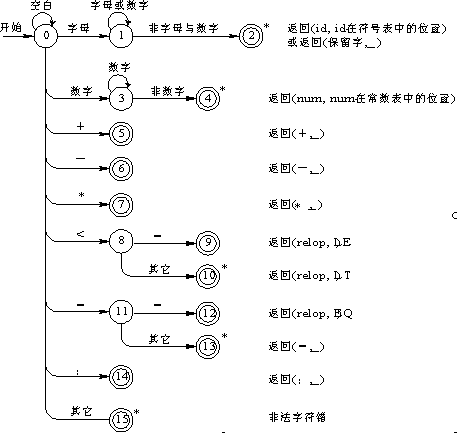


图2-1 词法分析

1.3.2词法分析过程考虑

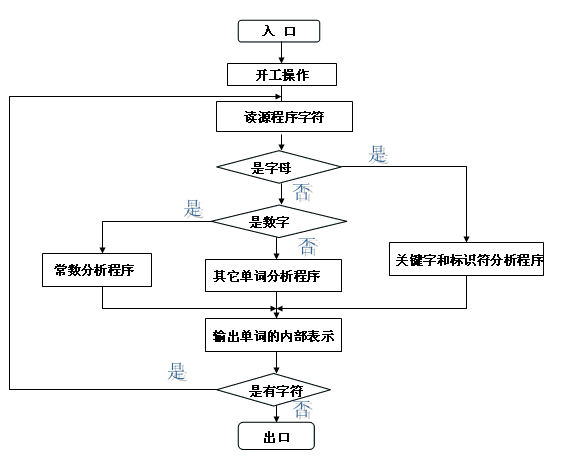


图1-2 程序框图

* 根据输入单词的第一个字符（有时还需读第二个字符），判断单词类，产生类号：以字符 k 表示关键字；id 表示标识符；ci 表示常数；s 表示分界符。
* 对于标识符和常数，需分别与标识符表和常数表中已登记的元素相比较， 如表中已有该元素，则记录其在表中的位置，如未出现过，将标识符按顺序填入 数组 id 中，将常数变为二进制形式存入数组中 ci 中，并记录其在表中的位置。

* **lexical** 过程中嵌有两个小过程：一个名为 **getchar，**其功能为从 **instring** 中按顺序取出一个字符，并将其指针 **pint** 加 1 ；另一个名为 **error，**当出现错误时，调用这个过程，输出错误编号。
* 要求：所有识别出的单词都用两个字节的等长表示，称为内部码。第一个字节为 t ，第二个字节为 i 。 t 为单词的种类。关键字的 t=１；分界符的 t=２；算术运算符的 t=３；关系运算符的 t=４；无符号数的 t=５；标识符的 t=６。i 为该单词在各自表中的指针或内部码值。表 1 为关键字表；表 2 为分界符表；表 3 为算术运算符的 i 值；表 4 为关系运算符的 i 值。取字符和统计字符行列位置子程序。

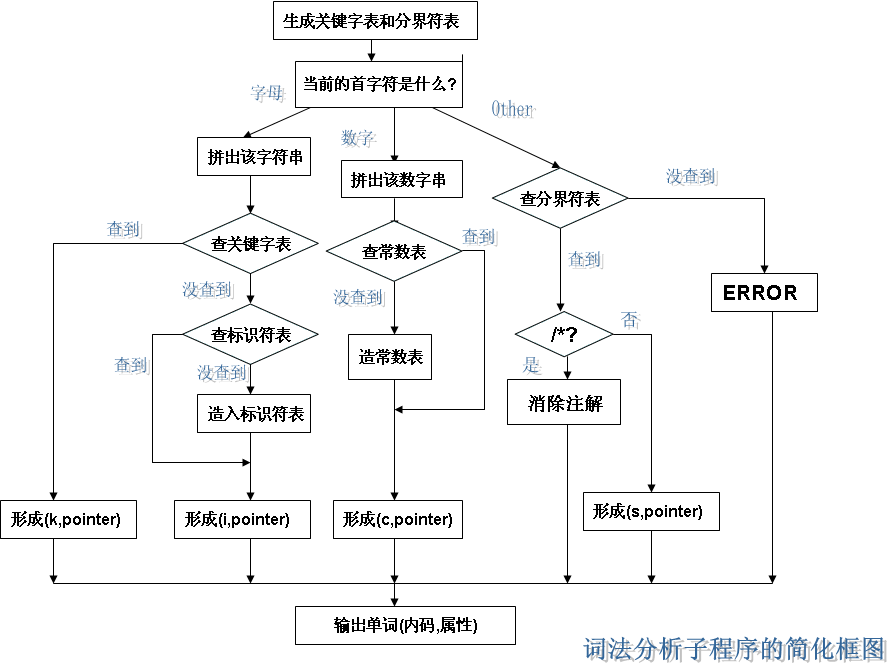


图1-2 简化框图

## 1.4 实验要求

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、将标识符填写的相应符号表须提供给编译程序的以后各阶段使用。

3、根据测试数据进行测试。测试实例应包括以下三个部分:

* 全部合法的输入。
* 各种组合的非法输入。
* 由记号组成的句子。

4、词法分析程序设计要求输出形式:

## 1.5实验环境



图1-3 实验环境

## 1.6 实验结果与反思

### 1.6.1 实验结果

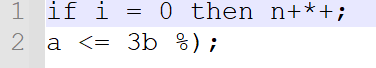


图1-4 实验输入

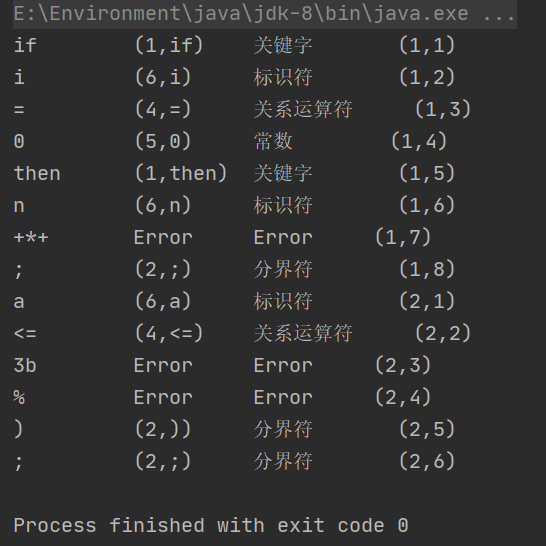


图1-5 实验结果

### 1.6.2 实验感想

1. 此次实验中我体验到了如何从编译器角度对高级语言的类似词法进行分析，并在一定程度上理解到编译过程的繁琐过程以及诸多考虑事项。这让我了解到了一些最基本的编译器原理。
2. 整个程序对于行列的分析以及其判别是浑然一体的，同时进行，并用二位数组进行数据的存储，提高工作效率。通过简单的超前搜索，就完成了本次工作。

## 1.7 核心代码

//查找识别方法

static void searchDis**(**String str**)** **{**

int i **=** 0**;**

//区分String的首符，1为字母，2为数字，3为其他

char ch **=** str**.**charAt**(**0**);**

**if** **((**ch**>=**65 **&&** ch**<=**90**)** **||** **(**ch**>=**97 **&&** ch**<=**122**))** **{**

i **=** 1**;**

**}**

**else** **if(**ch**>=**48 **&&** ch**<=**57**)** **{**

i **=** 2**;**

**}**

**else** **{**

i **=** 3**;**

**}**

//开始查找识别

**if** **(**i **==** 1**)** **{** //首符是字母，查关键字和标识符

//如果为k关键字

boolean find **=** k**.**contains**(**str**);**

**if(**find**==** **true)** **{**

//System.out.println(str+'\t'+'\t'+"(1,"+str+")"+'\t'+'\t'+"关键字"+'\t'+'\t'+"("+hangX+","+lieY+")");

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"(1,"**+**str**+**")"**,**"关键字"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

**}**

**else** **{** //不是关键字，查询标识符

find**=**id**.**contains**(**str**);**

**if(**find **==** **true)** **{**

//System.out.println(str+" (6,"+str+") "+"标识符"+" ("+hangX+","+lieY+")");

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"(6,"**+**str**+**")"**,**"标识符"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

**}**

**else** **{** //标识符表没有时，插入

//System.out.println(str+" (6,"+str+") "+"标识符"+" ("+hangX+","+lieY+")");

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"(6,"**+**str**+**")"**,**"标识符"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

id**.**add**(**str**);**

**}**

**}**

**}**

**else** **if(**i**==**2**)** **{** //首符是数字,查常数

boolean find**=**ci**.**contains**(**str**);**

**if(**find**==true)** **{**

//System.out.println(str+" (5,"+str+") "+"常数"+" ("+hangX+","+lieY+")");

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"(5,"**+**str**+**")"**,**"常数"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

**}**

**else** **{** //不在常数表中，插入

//System.out.println(str+" (5,"+str+") "+"常数"+" ("+hangX+","+lieY+")");

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"(5,"**+**str**+**")"**,**"常数"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

ci**.**add**(**str**);**

**}**

**}**

**else** **{** //查分界符，算术、逻辑运算符

**if(**s**.**contains**(**str**))** **{**

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"(2,"**+**str**+**")"**,**"分界符"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

**}**

**else** **if(**a**.**contains**(**str**))** **{**

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"(3,"**+**str**+**")"**,**"算术运算符"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

**}**

**else** **if(**r**.**contains**(**str**))** **{**

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"(4,"**+**str**+**")"**,**"关系运算符"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

**}**

**else** **{**

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**str**,**"Error"**,**"Error"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

**}**

**}**

**}**

//分析串函数

static void analysisString**(**String readStr**)** **{**

String strNow **=** ""**;** //当前识别的串

char ch1**;**

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

ch1 **=** readStr**.**charAt**(**i**);**

// 跳过空格

**if(**ch1 **==** ' '**)**

**continue;**

**if((**ch1**>=**65 **&&** ch1**<=**90**)** **||** **(**ch1**>=**97 **&&** ch1**<=**122**))** **{** //识别字母

strNow **+=** ch1**;**

i**++;**

//超前搜索

**do** **{**

ch1 **=** readStr**.**charAt**(**i**);**

//属于可识别字符

**if((**ch1**>=**65 **&&** ch1**<=**90**)** **||** **(**ch1**>=**97 **&&** ch1**<=**122**)** **||** **(**ch1**>=**48 **&&** ch1**<=**57**))** **{**

strNow **+=** ch1**;**

i**++;**

**}**

**else**

**break;**

**}while(**i **<** readStr**.**length**());**

i**--;** //指针复位

searchDis**(**strNow**);**

strNow**=**""**;**

colY**++;**

**continue;**

**}**

**else** **if(**ch1**>=**48**&&**ch1**<=**57**)** **{** //识别数字

strNow **+=** ch1**;**

i**++;**

//超前搜索

//将后续如果是字母或者是数字全部加进去

**while(**i**<**readStr**.**length**()){**

ch1**=**readStr**.**charAt**(**i**);**

**if((**ch1**>=**65**&&**ch1**<=**90**)||(**ch1**>=**97**&&**ch1**<=**122**)||(**ch1**>=**48**&&**ch1**<=**57**)||**ch1**==**'.'**)** **{**

strNow**+=**ch1**;**

i**++;**

**}**

**else**

**break;**

**}**

i**--;** //指针复位

//判断数字中是否有字母，有输出为error

int j**;**

boolean errorDone **=** **false;** //标志是否做出了出错判定

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

char ch2**=**strNow**.**charAt**(**j**);**

**if((**ch2**>=**65**&&**ch2**<=**90**)||(**ch2**>=**97**&&**ch2**<=**122**))** **{**

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**strNow**,**"Error"**,**"Error"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

errorDone**=true;**

**break;**

**}**

**}**

//判断数字中有几个小数点，有两个及以上出error

int m**;**

boolean tag **=** **false;** //tag为true时，表示已经识别到一个小数点

**for(**m **=** 0**;**m **<** strNow**.**length**();** m**++)** **{**

char ch2 **=** strNow**.**charAt**(**m**);**

**if(**ch2**==**'.' **&&** tag**)** **{**

**if(!**errorDone**)** **{** //由于上面已经输出error，下面就不用再输出

System**.**out**.**printf**(**"%-10s%-10s%-10s%-10s"**,**strNow**,**"Error"**,**"Error"**,**"("**+** rowX **+**","**+** colY **+**")"**);**

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

**}**

errorDone **=** **true;**

**break;**

**}**

**else** **if(**ch2**==**'.' **&&** **!**tag**)** **{** //第一个小数点识别

tag **=** **true;**

**}**

**}**

**if(!**errorDone**)**

searchDis**(**strNow**);**

strNow**=**""**;**

colY**++;**

**continue;**

**}**

**else** **{** //识别其他

strNow **+=** ch1**;**

**if(**s**.**contains**(**strNow**))** **{** //识别为分界符

searchDis**(**strNow**);**

strNow **=** ""**;**

colY**++;**

**continue;**

**}**

**else** **if(**a**.**contains**(**strNow**)||**r**.**contains**(**strNow**))** **{** //识别为算术运算符或关系运算符

i**++;**

//判断运算符后面是否是关系或算术运算符，是则继续添加到strNow中

**while(**i**<**readStr**.**length**())** **{**

ch1**=**readStr**.**charAt**(**i**);**

**if(**a**.**contains**(**ch1**+**""**)||**r**.**contains**(**ch1**+**""**))** **{**

strNow**+=**ch1**;**

i**++;**

**}**

**else**

**break;**

**}**

searchDis**(**strNow**);**

i**--;** //复位

strNow**=**""**;**

colY**++;**

**continue;**

**}**

**else** **{** //识别非法字符

searchDis**(**strNow**);**

strNow**=**""**;**

colY**++;**

**continue;**

**}**

**}**

**}**

# 实验二 LL(1)分析法

## 2.1 实验目的

通过完成预测分析法的语法分析程序，了解预测分析法和递归子程序法的区

别和联系。使学生了解语法分析的功能，掌握语法分析程序设计的原理和构造方

法，训练学生掌握开发应用程序的基本方法。有利于提高学生的专业素质，为培

养适应社会多方面需要的能力。

## 2.2 实验内容

 根据某一文法编制调试 LL （ 1 ）分析程序，以便对任意输入的符号串

进行分析。

 构造预测分析表，并利用分析表和一个栈来实现对上述程序设计语言的分

析程序。

 分析法的功能是利用 LL（1）控制程序根据显示栈栈顶内容、向前看符号

以及 LL（1）分析表，对输入符号串自上而下的分析过程。

## 2.3 LL（1）分析法实验设计思想及算法

### 2.3.1 模块结构

（1）定义部分：定义常量、变量、数据结构。

（2）初始化：设立 LL(1)分析表、初始化变量空间（包括堆栈、结构体、数组、

临时变量等）；

（3）控制部分：从键盘输入一个表达式符号串；

（4）利用 LL(1)分析算法进行表达式处理：根据 LL(1)分析表对表达式符号串进

行堆栈（或其他）操作，输出分析结果，如果遇到错误则显示错误信息。14

## 2.4 实验要求

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、如果遇到错误的表达式，应输出错误提示信息。

3、对下列文法，用 LL（1）分析法对任意输入的符号串进行分析：

**（1）E->TG**

**（2）G->+TG|—TG**

**（3）G->ε**

**（4）T->FS**

**（5）S->\*FS|/FS**

**（6）S->ε**

**（7）F->(E)**

**（8）F->i**

## 2.5 实验环境



图2-1 实验环境

## 2.6 实验结果与反思

### 2.6.1 实验结果

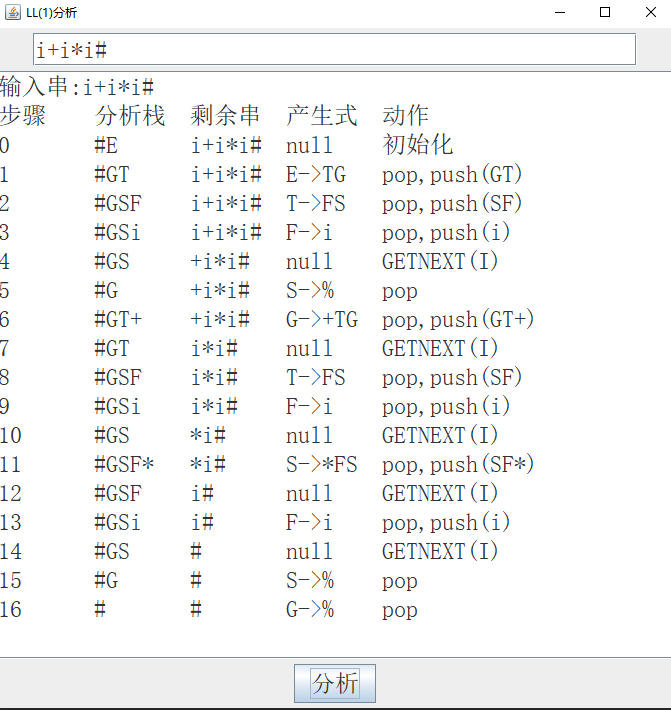


图2-2 实验结果

### 2.6.2 实验感想

1. 从文法分析到分析表的构造，中间涉及非终结符终结符的提取，first集合以及follow集合的构造，完全按照定义一步步进行，由于工作量较大，使用了部分库函数进行辅助。
2. 该程序实现时由理论到实践的步骤较多较为复杂，进行了较多次迭代才完成所有功能，下次进行较为复杂的逻辑设计时，应该先用纸笔进行逻辑的完善再进行代码编写会提高较大速率。
3. 该程序使用了MFC框架进行对话框形式的设计，由于较长时间没有上手对此生疏，在变量的外部连接以及控件的声明初始化调用方面花费较多时间，对此未来学习更新的框架以及效率更高的方式。

## 2.7 核心代码

//构造分析表

static void analysisTableCreate**()** **{**

analysisTable**=new** String**[**vn**.**size**()][**vt**.**size**()+**1**];**

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

ArrayList**<**String**>** arrayList1**=**produceFormula\_1**.**get**(**vn**.**get**(**i**));** //拿出第一个非终结符的产生式右部

**for(**int j**=**0**;**j**<**arrayList1**.**size**();**j**++)** **{**

String str1**=**arrayList1**.**get**(**j**);** //str1存储产生式

**if(**str1**.**equals**(**"%"**))** **{** //若是A->%的形式

ArrayList**<**String**>** arrayList2**=**follow**.**get**(**vn**.**get**(**i**));** //取出A的follow集

**for(**int m**=**0**;**m**<**vt**.**size**();**m**++)** **{** //遍历vt

**if(**arrayList2**.**contains**(**vt**.**get**(**m**)))** //非终结符b属于follow(A)

analysisTable**[**i**][**m**]=**vn**.**get**(**i**)+**"->"**+**str1**;** //A->%加至M[A,b]

**}**

**if(**arrayList2**.**contains**(**"#"**))**

analysisTable**[**i**][**vt**.**size**()]=**vn**.**get**(**i**)+**"->"**+**str1**;**

**}**

**else** **{** //若是A->a(的形式）

ArrayList**<**String**>** arrayList3**=**first**.**get**(**str1**.**charAt**(**0**)+**""**);** //取出产生式首符的first集

**for(**int m**=**0**;**m**<**vt**.**size**();**m**++)** **{** //遍历vt

**if(**arrayList3**.**contains**(**vt**.**get**(**m**)))** //非终结符b属于first(a)

analysisTable**[**i**][**m**]=**vn**.**get**(**i**)+**"->"**+**str1**;** //A->a加至M[A,b]

**}**

**if(**arrayList3**.**contains**(**"%"**))** **{** //如果first(a)里面有空串

ArrayList**<**String**>** arrayList2**=**follow**.**get**(**vn**.**get**(**i**));** //取出A的follow集

**for(**int m**=**0**;**m**<**vt**.**size**();**m**++)** **{** //遍历vt

**if(**arrayList2**.**contains**(**vt**.**get**(**m**)))** //非终结符b属于follow(A)

analysisTable**[**i**][**m**]=**vn**.**get**(**i**)+**"->"**+**str1**;** //A->a加至M[A,b]

**}**

**if(**arrayList2**.**contains**(**"#"**))**

analysisTable**[**i**][**vt**.**size**()]=**vn**.**get**(**i**)+**"->"**+**str1**;**

**}**

**}**

**}**

**}**

**}**

//分析程序

static void distinguish**()** **{**

vt**.**add**(**"#"**);** //暂时在终结符集中加入#

Stack**<**String**>** symbolStack**=new** Stack**<**String**>();** //符号栈

//读入字符串

String inStr**=** exp2Window**.**inputStr**;**

String a**;** //存放输入符号a

String x**;** //存放弹出的栈顶符号

int step **=** **-**1**;** //步骤数

String proStr **=** "null"**;** //存放所用产生式

String pushStr **=** ""**;** //存放push动作的对象

String actionStr **=** ""**;** //存放动作

//判断输入串是否正规

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

**if(!**vt**.**contains**(**inStr**.**charAt**(**i**)+**""**))** **{**//输入串中有不在vt中的符号

stepStr**+=**"!-----输入串错误,不包含此符号-----!"**;**

**return** **;**

**}**

**}**

symbolStack**.**push**(**"#"**);**//#入栈

symbolStack**.**push**(**vn**.**get**(**0**));** //开始符号入栈

actionStr **=** "初始化"**;**

stepStr **+=** **(**"输入串:"**+**inStr**+**'\n'**);**

stepStr **+=** **(**"步骤"**+**'\t'**+**"分析栈"**+**'\t'**+**"剩余串"**+**'\t'**+**"产生式"**+**'\t'**+**"动作"**+**'\t'**+**'\n'**);**

a **=** inStr**.**charAt**(**0**)+**""**;** //取一输入符号a

**while(true)** **{**

step**++;** //步骤+1

//获得分析栈所要打印的字符串

String stackStr **=** ""**;** //存放分析栈里的元素

**while(!**symbolStack**.**empty**())** **{**

stackStr **+=** symbolStack**.**pop**();**

**}**

String stackStr1 **=** **new** StringBuffer**(**stackStr**).**reverse**().**toString**();** //反转字符串

**for(**int i **=** 0**;**i **<** stackStr1**.**length**();** i**++){**

symbolStack**.**push**(**stackStr1**.**charAt**(**i**)+**""**);** //重新恢复栈

**}**

//打印步骤

stepStr **+=** **((**step**+**""**)+**'\t'**+**stackStr1**+**'\t'**+**inStr**+**'\t'**+**proStr**+**'\t'**+**actionStr**+**'\t'**+**'\n'**);**

proStr **=** "null"**;**

actionStr **=** ""**;**

x **=** symbolStack**.**pop**();** //弹出栈顶符号放入x

**if(!**vn**.**contains**(**x**))** **{** //x是不是非终结符

**if(**vt**.**contains**(**x**))** **{** //x是终结符

**if(**x**.**equals**(**a**))** **{** //x=a

**if(**x**.**equals**(**"#"**))** //x=# 代表结束，成功

**break;**

**else** **{** //x!=#开始下一步骤

inStr**=**inStr**.**substring**(**1**);**

a**=**inStr**.**charAt**(**0**)+**""**;** //取一输入输入符号a

actionStr**=**"GETNEXT(I)"**;**

**continue;**

**}**

**}**

**else** **{**

//System.out.println("x!=a"); //error

stepStr**+=(**"Error--------------------x!=a"**+**'\n'**);**

**break;**

**}**

**}**

**else** **{**

//System.out.println("not vt and not vn"); //error

stepStr**+=(**"Error--------------------not vt and not vn"**+**'\n'**);**

**break;**

**}**

**}**

**else** **{**

//取得x,a的位置

int xIndedx**=**vn**.**indexOf**(**x**);**

int aIndedx**=**vt**.**indexOf**(**a**);**

**if(**analysisTable**[**xIndedx**][**aIndedx**]!=null)** **{**

//symbolStack.pop(); //栈顶出栈

actionStr**+=**"pop"**;**

String str1**=**analysisTable**[**xIndedx**][**aIndedx**];** //str1=M[x,a]

proStr**=**str1**;**

String str2**=new** StringBuffer**(**str1**).**reverse**().**toString**();** //反转字符串

**if(**str2**.**charAt**(**0**)==**'%'**)** //若产生式是推出空串的情况

**continue;**

//倒序入栈

**for(**int j**=**0**;**j**<**str1**.**length**();**j**++)** **{**

**if(**str2**.**charAt**(**j**)==**'>'**)**

**break;**

symbolStack**.**push**(**str2**.**charAt**(**j**)+**""**);**

pushStr**+=**str2**.**charAt**(**j**)+**""**;**

**}**

actionStr**+=**",push("**+**pushStr**+**")"**;**

pushStr**=**""**;**

**continue;**

**}**

**else** **{**

//System.out.println("no M[A,a]"); //error

stepStr**+=(**"Error--------------------no M[A,a]"**+**'\n'**);**

**break;**

**}**

**}**

**}**

vt**.**remove**(**"#"**);** //移去#，恢复vt

**}**

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

produceVt\_Vn**();**//构造vn和vt

printVnAndVt**();**

produceFormulaHandle**();**

printProduction**();**

//构造first集

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

firstCreate**(**vn**.**get**(**i**));**

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

firstCreate**(**vt**.**get**(**i**));**

printFirst**();**

//构造follow集

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

followCreate**(**vn**.**get**(**i**));**

printFollow**();**

//构造分析表

analysisTableCreate**();**

printAnalysic**();**

//调用窗口

**new** exp2Window**();**

**}**

**}**

# 实验三 LR(1)分析

## 3.1 实验目的

构造 LR(1)分析程序，利用它进行语法分析，判断给出的符号串是否为该文

法识别的句子，了解 LR（K）分析方法是严格的从左向右扫描，和自底向上的

语法分析方法。

## 3.2 实验内容

对下列文法，用 LR（1）分析法对任意输入的符号串进行分析：

**（1）E-> E+T**

**（2）E->T**

**（3）T-> T\*F**

**（4）T->F**

**（5）F-> (E)**

**（6）F-> i**

## 3.3 LR（1）分析法实验设计思想及算法

(1)总控程序，也可以称为驱动程序。对所有的 LR 分析器总控程序都是相同的。

(2)分析表或分析函数，不同的文法分析表将不同，同一个文法采用的LR分析器不同时，分析表将不同，分析表又可以分为动作表（ACTION）和状态转换（GOTO）表两个部分，它们都可用二维数组表示。

(3)分析栈，包括文法符号栈和相应的状态栈，它们均是先进后出栈。分析器的动作就是由栈顶状态和当前输入符号所决定。

 LR 分析器由三个部分组成：

 其中:SP 为栈指针，S[i]为状态栈，X[i]为文法符号栈。状态转换表用

GOTO[i，X]=j 表示，规定当栈顶状态为 i，遇到当前文法符号为 X 时应转向状态 j，X 为终结符或非终结符。

 ACTION[i，a]规定了栈顶状态为 i 时遇到输入符号 a 应执行。动作有四种可能：

**(1)移进：**

action[i，a]= Sj：状态 j 移入到状态栈，把 a 移入到文法符号栈，其中 i,j 表

示状态号。

**(2)归约：**

action[i，a]=rk：当在栈顶形成句柄时，则归约为相应的非终结符 A，即文法中有 A- B 的产生式，若 B 的长度为 R(即|B|=R)，则从状态栈和文法符号栈中自顶向下去掉 R 个符号，即栈指针 SP 减去 R，并把 A 移入文法符号栈内，

j=GOTO[i,A]移进状态栈，其中 i 为修改指针后的栈顶状态。

**(3)接受 acc:**

当归约到文法符号栈中只剩文法的开始符号 S 时，并且输入符号串已结束即当前输入符是'#'，则为分析成功。

**(4)报错:**

当遇到状态栈顶为某一状态下出现不该遇到的文法符号时，则报错，说明输

入端不是该文法能接受的符号串。

## 3.4 实验要求

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、如果遇到错误的表达式，应输出错误提示信息。

3、程序输入/输出实例：

输入一以#结束的符号串(包括+\*（）i#)：在此位置输入符号串

## 3.5 实验环境



图3-1 实验环境

## 3.6 实验结果与反思

### 3.6.1 实验结果

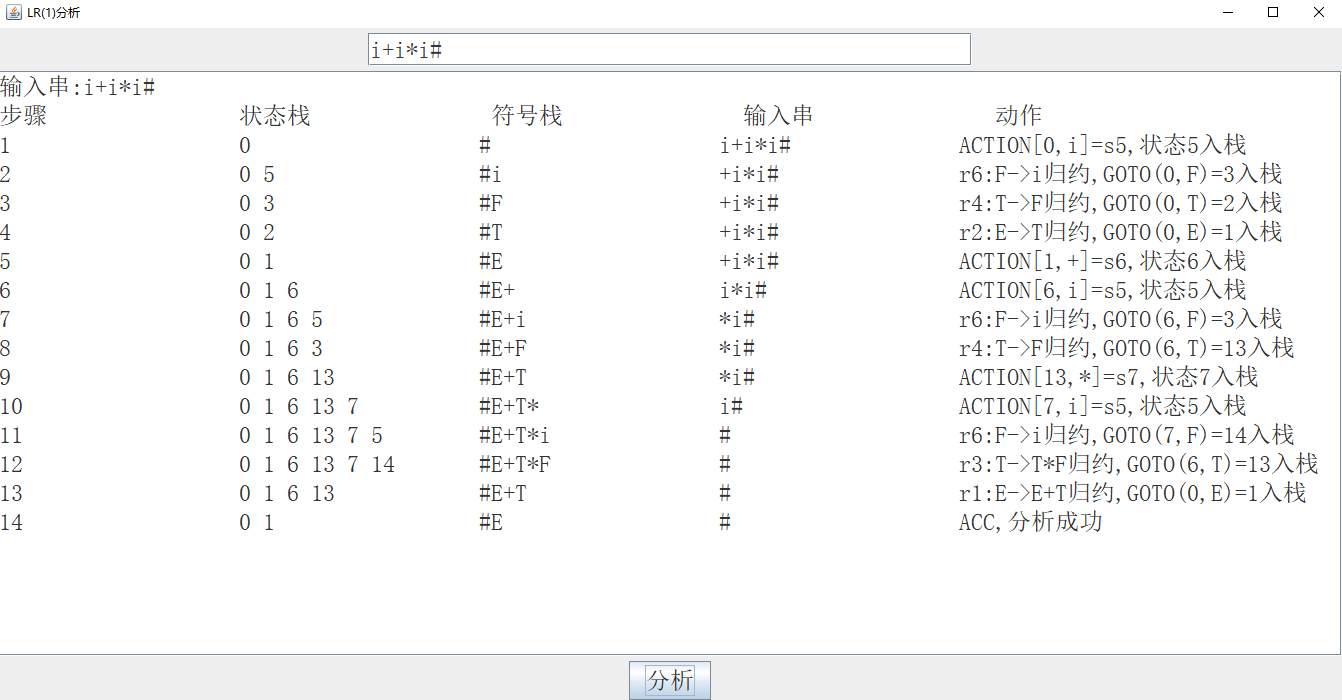


图3-2 实验结果

### 3.6.2 实验感想

1. 在完成本次实验后，对于first和follow集合求取了解更加深刻，关于first follow集合求取，文法递归的改变需要从新设计，工作量较大。
2. 本次试验所有求取严格按照课本定义进行，构造较为复杂，使用了还是比较大，若增加迭代次数可减少行数，故不累述。
3. 此次试验由于涉及步骤较多，所以变量较多，各种数据类型的转化较多，尤其是涉及到MFC特有的变量类型。

## 3.7 核心代码

//构造项目集

static void projectSetCreate**()**

**{**

first**.**put**(**"#"**,** **new** ArrayList**<**String**>(**Arrays**.**asList**(**"#"**)));** //在first集中添加<#,#>

String str**=**vn**.**get**(**0**);** //取得开始符(文法需要拓广,用A代替开始符',)

String str1**=**"A->."**+**str**+**",#"**;**

projectSet**[**0**][**1**]=**str1**;**

projectSet**[**0**][**0**]=**"1"**;**

int i**=**0**;** //正在处理的项目集的索引

closure**(**projectSet**[**0**]);** //构造I0闭包

**while(true)** **{**

**for(**int m**=**0**;**m**<**vn**.**size**();**m**++)** **{**

String x**=**vn**.**get**(**m**);**

String projectset**[]=**Goto**(**projectSet**[**i**],**x**);**

int num**=**Integer**.**parseInt**(**projectset**[**0**]);**

**if(**num**==**0**)** **{**

**continue;**

**}**

**else** **{**

int ip**=**isExistence**(**projectset**);**

**if(**ip**==-**1**)** **{** //不存在重复

ii**++;**

projectSet**[**ii**]=**projectset**;**

projectSetLink**[**i**][**ii**]=**x**;**

**}**

**else** **{**

projectSetLink**[**i**][**ip**]=**x**;**

**}**

**}**

**}**

**for(**int m**=**0**;**m**<**vt**.**size**();**m**++)** **{**

String x**=**vt**.**get**(**m**);**

String projectset**[]=**Goto**(**projectSet**[**i**],**x**);**

int num**=**Integer**.**parseInt**(**projectset**[**0**]);**

**if(**num**==**0**)** **{**

**continue;**

**}**

**else** **{**

int ip**=**isExistence**(**projectset**);**

**if(**ip**==-**1**)** **{** //不存在重复

ii**++;**

projectSet**[**ii**]=**projectset**;**

projectSetLink**[**i**][**ii**]=**x**;**

**}**

**else** **{**

projectSetLink**[**i**][**ip**]=**x**;**

**}**

**}**

**}**

**if(**i**==**ii**)** **{**

**break;**

**}**

i**++;**

**}**

**}**

//构造分析表

static void createAnalysisTable**()** **{**

vt**.**add**(**"#"**);** //终结符中加入#

action**=new** String**[**ii**+**1**][**vt**.**size**()];** //初始化action

GOTO**=new** String**[**ii**+**1**][**vn**.**size**()];** //初始化goto

int tag**[]=new** int**[**ii**+**1**];** //记录这个状态是不是末尾状态，1不是，0是

//部分

**for(**int i**=**0**;**i**<=**ii**;**i**++)** **{**

**for(**int j**=**0**;**j**<=**ii**;**j**++)** **{**

**if(**projectSetLink**[**i**][**j**]!=null)** **{**

tag**[**i**]=**1**;**

String str**=**projectSetLink**[**i**][**j**];**

**if(**vn**.**contains**(**str**))** **{** //非终结符

GOTO**[**i**][**vn**.**indexOf**(**str**)]=**j**+**""**;**

**}**

**else** **{** //终结符

action**[**i**][**vt**.**indexOf**(**str**)]=**"s"**+**j**;**

**}**

**}**

**}**

**}**

//归约部分

**for(**int i**=**0**;**i**<=**ii**;**i**++)** **{**

**if(**tag**[**i**]==**0**)** **{**

String projectset**[]=**projectSet**[**i**];** //取出末尾项目集

int num**=**Integer**.**parseInt**(**projectset**[**0**]);**

**for(**int j**=**1**;**j**<=**num**;**j**++)** **{**

String str**=**projectset**[**j**];**

int pointIndex**=**str**.**indexOf**(**'.'**);**

String strTail**=**str**.**substring**(**str**.**length**()-**1**);**

str**=**str**.**substring**(**0**,**pointIndex**);**

int produceIndex**=**produceFormula**.**indexOf**(**str**)+**1**;**

action**[**i**][**vt**.**indexOf**(**strTail**)]=**"r"**+**produceIndex**;**

**}**

**}**

**else** **{**

String projectset**[]=**projectSet**[**i**];** //取出含有可归约的项目的项目集（并非全都是）

int num**=**Integer**.**parseInt**(**projectset**[**0**]);**

**for(**int j**=**1**;**j**<=**num**;**j**++)** **{**

String str**=**projectset**[**j**];**

int pointIndex**=**str**.**indexOf**(**'.'**);**

int douIndex**=**str**.**indexOf**(**','**);**

**if(**douIndex**==**pointIndex**+**1**)** **{** //A->a.,?这种结构

**if(**str**.**charAt**(**0**)==**'A'**)** **{**

action**[**i**][**vt**.**indexOf**(**"#"**)]=**"acc"**;**

**}**

**else{**

String strTail**=**str**.**substring**(**str**.**length**()-**1**);**

str**=**str**.**substring**(**0**,**pointIndex**);**

int produceIndex**=**produceFormula**.**indexOf**(**str**)+**1**;**

action**[**i**][**vt**.**indexOf**(**strTail**)]=**"r"**+**produceIndex**;**

**}**

**}**

**}**

**}**

**}**

**}**