编译原理课程实验报告

实验 2：语法分析

姓名	张志路	院系	计算机学院	学号	1160300909
任课教师	辛明影		指导教师	辛明影	
实验地点	格物 208		实验时间	2019 年 4 月 21 日周日 5-6 节	
实验课表现	出勤、表现得分		实验报告得分		实验总分	
	操作结果得分					

一、需求分析

得分

要求：采用至少一种句法分析技术（SLR(1)、LR(1)）对类高级语言中的基本语句进行句法分析。阐述句法分析系统所要完成的功能。

在词法分析器的基础上设计实现类高级语言的语法分析器，基本功能如下：

1. 能识别以下几类语句：

(1) 声明语句（变量声明）

(2) 表达式及赋值语句（简单赋值）

(3) 分支语句：if_then_else

(4) 循环语句：do_while

2. 要求编写自动计算 CLOSURE(I)和 GOTO 函数的程序，并自动生成 LR 分析表。
3. 具备简单语法错误处理能力，能准确给出错误所在位置，并采用可行的错误恢复策略。输出的错误提示信息格式如下：Error at Line [行号]：[说明文字]
4. 系统的输入形式：要求可以通过文件导入文法和测试用例,测试用例要涵盖“实验内容”第 1 条中列出的各种类型的语句，并设置一些语法错误。
5. 系统的输出分为两部分：一部分是打印输出语法分析器的 LR 分析表。另一部分是打印输出语法分析结果，既输出归约时的产生式序列。

二、文法设计

得分

要求：给出如下语言成分的文法描述。

本文法在实验指导书给定的文法基础上做了相应的改进，消除了二义性。

1. 全局定义

P' → P

P → D

P → S

S → S S

2. 声明语句（变量声明）

D → D D | proc id ; D S | T id ;

T → X C | record D end

X → integer | real

C → [num] C | ε

3. 表达式及赋值语句

S → id = E ; | L = E ;

E → E + E1 | E1

E1 → E1 * E2 | E2

E2 → (E) | - E | id | num | L

L → id [E] | L [E]

4. 分支语句“if_then_else”和循环语句“do_while”

S → S1 | S2

S1 → if B then S1 else S1 | while B do S0

S2 → if B then S1 else S2 | if B then S0

S0 → begin S3 end

S1 → begin S3 end

S2 → begin S3 end

S3 → S3 ; S | S

5. 布尔表达式

B → B or B1 | B1

B1 → B1 and B2 | B2

B2 → not B | (B) | E R E | true | false

R → < | <= | == | != | > | >=

6. 过程调用

S → call id (EL)

EL → EL , E

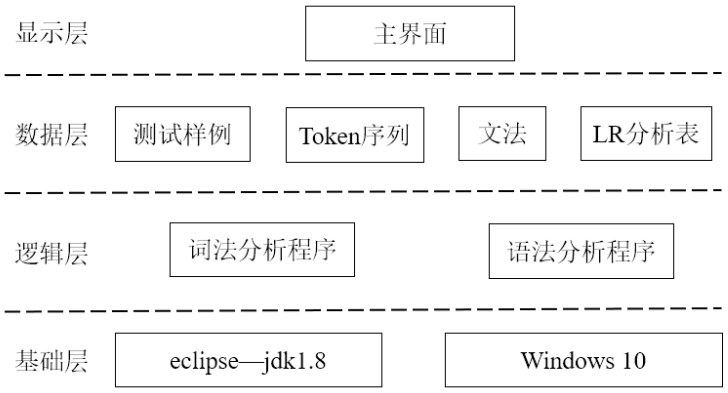
EL → E

三、系统设计	得分	

要求：分为系统概要设计和系统详细设计。

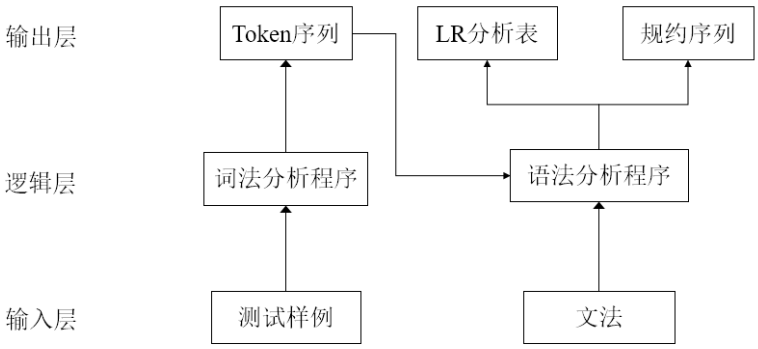
1. 系统概要设计：给出必要的系统宏观层面设计图，如系统框架图、数据流图、功能模块结构图等以及相应的文字说明。

(1) 系统框架图



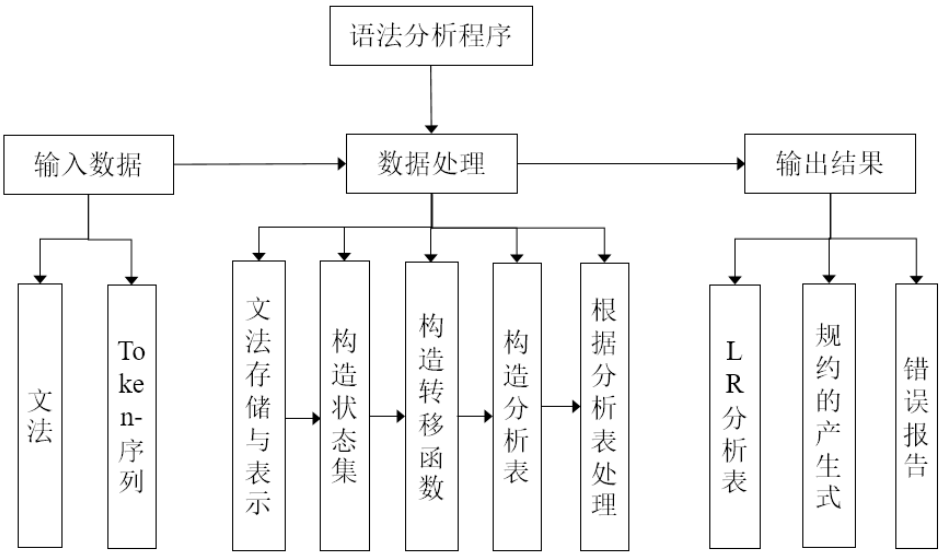
系统分为基础层、逻辑层、数据层和显示层，基础层即在 Windows10、jdk1.8 环境下利用 java 语言进行编程，逻辑层即词法分析程序和语法分析程序，数据层包含测试样例、Token 序列、文法和 LR 分析表，显示层显示一个主界面。

(2) 数据流图



语法分析程序通过读入文法，生成 LR 分析表，然后读入对测试样例的词法分析结果，根据分析表对其进行 LR(1)分析，从而生成规约的产生式序列和错误报告。

(3) 功能模块结构图



如上图所示，语法分析程序的功能主要为读入数据、分析数据和给出结果。对于数据处理模块，首先要设计合理的数据结构，对文法进行存储和表示；然后根据文法生成状态集和转移函数，进而构造分析表；最后对词法分析得到的 Token 序列利用分析表进行移进规约操作，直到分析完成。

2. 系统详细设计：对如下工作进行展开描述

(1) 核心数据结构的设计

a. Production 类

```
public String left;  // 产生式左部
public ArrayList<String> list = new ArrayList<String>();  //产生式右部

/**
 * 存储产生式，此时表示类似于"A->BCD"的形式
 * @param s 产生式字符串
 */
public Production(String s)
```

Production 类对一个普通的产生式进行表示，此时表示的是类似于“A → BCD”的形式。

b. ProductionState 类

```
public Production d;  // 产生式
public String lr;  // 后继符
public int index;  // 后继符位置

/**
 * DFA状态集中每一个状态
 * 此时表示类似于"A->BC.D, a"的形式
 * @param d  产生式
 * @param lr 后继符
 * @param index 后继符位置
 */
public ProductionState(Production d,String lr,int index)
```

ProductionState 类对一个产生式状态进行表示，此时表示类似于“A → BC.D, a”的形式。

c. DFAState 类

```
// 项目集编号,即DFA状态号
public int id;
// DFA中的状态集列表,每个元素表示一个产生式状态
public ArrayList<ProductionState> set = new ArrayList<ProductionState>();

/**
 * 一个DFA状态
 * @param id 项目集编号,即DFA状态号
 */
public DFAState(int id)
```

DFAState 类用来表示 LR(1)分析的一个 DFA 状态，即一个项目集闭包，id 为项目集编号。

d. DFAStateSet 类

```
// 所有项目集列表，每个元素为一个DFA状态
public ArrayList<DFAState> states = new ArrayList<DFAState>();
```

DFAState 类用来表示 LR(1)分析法的所有 DFA 状态，即所有项目集合。

e. GrammarProc 类

```
public static String emp = "ε";  // 空串
public static String end = "#";  // 结束符
public static TreeSet<String> VN = new TreeSet<String>();  // 非终结符集
public static TreeSet<String> VT = new TreeSet<String>();  // 终结符集
public static ArrayList<Production> F = new ArrayList<Production>();  // 产生式集
//  每个符号的first集
public static HashMap<String,TreeSet<String> > firstMap = new HashMap<String,TreeSet<String>>()
```

GrammarProc 类用来读入并存储类似于“A→BC|DE”形式的产生式，根据前述的数据结构进行存储，产生终结符、非终结符以及每个符号的 FIRST 集。

f. AnalyzeTable 类

AnalyzeTable 类用来构造一个分析表，此类构造 DFA 状态集和转移函数，进而构造 LR(1)分析表。

主要数据结构如下图所示。

```
public static String error = "--";  // 错误符号
public static String acc = "acc";  // ACC，接收成功符号
public DFAStateSet dfa;  // 所有DFA状态
public int stateNum;  // DFA状态数

public int actionLength;  // Action表列数
public int gotoLength;  // GoTo表列数
private String[] actionCol;  // Action表列名数组
private String[] gotoCol;  // GoTo表列名数组
private String[][] actionTable;  // Action表，二维数组
private int[][] gotoTable;  // GoTo表，二维数组

//  当第x号DFA状态,输入S符号时,转移到第y号DFA状态,则:
private ArrayList<Integer> gotoStart = new ArrayList<Integer>();  // 存储第x号DFA状态
private ArrayList<Integer> gotoEnd = new ArrayList<Integer>();  // 存储第y号DFA状态
private ArrayList<String> gotoPath = new ArrayList<String>();  // 存储S符号

/**
 * 构造分析表
 */
public AnalyzeTable()
```

g. SyntaxParser 类

SyntaxParser 类根据构造的语法分析表进行 LR(1)语法分析。

```
private Lexical lex;  // 词法分析器
private ArrayList<Token> tokenList = new ArrayList<Token>();  // 从词法分析器获得的所有token
private int length;  // tokenlist的长度
private int index;  // 语法分析进行到的位置

private AnalyzeTable table;  //构造的语法分析表
private Stack<Integer> stateStack;  //用于存储相应的DFA状态号
private static StringBuffer result = new StringBuffer();  // 保存规约结果
private static StringBuffer error = new StringBuffer();  // 保存错误报告结果
```

(2) 主要功能函数说明

a. 求 FIRST 集

不断应用下列规则，直到没有新的终结符或 ε 可以被加入到任何 FIRST 集合中为止。

1） 如果 X 是一个终结符，那么 FIRST(X)={X}。

2） 如果 X 是一个非终结符，且 $X \rightarrow Y_1 \ldots Y_k \in P$ ($k \geq 1$)，那么，

a) 如果对于某个 i，a 在 FIRST (Y_i)中，且ε在所有的 FIRST(Y_1)，…，FIRST(Y_{i-1})

中，就把 a 加入到 FIRST(X)中。

b) 如果对于所有的 j=1,2,. . . , k，ε在 FIRST(Y_j)中，那么将ε加入到 FIRST(X)。

3） 如果 X→ε∈P，那么将ε加入到 FIRST(X)中。

b. 求项目集闭包

根据下列式子构造项目集闭包。

CLOSURE(I) = I∪ {[B→.γ,b] | [A→α.Bβ,a]∈CLOSURE(I), B→γ∈P, b∈FIRST(βa)}

伪代码如下所示。

```
1.  CLOSURE(I)
2.  {
3.      repeat
4.          for (I 中的每个项[A→α.Bβ, a])
5.              for (G'的每个产生式 B→γ)
6.                  for (FIRST(βa)中的每个符号 b)
7.                      将[B→.γ, b]加入到集合 I 中;
8.          until 不能向 I 中加入更多的项;
9.      until I ;
10. }
```

c. 求 GOTO 函数

根据下列式子构造 GOTO 函数。

GOTO(I, X) = CLOSURE({ [A→αX.β, a] | [A→α.Xβ, a]∈I })

伪代码如下所示。

```
1.  GOTO(I, X)
2.  {
3.      将 J 初始化为空集;
4.      for(I 中的每个项[A→α·Xβ, a])
5.          将项[A→αX·β, a]加入到集合 J 中;
6.      return CLOSURE(J);
7.  }
```

d. 为文法 G’构造 LR(1)项集族

伪代码如下所示。

```
1.  items(G')
2.  {
3.      将 C 初始化为{CLOSURE({[S'→.S, #]})};
4.      repeat
5.          for(C 中的每个项集 I)
6.              for(每个文法符号 X)
```

```
                    if(GOTO(I, X)非空且不在 C 中)
                        将 GOTO(I, X)加入 C 中;
    until 不再有新的项集加入到 C 中;
}
```

e. 构造 LR(1)语法分析表

伪代码如下所示。

```
构造 G'的规范 LR(1)项集族 C={I,I,…,I}
根据 I 构造得到状态 i。状态 i 的语法分析动作按照下面的方法决定:
If [A→α·aβ,b]∈I and GOTO(I,a)=I
    ACTION[i,a]=sj
If [A→α·Bβ,b]∈I and GOTO(I,B)=I
    GOTO[i,B]=j
If [A→α·,a]∈I and A!=S'
    ACTION[i,a]=rj（j 是产生式 A→α 的编号）
If [S'→S·,#]∈I
    ACTION[i,#]=acc;
没有定义的所有条目都设置为“error”
```

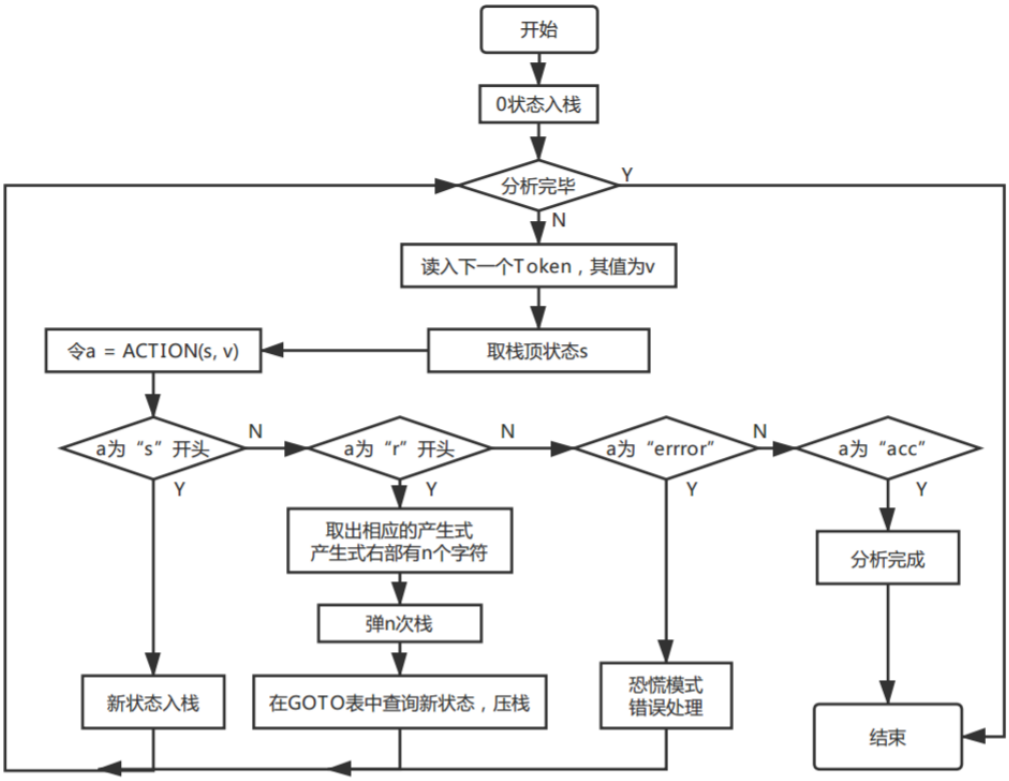
f. 错误处理

当栈顶状态为 s，当前输入符号为 a，查表得 Action[s,a]=error，这表明语法出错了。错误处理伪代码如下。

```
向栈内搜索非终结符 A，它的归约前状态为 S，将 A 前面的符号全部弹出栈。
不断地读入输入符号，直到读到 a∈Follow(A)集为止，
将 goto[s,A]压入栈
```

整个过程就相当于完成了一次归约，只不过归约的不是句柄。这样处理完后，就相当于从一个新的句子成分开始分析。

(3) 程序核心部分的程序流程图



四、系统实现及结果分析

得分

要求：对如下内容展开描述。

1. 系统实现过程中遇到的问题

(1) 文法二义性问题

实验指导书给出的文法是具有二义性的，主要体现在表达式符号的优先级和 if else 结构上，因此对这两部分进行改进。

此外，原文法不能确定 while 和 if else 语句的控制范围，因此对这两部分加上 begin、end 符号以确定控制范围。

改进后的文法已经在第二部分给出，此处不再赘述。

(2) 空符号的处理问题

在计算 FIRST 集和规约时，要对有空（ε）符号的产生式特殊处理，这部分很容易产生 bug，需要不断调试。

(3) 错误处理问题

错误恢复包括恐慌模式错误恢复和短语层次错误恢复，本程序采用的是恐慌模式的错误恢复。

2. 输出该句法分析器的分析表

语法分析表较大，这里仅仅展示小部分，全部的内容在“LR_Analysis_Table.txt”文件中给出。

	!=	(	)	*	+	,	-	;	<	<=	=	==	>	>=	[
0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
2	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
4	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
5	--	--	--	--	--	--	--	s6	--	--	--	--	--	--	--
6	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
7	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
8	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
9	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
10	--	--	--	--	--	--	--	--	--	--	s11	--	--	--	s83
11	--	s18	--	--	--	--	s67	--	--	--	--	--	--	--	--
12	--	--	--	--	s14	--	--	s13	--	--	--	--	--	--	--
13	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
14	--	s18	--	--	--	--	s67	--	--	--	--	--	--	--	--
15	--	--	--	s16	r15	--	--	r15	--	--	--	--	--	--	--
16	--	s18	--	--	--	--	s67	--	--	--	--	--	--	--	--
17	--	--	--	r17	r17	--	--	r17	--	--	--	--	--	--	--
18	--	s25	--	--	--	--	s30	--	--	--	--	--	--	--	--
19	--	--	s20	--	s21	--	--	--	--	--	--	--	--	--	--
20	--	--	--	r19	r19	--	--	r19	--	--	--	--	--	--	--
21	--	s25	--	--	--	--	s30	--	--	--	--	--	--	--	--
22	--	--	r15	s23	r15	--	--	--	--	--	--	--	--	--	--
23	--	s25	--	--	--	--	s30	--	--	--	--	--	--	--	--
24	--	--	r17	r17	r17	--	--	--	--	--	--	--	--	--	--
25	--	s25	--	--	--	--	s30	--	--	--	--	--	--	--	--
26	--	--	s27	--	s21	--	--	--	--	--	--	--	--	--	--
27	--	--	r19	r19	r19	--	--	--	--	--	--	--	--	--	--
28	--	--	r16	s23	r16	--	--	--	--	--	--	--	--	--	--

3. 针对一测试程序输出其句法分析结果

测试样例如下。

```
1.  proc maxminavg;
2.      integer[3] a;
3.      integer max;
4.      integer min;
5.      integer sum;
6.      integer i;
7.      real avg;
8.
9.      a[1]=1;
10.     a[2]=2;
11.     a[3]=3;
12.     sum=0;
13.     max=a[1];
14.     min=a[1];
15.     i=1;
16.
17.     while true do
18.     begin
19.         a=a+1;
20.     end
21.
22.     if a[i]>max then
23.     begin
24.         max=a[i];
25.     end
```

```
26.
27.     if a[i]>max then
28.     begin
29.         max=a[i];
30.     end
31.     else
32.     begin
33.         a=1;
34.     end
35.
36.     while true do
37.     begin
38.
39.         if i<=3 then
40.         begin
41.             sum=sum+a[i];
42.         end
43.
44.         if a[i]>max then
45.         begin
46.             max=a[i];
47.         end
48.
49.         if a[i]<min then
50.         begin
51.             min=a[i];
52.         end
53.         else
54.         begin
55.             call fuction(b,c,d)
56.         end
57.
58.         i=i+1;
59.         avg=sum*1;
60.     end
61.
62.     a[1]]>=1;;
63.
64.     if i<=<3 then
65.     begin
66.         a=a+1;
67.     end
68.
```

部分语法分析结果如下。全部结果在“Productions.txt”文件中给出。

产生式
X -> integer
C -> ε
C -> [num] C
T -> X C
D -> T id ;
X -> integer
C -> ε
T -> X C
D -> T id ;
X -> integer
C -> ε
T -> X C
D -> T id ;
X -> integer
C -> ε
T -> X C
D -> T id ;
X -> integer
C -> ε
T -> X C
D -> T id ;

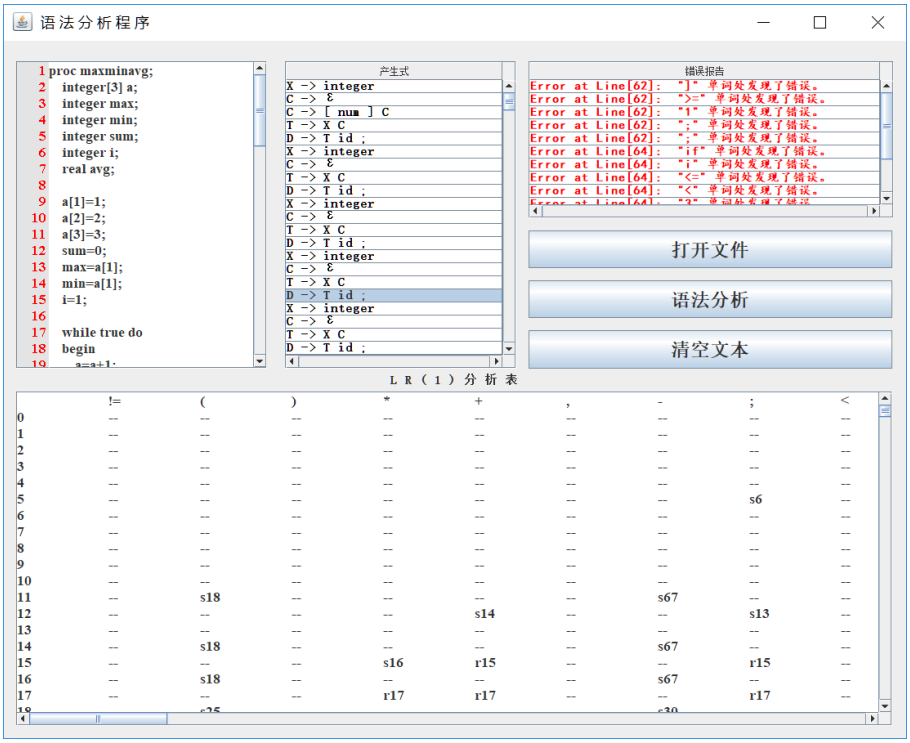
4. 输出针对此测试程序对应的语法错误报告；

   错误报告如下。

错误报告
Error at Line[62]: "]" 单词处发现了错误。
Error at Line[62]: ">=" 单词处发现了错误。
Error at Line[62]: "1" 单词处发现了错误。
Error at Line[62]: ";" 单词处发现了错误。
Error at Line[62]: ";" 单词处发现了错误。
Error at Line[64]: "if" 单词处发现了错误。
Error at Line[64]: "i" 单词处发现了错误。
Error at Line[64]: "<=" 单词处发现了错误。
Error at Line[64]: "<" 单词处发现了错误。
Error at Line[64]: "3" 单词处发现了错误。
Error at Line[64]: "then" 单词处发现了错误。
Error at Line[65]: "begin" 单词处发现了错误。
Error at Line[66]: "a" 单词处发现了错误。
Error at Line[67]: "end" 单词处发现了错误。

5. 对实验结果进行分析。

   整体效果如下所示。



经过反复地测试与实验，从最后的实验结果来看，规约的产生式以及次序均正确，例如 if else 语句、while 与 if else 的互相嵌套、call 语句、record 语句和多维数组等。 同时也对一些常见的错误进行了识别与恢复。例如最常见的“；”后又跟了一个“；”，系统自动忽略后一个分号；操作符不和规范，如出现“a>=>b”语句，系统对整个语句忽略掉。 结果表明，系统的结果与目标基本一致，系统目标达成。
指导教师评语： 日期：