编译系统课程实验报告

实验 2：语法分析

姓名	余涛	院系	计算学部	学号	1180300829
任课教师	陈鄞		指导教师	陈鄞	
实验地点	格物 213		实验时间	2021.5.8	
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（LL(1)、SLR(1)、LR(1)或 LALR(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。

实验目的：

1．巩固对语法分析的基本功能和原理的认识。

2．通过对语法分析表的自动生成加深语法分析表的认识。

3．理解并处理语法分析中的异常和错误。

实验内容：

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：

声明语句（包括变量声明、数组声明、记录声明和过程声明）

表达式及赋值语句（包括数组元素的引用和赋值）

分支语句：if_then_else

循环语句：do_while

过程调用语句

（2）能够识别出测试用例中的语法错误。在输出错误提示信息时，需要输出具体的错误类型（语法错误）、出错的位置（源程序行号）以及相关的说明文字，其格式为：

Syntax error at Line [行号]: [说明文字].

说明文字的内容没有具体要求，但是错误类型和出错的行号一定要正确，因为这是判断输出错误提示信息是否正确的唯一标准。

（3）系统的输入形式：要求通过文件导入测试用例。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。

（4）系统的输出形式：打印输出语法分析结果，格式如下：将构造好的语法分析树按照先序遍历的方式打印每一个结点的信息，这些信息包括结点的名称以及结点对应的成分在输入文件中的行号（行号被括号所包围，并且与结点名称之间有一个空格）。所谓某个成分在输入文件中的行号是指该成分产生的所有词素中的第一个在输入文件中出现的行号。对于叶结点，如果其 token 的属性值不为空，则将其属性值也打印出来。属性值与结点名称之间以一个冒号和空格隔开。每一个结点独占一行，而每个子结点的信息相对于其父结点的信息来说，在行首都要求缩进 2 个空格。

二、文法设计

得分

要求：给出如下语言成分的文法描述。

1.声明语句（包括变量声明、数组声明、记录声明和过程声明）

2.表达式及赋值语句（包括数组元素的引用和赋值）

3.分支语句：if_then_else

4.循环语句：do_while
5.过程调用语句
所设计的完整文法如下所示：

```
P' -> P
P -> D P | S P | ε
D -> D D | proc X id ( M ) { P } | record id { P } | T id A ; | struct id { D } ;
| X * id ;
A -> = F A | , id A | ε
M -> M , X id | X id
T -> X C
X -> int | float | bool | char
C -> [ num ] C | ε
S -> MachedS | OpenS
MachedS -> if ( B ) MachedS else MachedS | L = E ; | L = L ; | do S while ( B ) ;
| call id ( Elist ) ; | return E ;
OpenS -> if ( B ) S | if ( B ) MachedS else Opens
L -> L [ num ] | id
E -> E + G | G
G -> G * F | F
F -> ( E ) | num | id | real | string
B -> B || H | H
H -> H && I | I
I -> ! I | ( B ) | E Relop E | true | false
Relop -> < | <= | > | >= | == | !=
Elist -> Elist , E | 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。
1.1 系统框架图：
整个系统包括三个层次：外观表示层、常量数据层、处理层。外观表示层负责输出语法分析序列与错误信息的输出显示。常量数据层中包含四个部分：Lexical 类为实验一中对测试文件进行的词法分析后的 Token 序列，build_table 类为对文法文件进行 LR(1)分析后产生的 LR 分析表，还包含测试用例的 txt 文件和文法的 txt 文件。处理层中的 analyze 类是词法分析程序，用来对给定输入进行词法分析并对错误进行恐慌模式的错误处理，而 Lexical 类为词法分析程序，用于对测试文件进行词法分析。

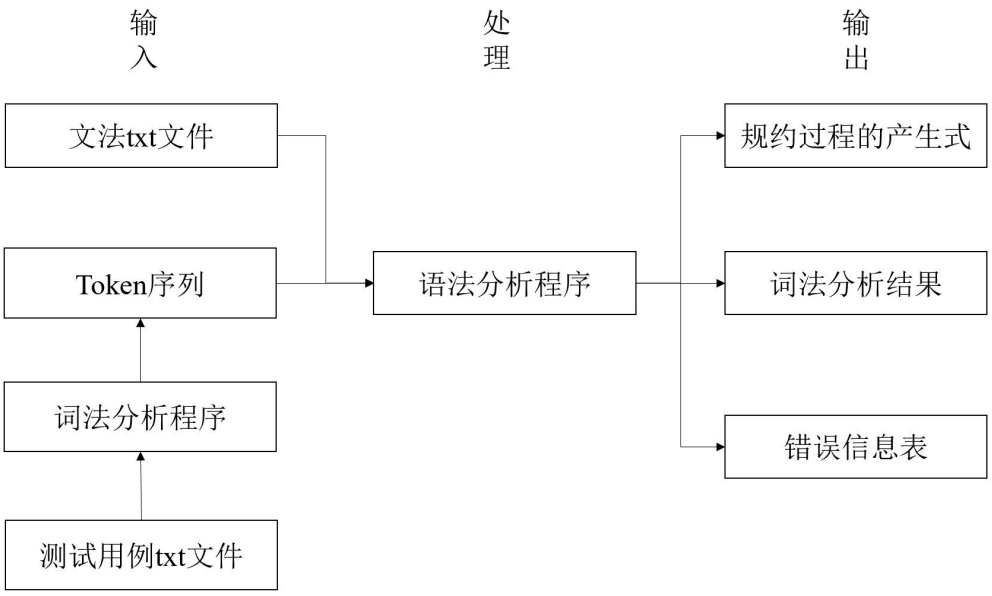
1.2 数据流图

数据流图包括三个部分：输入、处理、输出。

输入：首先需要输入测试用例的 txt 文件，然后调用词法分析分析程序将其进行词法分析的结果 Token 序列作为输入。然后需要输入文法 txt 文件用于 LR(1)分析。

处理：调用词法分析程序通过 LR 分析表对输入的 Token 序列进行 LR(1)分析，对于出现错误的情况进行恐慌模式的错误恢复策略。

输出：对语法分析后的规约过程中的产生式、语法分析结果和错误信息表进行 GUI 界面的输出。



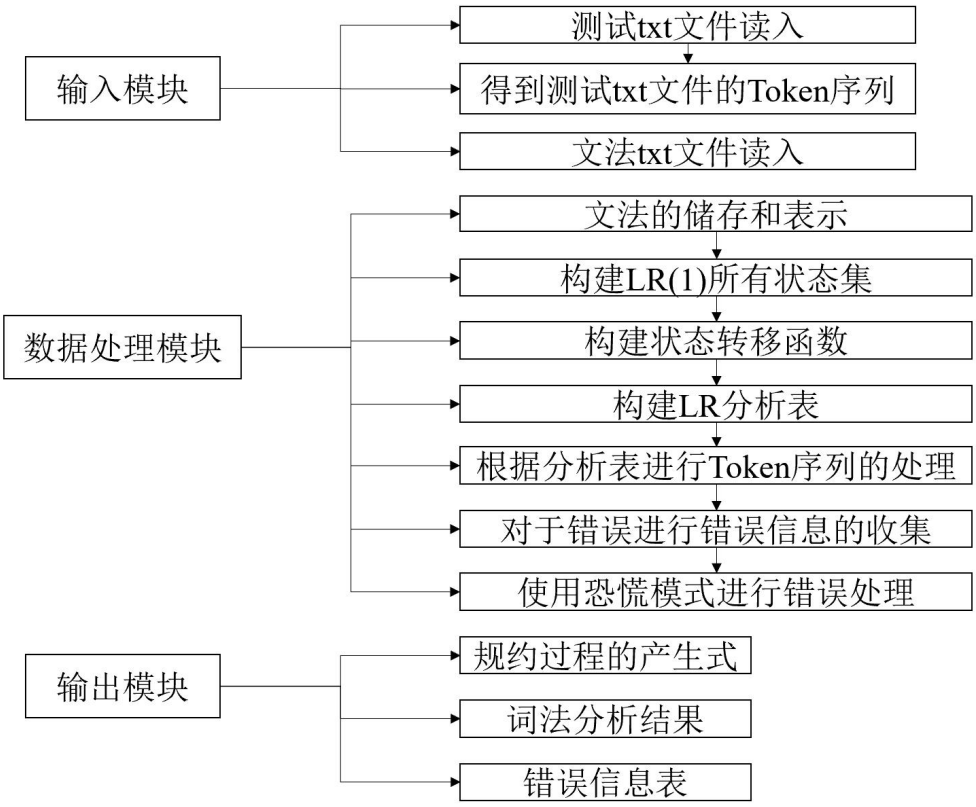
1.3 功能模块图

整个系统一共包括三个模块：输入模块、数据处理模块、输出模块

输入模块：负责通过 GUI 读入测试 txt 文件中的测试用例，并调用实验一的词法分析程序进行词法分析生成 Token 序列，同时读入文法 txt 文件。

数据处理模块：根据文法文件构建 LR(1)的所有状态集，构建 LR 分析表，根据 LR 文法分析表，根据 LR 分析表进行 Token 序列的处理，对于错误进行恐慌模式的处理。

输出模块：负责通过 GUI 输出语法分析后的规约过程中的产生式、词法分析结果和错误信息表。



（2）系统详细设计：对如下工作进行展开描述

核心数据结构的设计

主要功能函数说明

程序核心部分的程序流程图

✓ 核心数据结构的设计：

1.production 类：该类对应产生式，分别储存产生式左部和产生式右部，产生式右部的所有符号储存在一个 List 中，形式为”A -> BCD”。

```
public class production {
    private String production_left;  // 产生式左部
    private List<String> production_right_list;  // 产生式右部

    /**
     * 对输入的产生式进行分解，储存产生式左部和右部
     * @param s 产生式字符串
     */
    public production(String s) {
```

2.state_production 类：该类对应 LR(1)分析中的每一个包括展望符的产生式，相当于在 production 类的基础上加上展望符和”.”对应的位置，形式为”A -> B.CD, a”。

```java
public class state_production implements Cloneable{
    private production the_production;  // 产生式
    private String looking_forward_operator;  // 展望符
    private int index;  // .的位置

    /**
     * 在自动机中的每一个产生式的状态
     * @param the_production 产生式
     * @param looking_forward_operator 展望符
     * @param index .的位置
     */
    public state_production(production the_production, String looking_forward_operator, int index) {
```

3.state_lr1 类：该类对应 LR(1)分析中的每一个状态，包括的字段有状态号和 LR(1)状态集闭包。

```java
public class state_lr1 {
    private int state_id;  // lr1状态号
    private List<state_production> state_production_list;  //lr1状态集闭包

    /**
     * 初始化一个LR1状态
     * @param state_id  LR1状态号
     */
    public state_lr1(int state_id) {
```

4. grammer_init 类：该类用于一些程序需要用到的常量的初始化：可以从文法 txt 文件中读出非终结符集合、终结符集合和储存文件中的所有产生式，并包含求 first 集的方法。

```java
public class grammer_init {
    public static List<String> non_terminal = new ArrayList<>();  // 非终结符集合
    public static List<String> terminal = new ArrayList<>();  // 终结符集合
    public static List<String> all_symbol = new ArrayList<>();  // 所有符号
    public static Map<String, HashSet<String>> first = new HashMap<>();  // first集
    public static List<production> allproduction = new ArrayList<>();  // 文件中所有的产生式
```

5.build_table 类：该类主要用于 LR 分析表的创建

```java
public class build_table {
    private String begin_key;  // 文法开始符号
    private static String null_key = "ε";
    private static String init_key = "$";
    private static String error = "--";  // 错误符号
    private static String acc = "acc";  // ACC，接收成功符号
    private List<state_lr1> all_states_lr1 = new ArrayList<>();  // 所有状态
    private int statenum;  // lr1状态数
    private int actionLength;  // Action表列数
    private int gotoLength;  // GoTo表列数
    private String[] actionCol;  // Action表列名数组
    private String[] gotoCol;  // GoTo表列名数组
    private String[][] actionTable;  // Action表，与ppt一致，二维数组
    private int[][] gotoTable;  // GoTo表，与ppt不一样，包含所有的状态转化，二维数组，列中不含$

    //  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<>();  // 存储第x号lr1状态
    private ArrayList<Integer> gotoEnd = new ArrayList<>();  // 存储第y号lr1状态
    private ArrayList<String> gotoPath = new ArrayList<>();  // 存储S符号
```

6.analyze 类：该类用于调用 LR 分析表对词法分析的 Token 序列进行语法分析，输出分析后的结果，储存错误并对错误进行恐慌模式的错误恢复。

```java
public class analyze {
    private String text;  // 文法文件中的文法
    private List<String> allerror;  // 所有的错误
    private List<String> statute;  // 所有规约的式子
    private StringBuffer result = new StringBuffer();  // 语法分析的结果
    private List<Token> alltoken;  // 所有Token值
    private List<String> alltokenvalue;  // 转化为文法符号的Token值,加$
    private Stack<Integer> statestack;  // 状态栈
    private Stack<String> keystack;  // 符号栈
    private Stack<treenode> treestack;  // 树栈
    private String begin_key;  // 开始符号
    private static String init_key = "$";
    private static String error = "--";  // 错误符号
    private static String acc = "acc";  // ACC，接收成功符号\
    private Lexical myLexical;  //创建Lexical进行判断
```

7.treenode 类，用于在词法分析中构建词法分析树的节点，在规约时产生式右部的子节点全部加入产生式左部非终结符节点的 allchild 字段中。

```java
public class treenode implements Cloneable{
    private String val;  // 文法符号
    private List<treenode> allchild;  // 节点的所有子节点
    private String key;  // 属性值，对于有value值得文法符号才存在
    private int line;  // 行号

    /**
     * 初试化树的一个节点
     * @param val 树的文法符号
     * @param line 输的行数
     */
    public treenode(String val, int line) {
        this.val = val;
        this.allchild = new ArrayList<>();
        this.key = "";
        this.line = line;
    }
```

✓ 主要功能函数说明：

1 计算某个非终结符的 first 集：

```java
// 计算某个非终结符的first集
public static Set<String> findfirst(String temp) {
```

具体实现的思路如下：

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

（1.1）如果 X 是一个终结符，那么 FIRST(X)={X}。

（1.2）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，

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

（1.2.2）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)。

（1.3）如果 $X \rightarrow \varepsilon \in P$，那么将 ε 加入到 FIRST(X)中。

2.求状态集闭包：

根据下列式子构造状态集闭包。

$$CLOSURE(I) = I \cup \{[B \rightarrow .\gamma, b] \mid [A \rightarrow \alpha.B\beta, a] \in CLOSURE(I), B \rightarrow \gamma \in P, b \in FIRST(\beta a)\}$$

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. }
```

3.求 GOTO 表：

```
/**
 * 利用这个方法填充语法分析表的相关内容
 */
private void create_analyze_table() {
```

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.  }
```

4.构建所有状态集闭包：

```
/**
 * 利用这个递归方法建立一个用于语法分析的lr1自动机
 * 不再有项集可插入的判定标准为:
 */
private void create_allstate() {
```

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)
7.                  if(GOTO(I, X)非空且不在 C 中)
8.                      将 GOTO(I, X)加入 C 中；
9.      until 不再有新的项集加入到 C 中；
10. }
```

5.构建 LR 分析表：

```
/**
 * 利用这个方法填充语法分析表的相关内容
 */
private void create_analyze_table() {
```

伪代码如下所示。

```
1.  构造 G'的规范 LR(1)项集族 C={I,I,…,I}
2.  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：
3.  If [A→α·aβ,b]∈I and GOTO(I,a)=I
4.     ACTION[i,a]=sj
5.  If [A→α·Bβ,b]∈I and GOTO(I,B)=I
6.     GOTO[i,B]=j
7.  If [A→α·,a]∈I and A!=S'
8.     ACTION[i,a]=rj（j 是产生式 A→α的编号）
9.  If [S'→S·,#]∈I
10.    ACTION[i,#]=acc;
11. 没有定义的所有条目都设置为“error”，进行错误处理
```

6.LR 分析及恐慌模式的错误处理：

```
// 语法分析
public void grammer_analyze(String filename) {
```

本部分为整个实验最重要的实验：

其包括两部分：

第一种是在 LR 分析表中存在值的情况：即没有错误，并且为了输出树，在还建立了一个树节点栈，操作与符号栈同步，规约时让让规约后的节点的所有孩子节点为规约前的所有节点。伪代码如下：

```
1.  令 a 为 w$的第一个符号；
2.   while(1) { /* 永远重复*/
3.         令 s 是栈顶的状态；
4.         if ( ACTION [s，a] = st ) {
5.               将 t 压入栈中；
6.               令 a 为下一个输入符号；
7.         } else  if (ACTION [s，a] =归约 A → β ) {
8.               从栈中弹出 | β | 个符号；
9.               将 GOTO [t，A]压入栈中；
10.              输出产生式 A → β ；
11.        } else  if (ACTION [s，a] =接受) break；  /* 语法分析完成*/
12.        else 调用错误恢复例程；
13.     }
```

第二种是在 LR 分析表中不存在值的情况：即有错误，恐慌模式的错误处理思路如下：

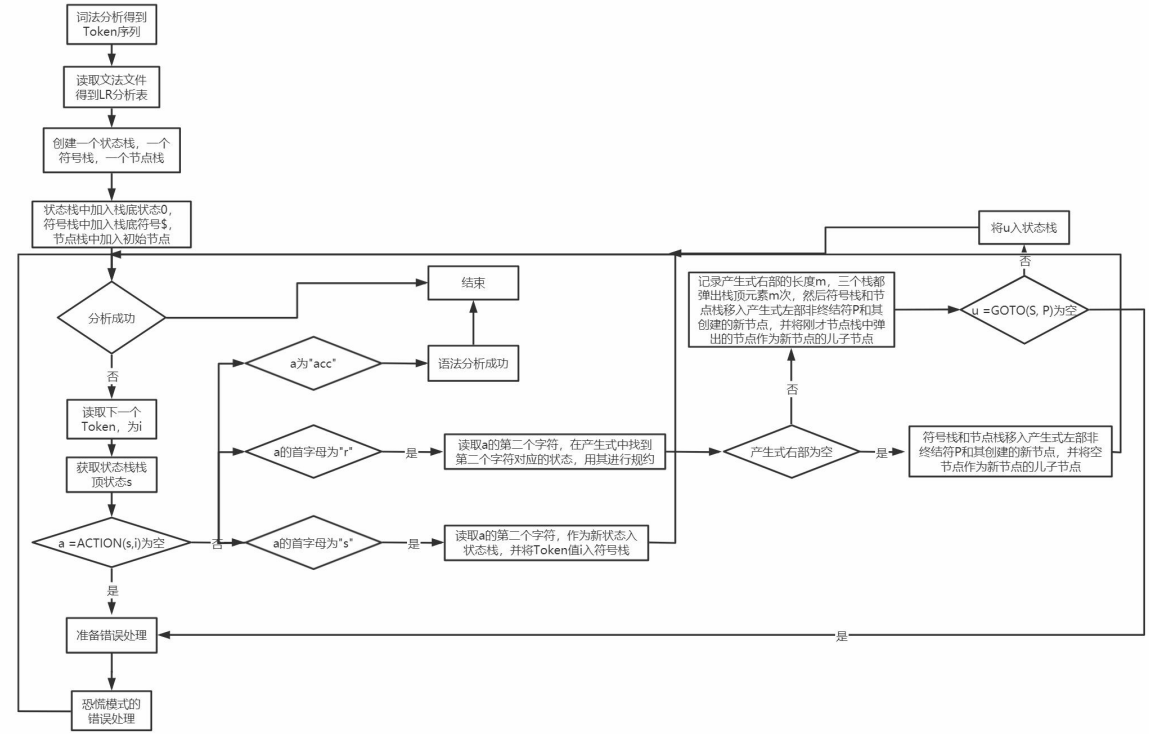
（1）从栈顶向下扫描，直到发现某个状态 si，它有一个对应于某个非终结符 A 的 GOTO 目标，可以认为从这个 A 推导出的串中包含错误。

（2）然后丢弃 0 个或多个输入符号，直到发现一个可能合法地跟在 A 之后的符号 a 为止。若找不到可以合法跟在 A 之后的符号，则继续执行（1）。

（3）之后将 si+1 = GOTO(si , A)压入栈中，继续进行正常的语法分析。

恐慌模式的错误处理可以理解为一种规约，只不过这种规约不是规约的句柄。

✓ 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；

（1.1）文法的设计：

实验指导书中的文法具有二义性， 对于比如结构体、指针、逻辑与、逻辑或等很多语法无法进行识别和分析，所以在实验指导书的基础上，参考网上的一些其他的文法，对文法进行了重新设计。

（1.2）规约过程中空产生式的处理：

由于文法含有空产生式，所以在使用 LR 分析表进行规约时与常规的规约方式不同，常规的规约方式是对符号栈中的规约产生式的右部进行规约，这个过程需要将状态栈和符号栈弹出很多次，次数为产生式右部符号的个数，然后将产生式左部的非终结符入符号栈。而对于空产生式来说，只需要将产生式左部的非终结符入栈即可。

（1.3）结果的输出：

由于实验要求的输出格式为先序输出树的格式，所以需要在规约的过程中建立规约树，创建一个新的节点类 treenode，并且建立一个树栈 treestack，树栈的操作和符号栈同步，只不过将操作对象是对于树节点。在规约时将规约式左部的非终结符对应的节点作为父亲节点，然后将树栈需要弹出的产生式右部符号对应的节点作为父亲节点的儿子节点，这样当最后规约出开始符号后，规约树也随之建好了。

（1.4）错误处理：

使用了 ppt 课件中提到的恐慌模式的错误处理，但是在 ppt 的错误处理基础上进行了改进。对于在 Token 识别过程中遇到 ACTION 表的状态转移结果为 error 的情况，需要打印错误信息并

进行错误处理。具体的错误处理过程为：

1. 1. 得到状态栈的栈顶状态 S 。
2. 1.1 如果 S 有对于某个非终结符 A 的 GOTO 表的值 S1 。
3. 1.1.1 观察 S 在 ACTION 表中对于哪些终结符可以有操作，然后看剩下的输入符号 。
4. 中是否包含这些终结符，如果包含，就将输入符号的指针移到这个终结符所在的位置，并将 A 入符号栈，S1 入状态栈，A 对应的节点入树栈，相当于进行了一次规约。
5. 1.1.2 S 在 ACTION 表中没有终结符可以有操作，则执行 1.1，遍历下一个非终结符。
6. 1.2 如果 1.1 中所有情况都不满足，弹出状态栈、符号栈、树栈的栈顶元素，并回到 1 。

（1.5）First 集的求法:

First 集正常求并不难，但需要尤其注意空值，这个容易导致错误。

（2）输出该句法分析器的分析表；

由于分析表太大，只截取了其中一部分粘贴，具体的分析表文件已放在项目里，文件名为“LR1 语法分析表.txt”

	ε	proc	id	(	)	{	}	record	;	struct	*	=	,	int	float	bool	char	[	num	]	if	else
0	--	s7	s271	--	--	--	--	s412	--	s421	--	--	--	s92	s93	s94	s95	--	--	--	s433	--
1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
2	--	s7	s271	--	--	--	--	s412	--	s421	--	--	--	s92	s93	s94	s95	--	--	--	s433	--
3	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
4	--	s7	s271	--	--	--	--	s412	--	s421	--	--	--	s92	s93	s94	s95	--	--	--	s433	--
5	--	s7	s271	--	--	--	--	s412	--	s421	--	--	--	s92	s93	s94	s95	--	--	--	s433	--
6	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
7	--	--	--	--	--	--	--	--	--	--	--	--	--	s387	s388	s389	s390	--	--	--	--	--
8	--	--	s9	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
9	--	--	--	s10	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
10	--	--	--	--	--	--	--	--	--	--	--	--	--	s387	s388	s389	s390	--	--	--	--	--
11	--	--	--	--	s12	--	--	--	--	--	--	--	s384	--	--	--	--	--	--	--	--	--
12	--	--	--	--	--	s13	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
13	--	s21	s271	--	--	--	r3	s30	--	s65	--	--	--	s92	s93	s94	s95	--	--	--	s96	--
14	--	--	--	--	--	--	s15	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
15	--	r5	r5	--	--	--	--	r5	--	r5	--	--	--	r5	r5	r5	r5	--	--	--	r5	--
16	--	s21	s271	--	--	--	r3	s30	--	s65	--	--	--	s92	s93	s94	s95	--	--	--	s96	--
17	--	--	--	--	--	--	r1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
18	--	s21	s271	--	--	--	r3	s30	--	s65	--	--	--	s92	s93	s94	s95	--	--	--	s96	--

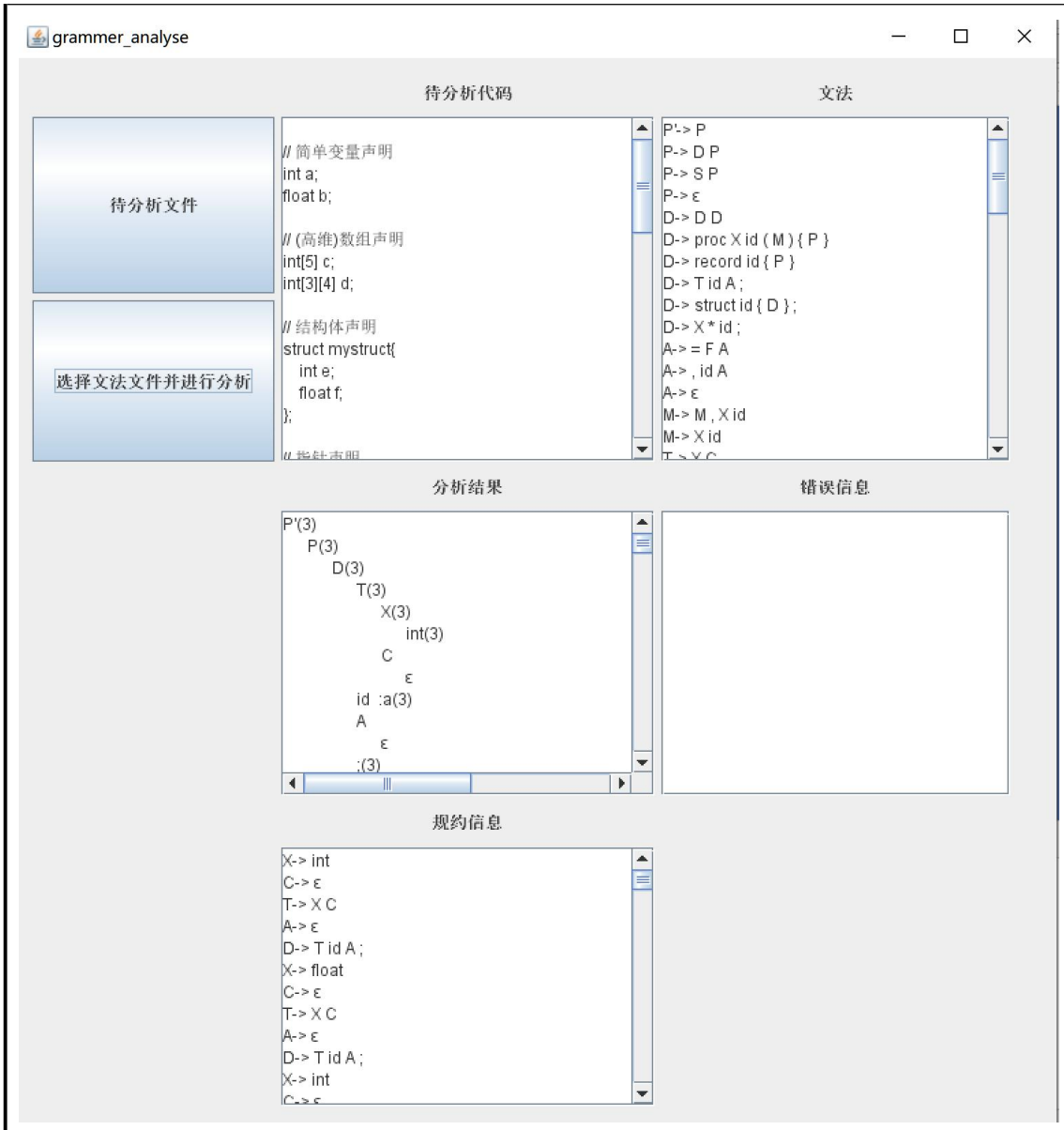
（3）针对某测试程序输出其句法分析结果；

满足实验验收要求的测试的文件名为：“grammer_test.txt”，内容如下：（注：这些测试用例里面部分内容的子测试文件名为“declaration.txt”，“do_while.txt”，“expression_and_assignment.txt”，“if_else_then.txt”）

```
// 简单变量声明
int a;
float b;

// (高维)数组声明
int[5] c;
int[3][4] d;

// 结构体声明
struct mystruct{
    int e;
    float f;
};

// 指针声明
```

```
int *g;

// 记录声明
record stu {
    int h = 1;
}

// 过程声明
proc int function(int i, int j){
     i = j + 1;
     return i;
}

// 算术表达式
k = k + 1;

// 关系与逻辑表达式，分支语句
if ((l == m) && ( m > 1))
    l = 1;
else
    m = 2;

// 高维数组引用和赋值
m = n[1][2];
n[1][2] = m;

// 循环语句
do
   o = o + 1;
while (o < p);

// 过程调用
call function(q, r);
```

结果如下：具体的输出结果太大所以以文件“grammer_test 的语法分析结果.txt”给出，已放在项目里

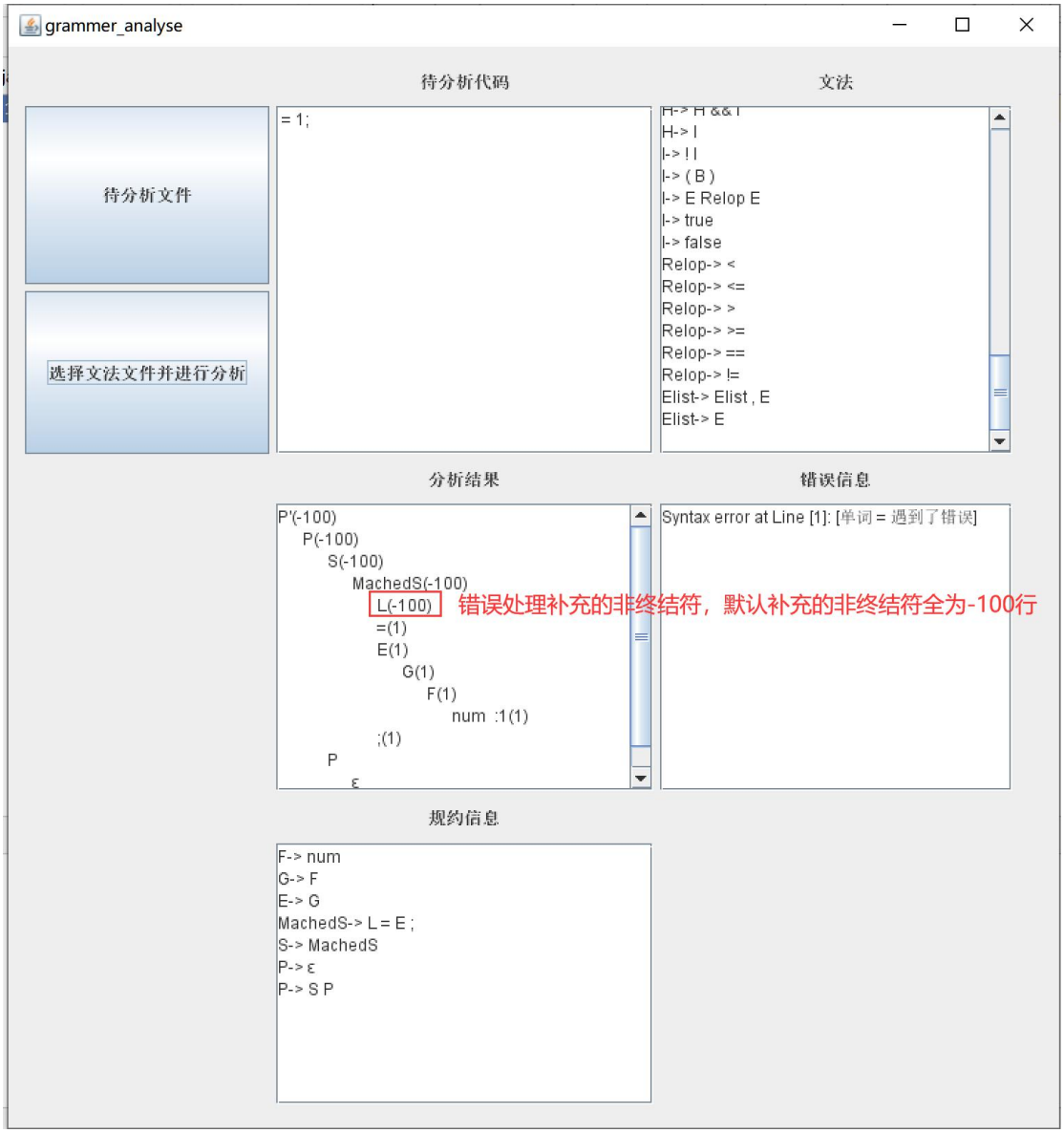


（4）输出针对此测试程序对应的语法错误报告；

错误处理的测试用例文件名为：“error.txt”，内容如下：

```
1.  = 1;
```

进行错误处理后会为其在等号左边补上一个非终结符，结果如下：



（5）对实验结果进行分析

首先分析测试文件：包含了实验验收要求的所有语句：简单变量的声明语句，（高维）数组声明语句，结构体声明语句，指针声明语句，过程声明语句，记录声明语句，表达式（算术、关系、逻辑），（高维）数组引用，（高维）数组赋值，分支语句，循环语句，过程调用语句。然后对输出结果进行分析，对于这些语句都完成了识别。

然后分析错误处理文件，按照 Syntax error at Line [行号]: [说明文字]的格式输出了错误，然后分析结果，使用了恐慌模式补充了缺失的非终结符，进行了错误处理。

综上，完成了实验要求。

指导教师评语：

日期：