

编译原理课程实验报告

实验 2：语法分析

姓名	朱明彦	院系	计算机学院	学号	1160300314	
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	2019/04/21		
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。						
1. 能够识别声明语句、表达式以及赋值语句、分支语句和循环语句。						
2. 可以自动计算 Closure 和 GOTO 函数，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。						
二、文法设计					得分	
要求：给出如下语言成分的文法描述。						
➤ 声明语句（变量声明）						
➤ 表达式及赋值语句						
➤ 分支语句：if_then_else						
➤ 循环语句：do_while						
完整语法见下页						
其中主要包含声明语句（包含普通变量的声明以及数组元素的声明），赋值语句，基本运算语句（包含加法和乘法运算），分支语句的声明（包含if-else和if两种），循环语句的声明（do-while语句），过程和结构体语句（record）的声明。						

1. Start \rightarrow P
2. P \rightarrow D P | S P | ϵ
3. D \rightarrow proc X id (M) { P } | record id { P } | T id A ;
4. A \rightarrow = F A | , id A | ϵ
5. M \rightarrow M , X id | X id
6. T \rightarrow X C
7. X \rightarrow int | float | bool | char
8. C \rightarrow [num] C | ϵ
9. S \rightarrow MachedS | OpenS
10. MachedS \rightarrow if (B) MachedS else MachedS | L = E ; | do S while (B) ;
| call id (Elist) ; | return E ;
11. OpenS \rightarrow if (B) S | if (B) MachedS else Opens
12. L \rightarrow L [num] | id
13. E \rightarrow E + G | G
14. G \rightarrow G * F | F
15. F \rightarrow (E) | num | id | real | string
16. B \rightarrow B || H | H
17. H \rightarrow H && I | I
18. I \rightarrow ! I | (B) | E Relop E | true | false
19. Relop \rightarrow < | <= | > | >= | == | !=
20. Elist \rightarrow Elist , E | E

三、系统设计

得分

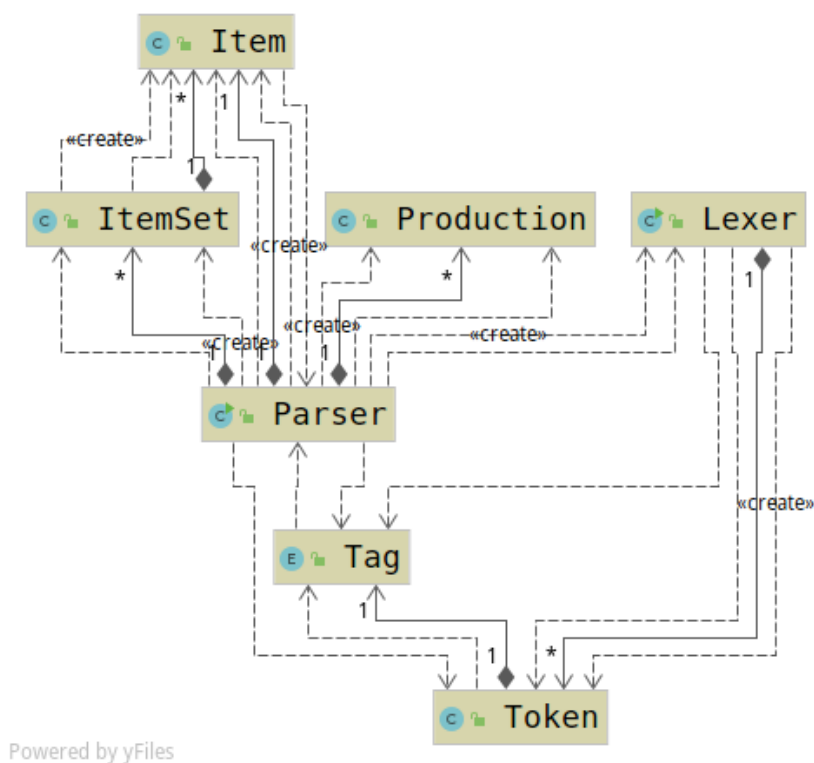
要求：分为系统概要设计和系统详细设计。

（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。

（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

（1）语法分析器的整体 UML 图如下所示



Parser 为语法分析器的主类，其负责完成文法的识别，自动计算闭包，生成 LR 分析表以及对 **Lexer** 分析源文件得到的 **token** 序列进行自底向上分析。

Item 为 LR 项目的类，其表示 LR 分析法中的项目，包含产生式以及当前的状态。

ItemSet 类为对于项目集类，表示 LR(1) 中的项目集的概念。

Production 类，则表示文法中的产生式，包含产生式的左部、右部。

对于 **Lexer**、**Token** 以及 **Tag** 类，均为词法分析中的相应类，分别表示词法分析器，词法分析结果的 **token**（词素），以及 **token** 中的 **Tag** 标签。

(2)

(i) 核心数据结构

核心的数据结构主要有三种，分别是 **Map**、**Set** 和二维数组。

其中 **Map** 用来记录非终结符的 **first** 集，以及针对每个项目闭包的 **GOTO** 表。**Set** 用来记录文法中的各类产生式、终结符、非终结符以及项目集族。

用 2 维数组记录 LR 分析表。

(ii) 主要函数说明

getFirst() 函数用来计算所有非终结符的 **First** 集，由于 LR(1) 文法可能有左递归（包含直接左递归和间接左递归），所以求 **First** 集的步骤为三步，首先初始化所有的非终结符的 **First** 集为空集，然后利用 LL 分析中求 **First** 集的方法依次对所有的非终结符求 **First** 集，直到所有的非终结符对应的 **First** 集均不发生变化；最后将能导出空且包含左递归的产生式中的左部进行进一步的求 **First** 集，得到的即为所有的非终结符的 **First** 集。

getFirstFromString() 用于计算某一个串的 **First** 集。

getClosure() 函数用于求某一个项目的闭包。

goTo() 函数用来求某一闭包的转移，首先需要判断针对某一个文法符号是否有可以转移到的项目，其次再看该项目的闭包是否已经存在即可。

items() 函数用来计算该文法的项目集族以及内部的转移。

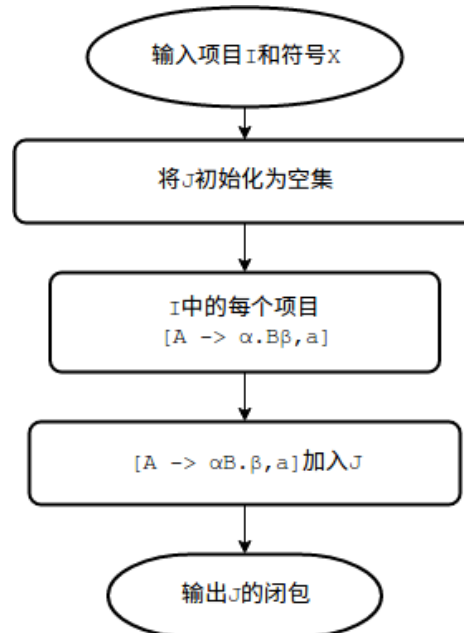
fillLrTable()函数用来填 LR 分析表。

reduce()函数是将给定的 token 序列进行规约，并返回对应的产生式序列。

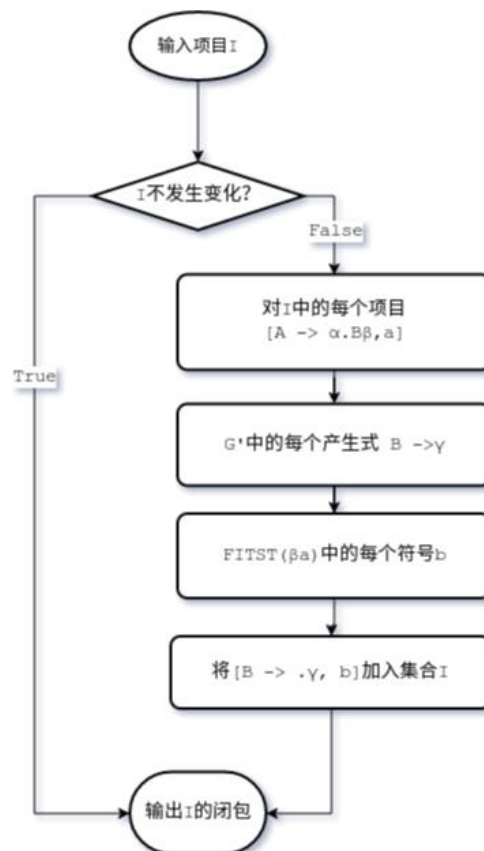
(iii) 主要函数的流程图

由于针对整个 Parser 而言其处理流程过于复杂，故此处仅仅给出核心函数的处理流程图。

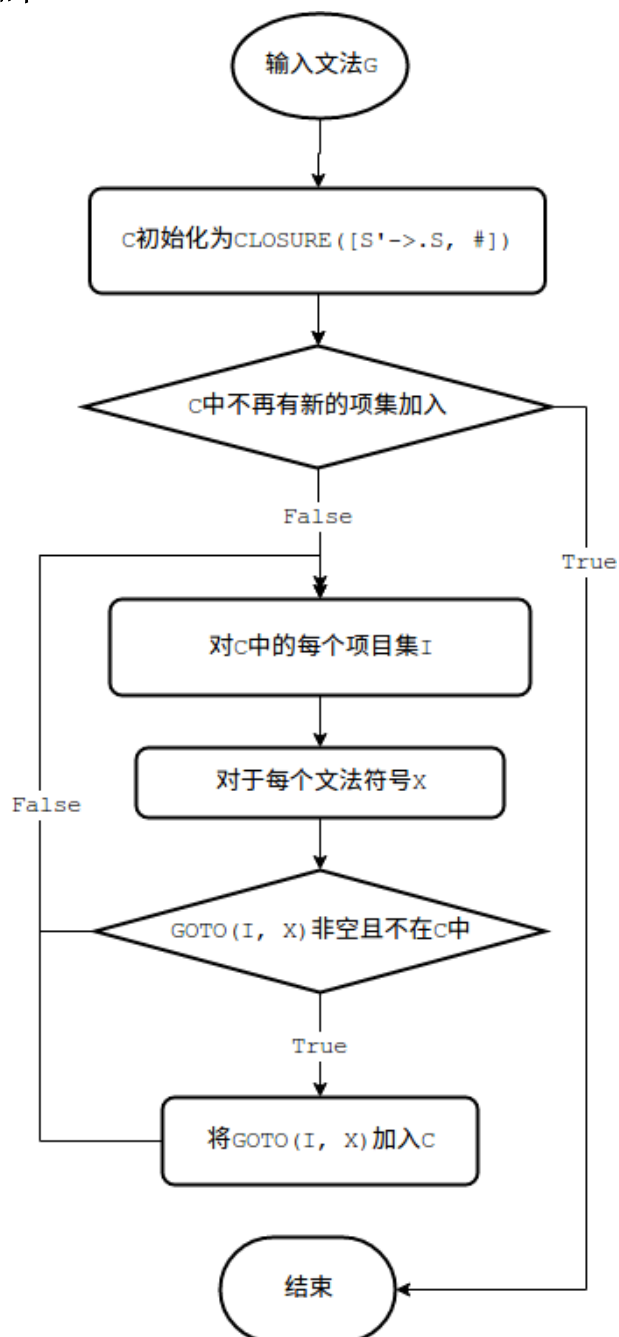
Closure 闭包函数流程图如下



goTo(Item, x)函数流程图如下



items 函数流程图如下



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；
- (2) 输出该句法分析器的分析表；
- (3) 针对一测试程序输出其句法分析结果；
- (4) 输出针对此测试程序对应的语法错误报告；
- (5) 对实验结果进行分析。

注：其中的测试样例需先用已编写的词法分析程序进行处理。

- (1) 系统实现中遇到的问题

实验中遇到的主要问题在错误处理、错误恢复以及错误信息的提示，如果仅仅使用恐慌模式进行不断读入，会导致当前的状态栈和符号栈中的信息仍然与输入的串不符合，会导致进一步的向后的正确的代码段被掩盖。

因此，最终采用的是类似恐慌模式的错误处理，即先弹出状态栈中的栈顶状态，利用状态栈和符号栈相差为一的性质，对符号栈栈顶的非终结符和状态栈栈顶的状态，寻找 GOTO 表中非空的部分，并将其压入状态栈，以此模拟以此规约过程，并弹出了前面错误的已规约的串（此时可以维护状态栈和符号栈高度相同这一性质）。

然后寻找状态栈栈顶和输入符号中可以进行 ACTION 的输入符号，即可以恢复正常的语法分析器的动作。

可以利用出错时的输入符号和最终恢复正常的语法分析的符号进行推测出可能的出错信息，并作为错误信息的提示。

(2) 文法的 LR 分析表

				&&		<=		Opens	string		bool		num		do
0											s9				s1
1															s29
2															
3															
4															
5															
6											r(S -> OpenS)				r(S -> OpenS)
7											r(S -> MachedS)				r(S -> MachedS)
8											s9				s1
9															
10															
11															
12											s9				s1
13															
14															
15											s40				
16															
17															
18															
19								s25					s21		
20								s70					s63		
21															
22															
23								s56					s52		
24															
25															
26															
27															
28															
29															s29
30															
31															
32															
33															
34															
35															

由于该文法最终产生的 LR 分析表有 425 个状态，且有几十个文法符号，故其最终的表格非常大，此处仅仅列出了部分表，详细 LR 分析表见压缩包中 parser/LRTable.txt 文件。

(3) 首先给出测试程序的源码文件

如下所示。

```

1  int a = 10;
2  a = x + y;
3  /*int a = x + 1;*/
4  int [100][200] b;
5  a = c;
6  int d = 0;
7  if (a == c)
8      d = 1;
9  else
10     d = 2;
11
12  /**Hello world!
13  Hello Java! **/
14
15  do
16      a = a + 1;
17  while (a < 20);
18
19  float f = 10.0;
20
21  char [10] s = "Hello";
22
23  proc int function(int x, int y){
24      y = x + 1;
25      return y;
26  }
27
28  call function(a, d);
29  record stu {
30      int z = 1;
31  }
32  int xx
33  int yy;

```

语法分析的结果即为产生式序列如下：

$X \rightarrow \text{int}$
 $C \rightarrow \epsilon$
 $T \rightarrow X C$
 $F \rightarrow \text{num}$
 $A \rightarrow \epsilon$
 $A \rightarrow = F A$
 $D \rightarrow T \text{ id } A ;$
 $L \rightarrow \text{id}$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow E + G$
 $\text{MachedS} \rightarrow L = E ;$
 $S \rightarrow \text{MachedS}$
 $X \rightarrow \text{int}$
 $C \rightarrow \epsilon$
 $C \rightarrow [\text{ num }] C$
 $C \rightarrow [\text{ num }] C$
 $T \rightarrow X C$
 $A \rightarrow \epsilon$
 $D \rightarrow T \text{ id } A ;$
 $L \rightarrow \text{id}$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $\text{MachedS} \rightarrow L = E ;$
 $S \rightarrow \text{MachedS}$
 $X \rightarrow \text{int}$
 $C \rightarrow \epsilon$
 $T \rightarrow X C$
 $F \rightarrow \text{num}$
 $A \rightarrow \epsilon$
 $A \rightarrow = F A$
 $D \rightarrow T \text{ id } A ;$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $\text{Relop} \rightarrow ==$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $I \rightarrow E \text{ Relop } E$

$H \rightarrow I$
 $B \rightarrow H$
 $L \rightarrow \text{id}$
 $F \rightarrow \text{num}$
 $G \rightarrow F$
 $E \rightarrow G$
 $\text{MachedS} \rightarrow L = E ;$
 $L \rightarrow \text{id}$
 $F \rightarrow \text{num}$
 $G \rightarrow F$
 $E \rightarrow G$
 $\text{MachedS} \rightarrow L = E ;$
 $\text{MachedS} \rightarrow \text{if } (B) \text{ MachedS else}$
 MachedS
 $S \rightarrow \text{MachedS}$
 $L \rightarrow \text{id}$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $F \rightarrow \text{num}$
 $G \rightarrow F$
 $E \rightarrow E + G$
 $\text{MachedS} \rightarrow L = E ;$
 $S \rightarrow \text{MachedS}$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $\text{Relop} \rightarrow <$
 $F \rightarrow \text{num}$
 $G \rightarrow F$
 $E \rightarrow G$
 $I \rightarrow E \text{ Relop } E$
 $H \rightarrow I$
 $B \rightarrow H$
 $\text{MachedS} \rightarrow \text{do } S \text{ while } (B) ;$
 $S \rightarrow \text{MachedS}$
 $X \rightarrow \text{float}$
 $C \rightarrow \epsilon$
 $T \rightarrow X C$
 $F \rightarrow \text{real}$
 $A \rightarrow \epsilon$
 $A \rightarrow = F A$
 $D \rightarrow T \text{ id } A ;$
 $X \rightarrow \text{char}$

$C \rightarrow \epsilon$
 $C \rightarrow [\text{ num }] C$
 $T \rightarrow X C$
 $F \rightarrow \text{string}$
 $A \rightarrow \epsilon$
 $A \rightarrow = F A$
 $D \rightarrow T \text{ id } A ;$
 $X \rightarrow \text{int}$
 $X \rightarrow \text{int}$
 $M \rightarrow X \text{ id}$
 $X \rightarrow \text{int}$
 $M \rightarrow M , X \text{ id}$
 $L \rightarrow \text{id}$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $F \rightarrow \text{num}$
 $G \rightarrow F$
 $E \rightarrow E + G$
 $\text{MachedS} \rightarrow L = E ;$
 $S \rightarrow \text{MachedS}$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $\text{MachedS} \rightarrow \text{return } E ;$
 $S \rightarrow \text{MachedS}$
 $P \rightarrow \epsilon$
 $P \rightarrow S P$
 $P \rightarrow S P$
 $D \rightarrow \text{proc } X \text{ id } (M) \{ P \}$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $\text{Elist} \rightarrow E$
 $F \rightarrow \text{id}$
 $G \rightarrow F$
 $E \rightarrow G$
 $\text{Elist} \rightarrow \text{Elist} , E$
 $\text{MachedS} \rightarrow \text{call id } (\text{Elist}) ;$
 $S \rightarrow \text{MachedS}$
 $X \rightarrow \text{int}$
 $C \rightarrow \epsilon$
 $T \rightarrow X C$
 $F \rightarrow \text{num}$

$A \rightarrow \epsilon$
 $A \rightarrow = F A$
 $D \rightarrow T \text{ id } A ;$
 $P \rightarrow \epsilon$
 $P \rightarrow D P$
 $D \rightarrow \text{record id } \{ P \}$
 $X \rightarrow \text{int}$
 $C \rightarrow \epsilon$
 $T \rightarrow X C$
 $A \rightarrow \epsilon$
 $D \rightarrow T \text{ id } A ;$
 $P \rightarrow \epsilon$
 $P \rightarrow D P$
 $P \rightarrow D P$
 $P \rightarrow S P$
 $P \rightarrow D P$
 $P \rightarrow D P$
 $P \rightarrow D P$
 $P \rightarrow S P$
 $P \rightarrow S P$
 $P \rightarrow D P$
 $P \rightarrow S P$
 $P \rightarrow D P$
 $P \rightarrow S P$
 $P \rightarrow D P$

(4) 错误报告

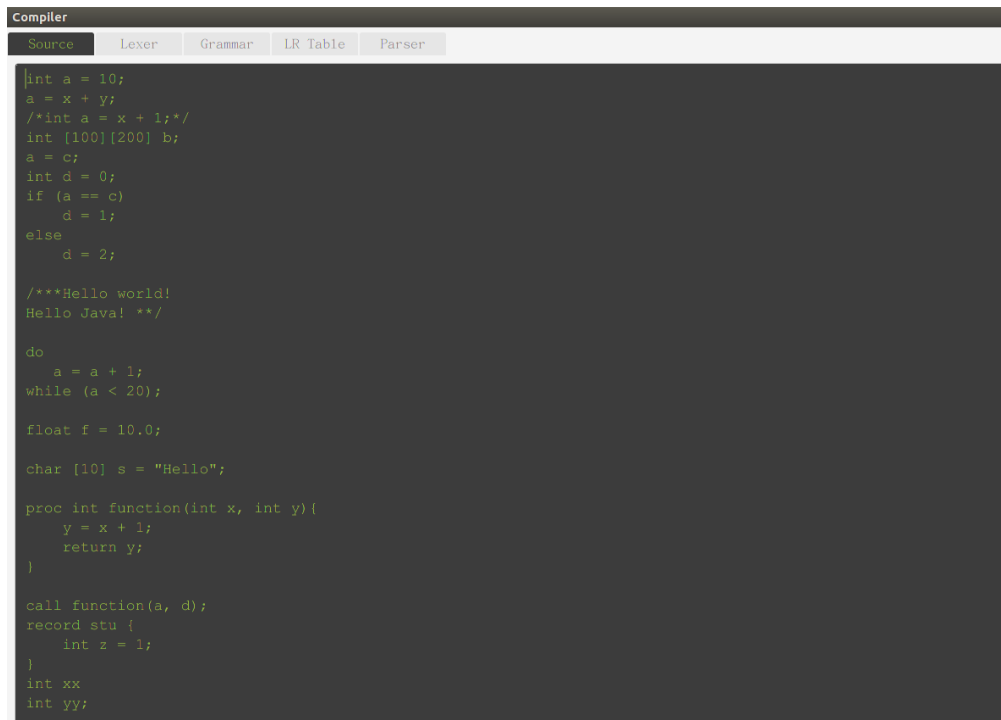
Error at line[33], Maybe missing a ";"

即通过前面产生错误的符号和后面最终能够重新进行合理规约的符号进行推断,对于源文件中存在的句子末尾缺失分号的情况,可以通过前后均为关键字来推断错误信息,如上所示。

(5) 分析结果

可以看到, LR 的自底向上分析可以在最后规约出来 P, 即证明可以规约出来文法的开始符号。其中词法分析能够合理的过滤掉注释和空符号, 保证后面的语法分析正确执行。

其中测试的 GUI 界面如下所示



```
Compiler
Source  Lexer  Grammar  LR Table  Parser

int a = 10;
a = x + y;
/*int a = x + 1;*/
int [100][200] b;
a = c;
int d = 0;
if (a == c)
    d = 1;
else
    d = 2;

/**Hello world!
Hello Java! */

do
    a = a + 1;
while (a < 20);

float f = 10.0;

char [10] s = "Hello";

proc int function(int x, int y){
    y = x + 1;
    return y;
}

call function(a, d);
record stu {
    int z = 1;
}
int xx
int yy;
```

Compiler

Source | **Lexer** | Grammar | LR Table | Parser

Tokens Errors

```
<INT>
<ID, a>
<ASSIGN>
<NUM, 10>
<SEMICOLON>
<ID, a>
<ASSIGN>
<ID, x>
<ADD>
<ID, y>
<SEMICOLON>
<INT>
<MLP>
<NUM, 100>
<MRP>
<MLP>
<NUM, 200>
<MRP>
<ID, b>
<SEMICOLON>
<ID, a>
<ASSIGN>
<ID, c>
<SEMICOLON>
<INT>
<ID, d>
<ASSIGN>
```

Compiler

Source | Lexer | **Grammar** | LR Table | Parser

```
Start -> P
P -> D P | S P | e
D -> proc X id ( M ) { P } | record id { P } | T id A ;
A -> = F A | , id A | e
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 ; | do S while ( B ) ; | call id ( Elist
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
```

