

# Memoria de la Práctica de Procesadores de Lenguajes: Analizador Léxico

Diego José Abengózar Vilar, Alejandro García Castellanos,  
Ignacio Javier Encinas Ramos

Grupo 82

November 26, 2019

## Índice

<b>1</b>	<b>Diseño del Analizador Léxico</b>	<b>2</b>
1.1	Tokens . . . . .	2
1.2	Gramática . . . . .	2
1.3	Autómata Finito Determinista . . . . .	3
1.4	Acciones Semánticas . . . . .	4
1.5	Errores . . . . .	5
1.6	Matriz de Transiciones . . . . .	5
<b>2</b>	<b>Tabla de Símbolos: Estructura e implementación</b>	<b>5</b>
<b>3</b>	<b>Anexo de Pruebas</b>	<b>6</b>

# 1 Diseño del Analizador Léxico

## 1.1 Tokens

<PuntoComa, - >  
<CorcheteAbrir, - >  
<CorcheteCerrar, - >  
<ID, posTS> (Identificador)  
<ENT, valor> (Dato de tipo entero)  
<CAD, lex> (Dato de tipo cadena)  
<ParentesisCerrar, - >  
<ParentesisAbrir, - >  
<SUMA, - > (Operador suma)  
<MENOR, - > (Operador lógico menor)  
<NOT, - > (Operador lógico de negación)  
<ASIG, - > (Operador de asignación)  
<ASIGOR, - > (Asignación con o lógico)  
<DEC, - > (“var”)  
<TipoVarENT, - > (“int”)  
<TipoVarLOG, - > (“boolean”)  
<TipoVarCAD, - > (“string”)  
<Print, - >  
<Input, - >  
<Coma, - >  
<Return, ->  
<DECFunc, - > (“function”)  
<IF, - >  
<ELSE, - >

## 1.2 Gramática

$G(N, T, S, P)$

$S = A$

$N = \{ A, B, C, D, E, F, G, H \}$

$T = \{ del, ;, \{, \}, (, ), +, <, !, =, ,, l, d, ', /, -, *, c \}$

$P:$

$A \rightarrow delA \mid ; \mid \{ \mid \} \mid ( \mid ) \mid + \mid < \mid ! \mid = \mid ,$

$A \rightarrow \mid B \mid lC \mid dD \mid 'E \mid /F$

$B \rightarrow =$

$C \rightarrow lC \mid dD \mid _C \mid \lambda$

$D \rightarrow dD \mid \lambda$

$E \rightarrow cE \mid *E \mid /E \mid ' ,$

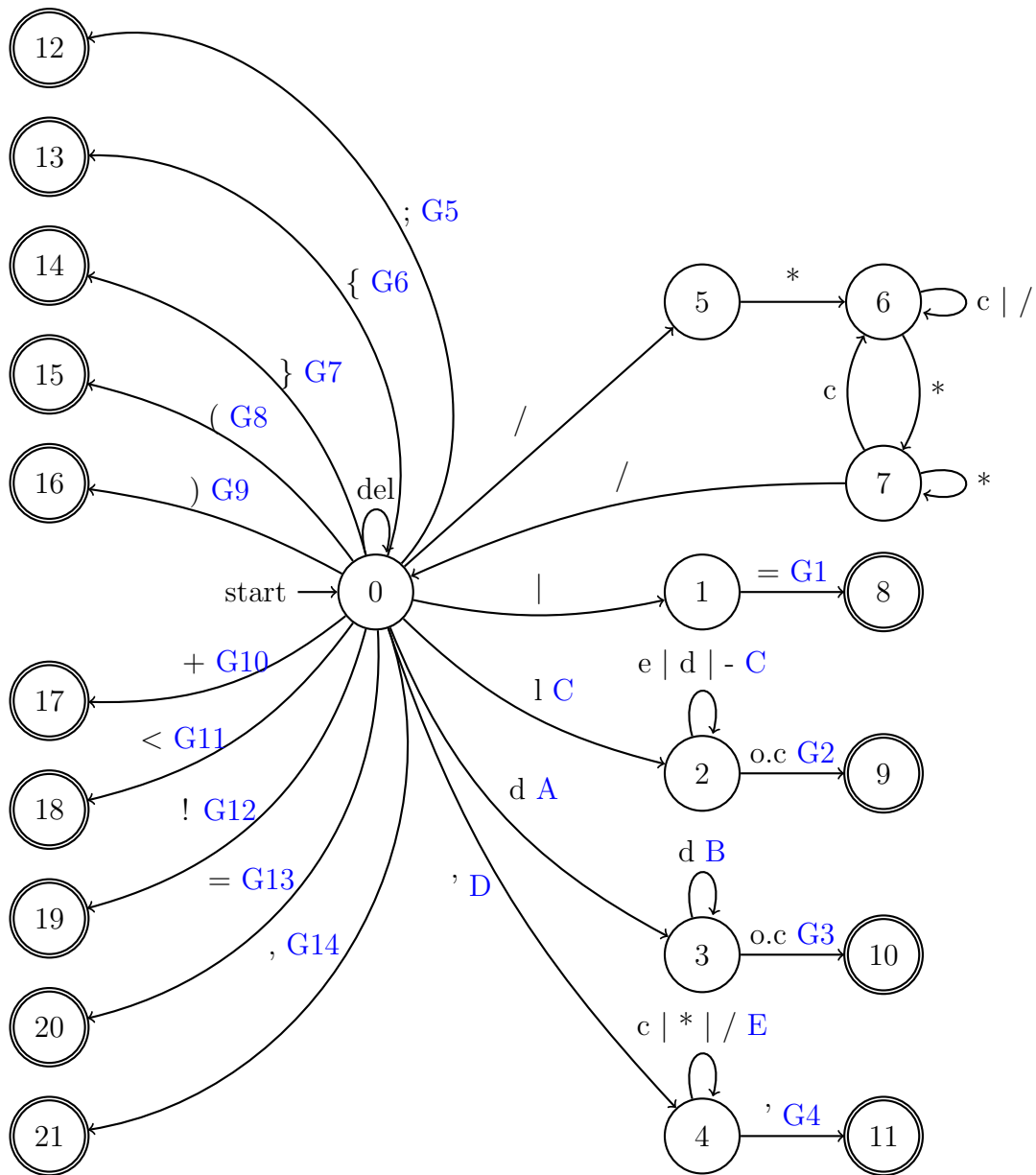
$F \rightarrow *G$

$G \rightarrow cG \mid /G \mid *H$

$H \rightarrow /A \mid cG \mid *H$

Donde,  $c = T - \{*, /\}$

### 1.3 Autómata Finito Determinista



## 1.4 Acciones Semánticas

Lee  $\forall$  transicion menos o.c

C: CONCAT()

$G_1$ : GEN\_TOKEN(ASIGOR, -)

```
 $G_2$ : if (lex  $\in$  palRes) GEN_TOKEN(palRes, -)
else if (FlagDeclUso = Decl)
    if (estaEnTSActual(lex))
        Error("Variable ya declarada")
    else
        p = INSERTAR_TS(lex)
        GEN_TOKEN(ID, p)
else
    p = BUSCA_TS(lex)
    if (p = null) p = INSERTAR_TS(lex)
    GEN_TOKEN(ID, p)
```

A: num = valor(d)

B: num = num \* 10 + valor(d)

D: cont = 0

E: cont = cont + 1  
CONCAT()

$G_3$ : if (num  $\geq 2^{15}$ ) Error("Numero se sale del rango")  
else GEN\_TOKEN(ENT, num)

$G_4$ : if (num > 64) Error("Exceso de caracteres en la cadena")  
else GEN\_TOKEN(CAD, lex)

$G_5$ : GEN\_TOKEN(PuntoComa, -)

$G_6$ : GEN\_TOKEN(CorcheteAbrir, -)

$G_7$ : GEN\_TOKEN(CorcheteCerrar, -)

$G_8$ : GEN\_TOKEN(ParentesisAbrir, -)

$G_9$ : GEN\_TOKEN(ParentesisCerrar, -)

$G_{10}$ : GEN\_TOKEN(SUMA, -)

$G_{11}$ : GEN\_TOKEN(MENOR, -)

$G_{12}$ : GEN\_TOKEN(NOT, -)

$G_{13} : \text{GEN\_TOKEN}(\text{ASIG}, -)$

$G_{14} : \text{GEN\_TOKEN}(\text{Coma}, -)$

Donde,  $\text{palRes} = \{\text{var}, \text{int}, \text{boolean}, \text{string}, \text{print}, \text{input}, \text{function}, \text{return}, \text{if}, \text{else}\}$

## 1.5 Errores

Los errores que pueden ocurrir son errores de transiciones imprevistas, error de que un número esté fuera de rango y error de identificador ya declarado previamente (cuando el  $\text{FlagDeclUso} = \text{DECL}$ ).

## 1.6 Matriz de Transiciones

MT_AFD		letra	digito	'	/	-	carácter	*	delimitador
$\rightarrow 0$	1 lee	2 C	3A	4 D	5 lee	-1 error	-1 error	-1 error	0 lee
1	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error
2	9 G2	2 C	2 C	9 G2	9 G2	2 C	9 G2	9 G2	9 G2
3	10 G3	10 G3	3 B	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3
4	4 E	4 E	4 E	11 G4	4 E	4 E	4 E	4 E	4 E
5	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	6 lee	-1 error
6	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	7 lee	6 lee
7	6 lee	6 lee	6 lee	6 lee	0 lee	6 lee	6 lee	7 lee	6 lee

MT_AFD	;	{	}	(	)	+	<	!	=	,
$\rightarrow 0$	12 G5	13 G6	14 G7	15 G8	16 G9	17 G10	18 G11	19 G12	20 G13	21 G14
1	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	8 G1	-1 error
2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2
3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3
4	4 E	4 E	4 E	4 E	4 E	4 E	4 E	4 E	4 E	4 E
5	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error
6	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee
7	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee

## 2 Tabla de Símbolos: Estructura e implementación

Contiene la información de los identificadores, de los cuales se guardan los campos: lexema, tipo y desplazamiento. Para las funciones, además, se guardará el número de parámetros, su tipo, la forma de paso de parámetros y el tipo del valor de retorno.

La tabla de símbolos estará formada por dos matrices de tamaño dinámico; la primera contendrán los identificadores de ámbito global y la segunda del local. Así pues, esta segunda se creará al encontrar la declaración de una función y se borrará al acabar de ser declarada. También se utiliza un flag de declaración o uso ( $\text{FlagDeclUso}$ ), un flag para saber cual es la tabla actual y dos más para el valor del desplazamiento en cada una de las tablas.

Sin embargo, en la implementación actual sólo se usa una tabla y siempre se supone que está el  $\text{FlagDeclUso} = \text{Uso}$ , pero en el caso de que no este declarada la variable se insertará en la tabla actual, ya que requerimos de la implementación del Analizador

Semántico para poder saber cuando se cambia de ámbito y cuando se están declarando o usando identificadores. Así que, la acción semántica que genera los tokens de los identificadores quedaría temporalmente así:

```
$G_2$: if (lex $\in$ palRes)
    GEN_TOKEN(palRes, -)
else if ((p:= BUSCA_TS(lex))=NULL)
    p:=INSERTAR_TS(lex)
    GEN_TOKEN(ID, p)
```

### 3 Anexo de Pruebas

#### Error 1:

```
1 var int a;
2 var int b;
3 /*65 caracteres*/
4 cadena = '
    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
    ;
5 a = 3;
6 b = a;
7 var boolean c;
8 c = a < b;
9 if (c) {
10     b = 1;
11 }
12 c = b < a;
13 if (c) b = 4;
14 a = a + b;
15 print (a);
16 print (b);
```

```
> Error Lexico: Exceso de caracteres en la cadena. Linea: 4
```

#### Error 2:

```
1 var int a;
2 var int b;
3 a = 33333;
4 b = a;
5 if (a < b) b = 1;
6 if (b < a) b = 8;
7 a = a + b;
8 print (a);
9 print (b);
```

```
> Error Lexico: Numero fuera de rango. Linea: 3
```

#### Error 3:

```
1 var string texto; /*Comentario bueno*/
2 function imprime (string msg)
3 {
4     print (msg);
```

```

5 }
6 / Comentario malo*/
7 function pideTexto ()
8 {
9     print ( 'Introduce un texto' );
10    input (texto);
11 }
12 pideTexto();
13 var string textoAux;
14 textoAux = texto;
15 imprime (textoAux);

```

> Error Lexico: Transicion no prevista. Linea: 6

## Prueba 1 Correcta:

```

1 var int a;
2 var int b;
3 a = 3;
4 b = a;
5 var boolean c;
6 c = a < b;
7 if (c) b = 1;
8 c = b < a;
9 if (c) b = 4;
10 a = a + b;
11 print (a);
12 print (b);

```

### Tokens:

```

<DEC, >
<TipoVarENT, >
<ID, 0>
<PuntoComa, >
<DEC, >
<TipoVarENT, >
<ID, 1>
<PuntoComa, >
<ID, 0>
<ASIG, >
<ENT, 3>
<PuntoComa, >
<ID, 1>
<ASIG, >
<ID, 0>
<PuntoComa, >
<DEC, >
<TipoVarLOG, >
<ID, 2>
<PuntoComa, >
<ID, 2>
<ASIG, >
<ID, 0>

```

<MENOR, >  
 <ID, 1>  
 <PuntoComa, >  
 <IF, >  
 <ParentesisAbrir, >  
 <ID, 2>  
 <ParentesisCerrar, >  
 <ID, 1>  
 <ASIG, >  
 <ENT, 1>  
 <PuntoComa, >  
 <ID, 2>  
 <ASIG, >  
 <ID, 1>  
 <MENOR, >  
 <ID, 0>  
 <PuntoComa, >  
 <IF, >  
 <ParentesisAbrir, >  
 <ID, 2>  
 <ParentesisCerrar, >  
 <ID, 1>  
 <ASIG, >  
 <ENT, 4>  
 <PuntoComa, >  
 <ID, 0>  
 <ASIG, >  
 <ID, 0>  
 <SUMA, >  
 <ID, 1>  
 <PuntoComa, >  
 <Print, >  
 <ParentesisAbrir, >  
 <ID, 0>  
 <ParentesisCerrar, >  
 <PuntoComa, >  
 <Print, >  
 <ParentesisAbrir, >  
 <ID, 1>  
 <ParentesisCerrar, >  
 <PuntoComa, >

### Tabla de símbolos:

Tabla Simbolos #1:

\* LEXEMA: 'a'

\* LEXEMA: 'b'



\* LEXEMA: 'c'

## Prueba 2 Correcta:

```
1 var boolean booleano;
2 function boolean bisiestro (int a)
3 { var int bis;
4   print ('Es bisiestro?');
5   input(bis);
6   return (!(a + 4 < 0));
7 }
8 function int dias (int m, int a)
9 {
10  var int dd;
11  print ('di cuantos dias tiene el mes ');
12  print (m);
13  input(dd);
14  if (bisiestro(a)) dd = dd + 1;
15  return dd;
16 }
17 var boolean zzz;
```

### Tokens:

<DEC, >  
<TipoVarLOG, >  
<ID, 0>  
<PuntoComa, >  
<DECFunc, >  
<TipoVarLOG, >  
<ID, 1>  
<ParentesisAbrir, >  
<TipoVarENT, >  
<ID, 2>  
<ParentesisCerrar, >  
<CorcheteAbrir, >  
<DEC, >  
<TipoVarENT, >  
<ID, 3>  
<PuntoComa, >  
<Print, >  
<ParentesisAbrir, >  
<CAD, "Es bisiestro?">  
<ParentesisCerrar, >  
<PuntoComa, >  
<Input, >  
<ParentesisAbrir, >  
<ID, 3>  
<ParentesisCerrar, >  
<PuntoComa, >  
<Return, >

```

<ParentesisAbrir , >
<NOT, >
<ParentesisAbrir , >
<ID, 2>
<SUMA, >
<ENT, 4>
<MENOR, >
<ENT, 0>
<ParentesisCerrar , >
<ParentesisCerrar , >
<PuntoComa , >
<CorcheteCerrar , >
<DECFunc, >
<TipoVarENT, >
<ID, 4>
<ParentesisAbrir , >
<TipoVarENT, >
<ID, 5>
<Coma, >
<TipoVarENT, >
<ID, 2>
<ParentesisCerrar , >
<CorcheteAbrir , >
<DEC, >
<TipoVarENT, >
<ID, 6>
<PuntoComa , >
<Print , >
<ParentesisAbrir , >
<CAD, "di cuantos dias tiene el mes ">
<ParentesisCerrar , >
<PuntoComa , >
<Print , >
<ParentesisAbrir , >
<ID, 5>
<ParentesisCerrar , >
<PuntoComa , >
<Input , >
<ParentesisAbrir , >
<ID, 6>
<ParentesisCerrar , >
<PuntoComa , >
<IF, >
<ParentesisAbrir , >
<ID, 1>
<ParentesisAbrir , >
<ID, 2>
<ParentesisCerrar , >

```

```

<ParentesisCerrar , >
<ID , 6>
<ASIG , >
<ID , 6>
<SUMA , >
<ENT , 1>
<PuntoComa , >
<Return , >
<ID , 6>
<PuntoComa , >
<CorcheteCerrar , >
<DEC , >
<TipoVarLOG , >
<ID , 7>
<PuntoComa , >

```

### Tabla de símbolos:

Tabla Simbolos #1:

```

* LEXEMA: 'booleano '
* LEXEMA: 'bisiesto '
* LEXEMA: 'a '
* LEXEMA: 'bis '
* LEXEMA: 'dias '
* LEXEMA: 'm'
* LEXEMA: 'dd'
* LEXEMA: 'zzz '

```

### Prueba 3 Correcta:

```

1 var int a;
2 var int b;
3 var int c;
4 print ('Introduce el primer operando');
5 input (a);
6 print ('Introduce el segundo operando');
7 input (b);
8 function int suma (int num1, int num2)
9 {
10     var int res;
11     res = num1+num2;
12     return res;
13 }
14 c = suma (a, b);

```

15 `print (c);`

**Tokens:**

<DEC, >  
<TipoVarENT, >  
<ID, 0>  
<PuntoComa, >  
<DEC, >  
<TipoVarENT, >  
<ID, 1>  
<PuntoComa, >  
<DEC, >  
<TipoVarENT, >  
<ID, 2>  
<PuntoComa, >  
<Print, >  
<ParentesisAbrir, >  
<CAD, "Introduce el primer operando">  
<ParentesisCerrar, >  
<PuntoComa, >  
<Input, >  
<ParentesisAbrir, >  
<ID, 0>  
<ParentesisCerrar, >  
<PuntoComa, >  
<Print, >  
<ParentesisAbrir, >  
<CAD, "Introduce el segundo operando">  
<ParentesisCerrar, >  
<PuntoComa, >  
<Input, >  
<ParentesisAbrir, >  
<ID, 1>  
<ParentesisCerrar, >  
<PuntoComa, >  
<DECFunc, >  
<TipoVarENT, >  
<ID, 3>  
<ParentesisAbrir, >  
<TipoVarENT, >  
<ID, 4>  
<Coma, >  
<TipoVarENT, >  
<ID, 5>  
<ParentesisCerrar, >  
<CorcheteAbrir, >  
<DEC, >  
<TipoVarENT, >

```

<ID, 6>
<PuntoComa, >
<ID, 6>
<ASIG, >
<ID, 4>
<SUMA, >
<ID, 5>
<PuntoComa, >
<Return, >
<ID, 6>
<PuntoComa, >
<CorcheteCerrar, >
<ID, 2>
<ASIG, >
<ID, 3>
<ParentesisAbrir, >
<ID, 0>
<Coma, >
<ID, 1>
<ParentesisCerrar, >
<PuntoComa, >
<Print, >
<ParentesisAbrir, >
<ID, 2>
<ParentesisCerrar, >
<PuntoComa, >

```

### Tabla de símbolos:

Tabla Simbolos #1:

- \* LEXEMA: 'a'
- \* LEXEMA: 'b'
- \* LEXEMA: 'c'
- \* LEXEMA: 'suma'
- \* LEXEMA: 'num1'
- \* LEXEMA: 'num2'
- \* LEXEMA: 'res'