# analizador léxico

Procesadores de Lenguajes

MEMORIA DE LA PRÁCTICA

## GRUPO 20

Elías Dorado Rodríguez

A18i018

Roberto José Peris Acedo

A18i014

Álvaro Vega Motrel

A18i045

Contenido

[analizador léxico 0](https://upm365.sharepoint.com/sites/PdL-Grupo20/Documentos%20compartidos/Documentos/Memoria%20-%20Analizador%20léxico.docx#_Toc87182537)

[GRUPO 20 0](https://upm365.sharepoint.com/sites/PdL-Grupo20/Documentos%20compartidos/Documentos/Memoria%20-%20Analizador%20léxico.docx#_Toc87182538)

[Diseño Análisis Léxico 2](#_Toc87182539)

[Tokens 2](#_Toc87182540)

[Gramática 4](#_Toc87182541)

[Autómata 5](#_Toc87182542)

[Acciones semánticas 6](#_Toc87182543)

[Diseño Tabla de simbolos 9](#_Toc87182544)

[Anexo 9](#_Toc87182545)

[Pruebas correctas 9](#_Toc87182546)

[Caso de prueba 1 9](#_Toc87182547)

[Resultados 1 9](#_Toc87182548)

[Caso de prueba 2 1](#_Toc87182549)

[Resultados 2 2](#_Toc87182550)

[Caso de prueba 3 0](#_Toc87182551)

[Resultados 3 1](#_Toc87182552)

[Pruebas fallidas 1](#_Toc87182553)

[Caso de prueba 4 1](#_Toc87182554)

[Resultados 4 1](#_Toc87182555)

[Caso de prueba 5 1](#_Toc87182556)

[Resultados 5 1](#_Toc87182557)

[Caso de prueba 6 1](#_Toc87182558)

[Resultados 6 2](#_Toc87182559)

# Diseño Análisis Léxico

La estructura general del programa que vamos a crear esta compuesta por varios elementos como pueden ser: funciones, declaraciones y sentencias. Sentencias formadas por:

Variables: Tipos enteros, lógicos y cadenas. Cadenas con “”. Declaración de variables globales, locales y funciones. Declaración explícita de variables enteras en asignación o en “input”.

Operadores: Operadores aritméticos (Suma y Resta), Operadores lógicos (Igualdad (==), Desigualdad (!=) y And (&&)) y Operadores de asignación (Igual (=) y O Logico (|=))

Sentencias: Asignación, Condicionales simples (if), llamadas a funciones con retorno y Sentencia repetitiva (for).

Operaciones de Entrada/Salida: input

Comentarios (//)

# Tokens

1. <ENTERO, valor>

2. <CADENA, lexema>

3. <FUNCTION, ->

4. <BOOLEAN, ->

5. <TRUE, ->

6. <FALSE, ->

7. <PARENTA, ->

8. <PARENTC, ->

9. <SUMA, ->

10. <RESTA, ->

11. <IGUIGU, ->

12. <DESIGU, ->

13. <AND, ->

14. <IGUAL, ->

15. <OLOG, ->

16. <ID,posid>

17. <LET, ->

18. <INT, ->

19. <STRING, ->

20. <PRINT, ->

21. <INPUT, ->

22. <PYC, ->

23. <COMA, ->

24. <RETURN, ->

25. <IF, ->

26. <FOR, ->

27. <CORCHA, ->

28. <CORCHC, ->

29. <EOF, ->

# Gramática

Leyenda de la gramática:

del = delimitador

l = letras

d = dígitos

c = cualquier carácter - {del}

\n=salto de linea

t=todo carácter – {\n}

S(0) -> (11) { | (12) } | (13) EOF | (14) , | (15) ; | (16) + | (17) - | (18) ( | (19) ) | l A | d B | = C | ! D | & E | / F | " G | '|' H | del S

A(1) -> l A | d I | \_ I |λ

B(2) -> d B | λ

C(3) -> = | λ

D(4) -> =

E(5) -> &

F(6) -> / J

G(7) -> c G |"

H(8) -> =

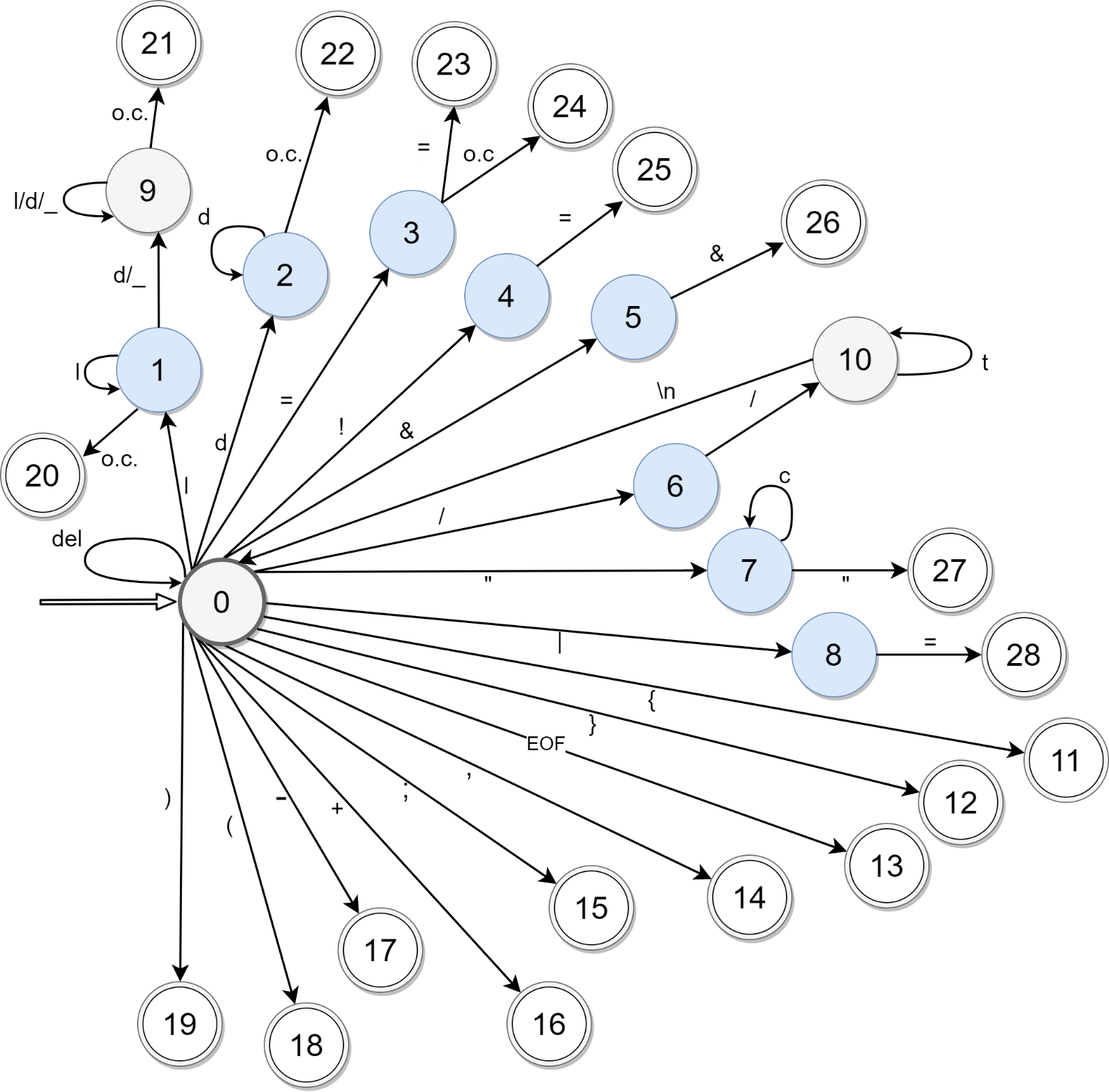
I(9) -> d I| l I | λ

J(10) -> t J | \n S

# Autómata

Leyenda (adicional a la de la gramática):

EOF = End Of File

o.c. = Otro carácter

# Acciones semánticas

0-0: LEER

0-1: LEER; Lexema(l)

1-1: LEER; CONCAT(lexema) (lexema = lexema + l)

1-9: LEER; CONCAT(lexema) (lexema = lexema + d)

1-20: if(esPalabraReservada(lexema) = true){

GENTOKEN(lexema.toUpperCase(),-)

}

else{

GENTOKEN(ID,posID)

};LEER

9-9: LEER; CONCAT(lexema) (lexema = lexema + l/d)

9-21: if(zona declaracion = true){

if(buscarTS(lexema) = false){

añadirTS(lexema,posID)

GENTOKEN(ID,posID)

}

else{

ERROR("Variable ya existe")

}

else{

GENTOKEN(ID,posID)

};LEER

0-2: LEER; valor(d)

2-2: valor = 10 \* valor + d; LEER

2-22: if(valor < 32767){

GENTOKEN(ENTERO,valor)

}

else{

ERROR("Numero muy grande")

}

0-3: LEER

3-23: GENTOKEN(IGUIGU,-); LEER

3-24: GENTOKEN(IGUAL,-)

0-4: LEER

4-25: GENTOKEN(DESIGU,-);LEER

0-5: LEER

5-26: GENTOKEN(AND,-); LEER

0-6: LEER

6-10: LEER

10-10: LEER

10-0: LEER

0-7: LEER; lexema(l)

Cont=1;

7-7: CONCAT(lexema) (lexema=lexema + c); LEER

Cont++;

7-27: if(Cont < 64){

GENTOKEN(CADENA,lexema)

}

else{

ERROR("Lexema demasiado grande)

}; LEER

0-8: LEER

8-28: GENTOKEN(OLOG,-)

0-11: GENTOKEN(CORCHA,-)

0-12: GENTOKEN(CORCHC,-)

0-13: GENTOKEN(EOF,-)

0-14: GENTOKEN(COMA,-)

0-15: GENTOKEN(PYC,-)

0-16: GENTOKEN(SUMA,-)

0-17: GENTOKEN(RESTA,-)

0-18: GENTOKEN(PARENTA,-)

0-19: GENTOKEN(PARENTC,-)

# Diseño Tabla de simbolos

Para la tabla de símbolos hemos creado una clase distinta que será la propia tabla de símbolos y se usará para definir las entradas de la tabla de símbolos. Cada tabla de símbolos está formada por un Hash map que será donde almacenemos todas las entradas de la tabla. Un nombre de tabla. Una posición para cada entrada. El numero de la tabla para saber que tabla es la que estamos tratando.

Por el momento la clase tiene 4 funciones. Una función para insertar en la tabla, otra para buscar si está el id en la tabla, una función para obtener su posición en la tabla y por último una función para imprimir la tabla de símbolos en su formato

# Anexo

## Pruebas correctas

### Caso de prueba 1

let int a;

let int b;

let int int;

print ("Introduce el primer operando");

input (a);

print ("Introduce el segundo operando");

input (b);

function int operacion (int num1, int num2)

{

return num1-num2;

}

int = 0;

print (operacion (a, b));

### Resultados 1

Tokens:

<PARENTC, >

<PYC, >

<INPUT, >

<PARENTA, >

<ID , 1 >

<PARENTC, >

<PYC, >

<FUNCTION, >

<INT, >

<ID , 2 >

<PARENTA, >

<INT, >

<ID , 3 >

<COMA, >

<INT, >

<ID , 4 >

<PARENTC, >

<CORCHA, >

<RETURN, >

<ID , 3 >

<RESTA, >

<ID , 4 >

<PYC, >

<CORCHC, >

<INT, >

<IGUAL, >

<ENTERO , 0 >

<PYC, >

<PRINT, >

<PARENTA, >

<ID , 2 >

<PARENTA, >

<ID , 0 >

<COMA, >

<ID , 1 >

<PARENTC, >

<PARENTC, >

<PYC, >

<EOF, >

Tabla de Simbolos Principal #1:

\* LEXEMA : 'a'

\* LEXEMA : 'b'

\* LEXEMA : 'operacion'

\* LEXEMA : 'num1'

\* LEXEMA : 'num2'

### Caso de prueba 2

/// prueba correcta

let int entero;

let string cadena;

let boolean logico;

let int abcdefghij\_\_\_\_\_\_\_\_;

function FuncionNumero ()

{

let boolean i;

i = logico;

logico=i&&logico;

i = logico;

}

function FuncionRetorno string (string otracadena, int x, boolean z)

{

x = 33 + (x - entero); return otracadena;

}

function FuncionSentencia (int b, boolean z)

{

for (b=1 ; z; )

{

print (88);

}

}

function Funcion (int x\_\_, boolean b)

{

if (x\_\_ == 0)FuncionSentencia(x\_\_,b);

print

(cadena);return;

}

let int iii;

let boolean bbb;

// esto

// es

// un

// comentario !

//// multi \*\*\*/

//línea

input(iii);

print ( iii ) ;

iii = iii;

bbb = bbb;

Funcion

( iii,

bbb )

;

### Resultados 2

Tokens:

<LET, >

<INT, >

<ID , 0 >

<PYC, >

<LET, >

<STRING, >

<ID , 1 >

<PYC, >

<LET, >

<BOOLEAN, >

<ID , 2 >

<PYC, >

<LET, >

<INT, >

<ID , 3 >

<PYC, >

<FUNCTION, >

<ID , 4 >

<PARENTA, >

<PARENTC, >

<CORCHA, >

<LET, >

<BOOLEAN, >

<ID , 5 >

<PYC, >

<ID , 5 >

<IGUAL, >

<ID , 2 >

<PYC, >

<ID , 2 >

<IGUAL, >

<ID , 5 >

<AND, >

<ID , 2 >

<PYC, >

<ID , 5 >

<IGUAL, >

<ID , 2 >

<PYC, >

<CORCHC, >

<FUNCTION, >

<ID , 6 >

<STRING, >

<PARENTA, >

<STRING, >

<ID , 7 >

<COMA, >

<INT, >

<ID , 8 >

<COMA, >

<BOOLEAN, >

<ID , 9 >

<PARENTC, >

<CORCHA, >

<ID , 8 >

<IGUAL, >

<ENTERO , 33 >

<SUMA, >

<PARENTA, >

<ID , 8 >

<RESTA, >

<ID , 0 >

<PARENTC, >

<PYC, >

<RETURN, >

<ID , 7 >

<PYC, >

<CORCHC, >

<FUNCTION, >

<ID , 10 >

<PARENTA, >

<INT, >

<ID , 11 >

<COMA, >

<BOOLEAN, >

<ID , 9 >

<PARENTC, >

<CORCHA, >

<FOR, >

<PARENTA, >

<ID , 11 >

<IGUAL, >

<ENTERO , 1 >

<PYC, >

<ID , 9 >

<PYC, >

<PARENTC, >

<CORCHA, >

<PRINT, >

<PARENTA, >

<ENTERO , 88 >

<PARENTC, >

<PYC, >

<CORCHC, >

<CORCHC, >

<FUNCTION, >

<ID , 12 >

<PARENTA, >

<INT, >

<ID , 13 >

<COMA, >

<BOOLEAN, >

<ID , 11 >

<PARENTC, >

<CORCHA, >

<IF, >

<PARENTA, >

<ID , 13 >

<IGUIGU, >

<ENTERO , 0 >

<PARENTC, >

<ID , 10 >

<PARENTA, >

<ID , 13 >

<COMA, >

<ID , 11 >

<PARENTC, >

<PYC, >

<PRINT, >

<PARENTA, >

<ID , 1 >

<PARENTC, >

<PYC, >

<RETURN, >

<PYC, >

<CORCHC, >

<LET, >

<INT, >

<ID , 14 >

<PYC, >

<LET, >

<BOOLEAN, >

<ID , 15 >

<PYC, >

<INPUT, >

<PARENTA, >

<ID , 14 >

<PARENTC, >

<PYC, >

<PRINT, >

<PARENTA, >

<ID , 14 >

<PARENTC, >

<PYC, >

<ID , 14 >

<IGUAL, >

<ID , 14 >

<PYC, >

<ID , 15 >

<IGUAL, >

<ID , 15 >

<PYC, >

<ID , 12 >

<PARENTA, >

<ID , 14 >

<COMA, >

<ID , 15 >

<PARENTC, >

<PYC, >

<EOF, >

Tabla de Simbolos Principal #1:

\* LEXEMA : 'entero'

\* LEXEMA : 'cadena'

\* LEXEMA : 'logico'

\* LEXEMA : 'abcdefghij\_\_\_\_\_\_\_\_'

\* LEXEMA : 'FuncionNumero'

\* LEXEMA : 'i'

\* LEXEMA : 'FuncionRetorno'

\* LEXEMA : 'otracadena'

\* LEXEMA : 'x'

\* LEXEMA : 'z'

\* LEXEMA : 'FuncionSentencia'

\* LEXEMA : 'b'

\* LEXEMA : 'Funcion'

\* LEXEMA : 'x\_\_'

\* LEXEMA : 'iii'

\* LEXEMA : 'bbb'

### Caso de prueba 3

let int a;

let int b;

let boolean bbb;

a = 3;

b=a

;

input(z);

print(z);

x=z;

print (z-1);

b=b&&b;a = a - b;

print (a) ;

print(b);

### Resultados 3

Tokens:

<LET, >

<INT, >

<ID , 0 >

<PYC, >

<LET, >

<INT, >

<ID , 1 >

<PYC, >

<LET, >

<BOOLEAN, >

<ID , 2 >

<PYC, >

<ID , 0 >

<IGUAL, >

<ENTERO , 3 >

<PYC, >

<ID , 1 >

<IGUAL, >

<ID , 0 >

<PYC, >

<INPUT, >

<PARENTA, >

<ID , 3 >

<PARENTC, >

<PYC, >

<PRINT, >

<PARENTA, >

<ID , 3 >

<PARENTC, >

<PYC, >

<ID , 4 >

<IGUAL, >

<ID , 3 >

<PYC, >

<PRINT, >

<PARENTA, >

<ID , 3 >

<RESTA, >

<ENTERO , 1 >

<PARENTC, >

<PYC, >

<ID , 1 >

<IGUAL, >

<ID , 1 >

<AND, >

<ID , 1 >

<PYC, >

<ID , 0 >

<IGUAL, >

<ID , 0 >

<RESTA, >

<ID , 1 >

<PYC, >

<PRINT, >

<PARENTA, >

<ID , 0 >

<PARENTC, >

<PYC, >

<PRINT, >

<PARENTA, >

<ID , 1 >

<PARENTC, >

<PYC, >

<EOF, >

Tabla de Simbolos Principal #1:

\* LEXEMA : 'a'

\* LEXEMA : 'b'

\* LEXEMA : 'bbb'

\* LEXEMA : 'z'

\* LEXEMA : 'x'

## Pruebas fallidas

### Caso de prueba 4

let int x;

let int z;

let boolean b;

input (z);

print (z);

x==z;

print (z/1);

b=b||b;if (b) z =

x / 6

+ z

/ (1

- 2

- y)

/ 8;

### Resultados 4

ERROR: el caracter 1 no es valido. Los comentarios son con 2 barras: //. En la linea 7

ERROR: el caracter | no es un O-lógico. Carácter no reconocido. En la linea 8

ERROR: el caracter b no es un O-lógico. Carácter no reconocido. En la linea 8

ERROR: el caracter no es valido. Los comentarios son con 2 barras: //. En la linea 9

ERROR: el caracter no es valido. Los comentarios son con 2 barras: //. En la linea 11

ERROR: el caracter no es valido. Los comentarios son con 2 barras: //. En la linea 14

### Caso de prueba 5

let int a ;

let int b ;

let int number;

print ( "Introduce el primer operando" );

input (a);

print ("Introduce el segundo operando");input(b);

function operacion int(int num2,int num1)

{

let int res;

res=num1\*num2;

return ((res));

}

number = 0;

print(operacion(b,a));

### Resultados 5

ERROR: el caracter \* no es valido. En la linea 10

### Caso de prueba 6

let int a ;

let int b ;

let int number;

print ( "Introduce el primer operando" );

input (a);

print ("Introduce el segundo operando");input(b);

function operacion int(int num2,int num1)

{

let int res;

res=num1-num2;

return ((res));

}

number = 1000000;

print(operacion(b,a));

### Resultados 6

ERROR: el valor es superior al permitido. En la linea 13