
Traductores de Lenguajes

MEMORIA FINAL

Grupo 55

Daniel Tomás Sánchez
Aarón Cabero Blanco
Alejandro Cuadrón Lafuente

Curso 2020/2021

Índice

1	Introducción	2
2	Diseño Analizador Léxico	3
2.1	Tokens	3
2.2	Gramática Regular	4
2.3	Autómata Finito Determinista	4
2.4	Acciones Semánticas	5
2.5	Errores	6
3	Diseño Analizador Sintáctico	7
3.1	Gramática	7
3.2	Tabla LR(1)	8
4	Diseño Analizador Semántico	11
5	Diseño Tabla de Símbolos	12
6	Anexo	13
6.1	Casos de prueba correctos	13
6.1.1	Prueba correcta 1	13
6.1.2	Prueba correcta 2	16
6.1.3	Prueba correcta 3	18
6.1.4	Prueba correcta 4	20
6.1.5	Prueba correcta 5	23
6.2	Casos de prueba erróneos	27
6.2.1	Prueba errónea 1	27
6.2.2	Prueba errónea 2	27
6.2.3	Prueba errónea 3	27
6.2.4	Prueba errónea 4	27
6.2.5	Prueba errónea 5	28

1 | Introducción

Hemos decidido usar, como lenguaje de programación, Python, ya que nos apetecía aprender un lenguaje nuevo, además de que es muy usado en la industria.

El trabajo completo, tanto el léxico, como el sintáctico y el semántico, ha sido realizado con la herramienta o librería externa "SLY"[**SLY**].

Opciones de grupo:

- Sentencias: Sentencia repetitiva (**for**)
- Operadores especiales: Post-auto-decremento (**-- como sufijo**)
- Técnicas de Análisis Sintáctico: **Ascendente**
- Comentarios: Comentario de bloque (**/* */**)
- Cadenas: Con comillas dobles (**" "**)

2 | Diseño Analizador Léxico

2.1 Tokens

▪ Identificador	<ID, punteroTS>
▪ Constante entera	<CTEENTERA, valor>
▪ Cadena de caracteres	<CADENA, lexema>
▪ false	<CTELOGICA, 0>
▪ true	<CTELOGICA, 1>
▪ Palabra reservada Number	<NUMBER, ->
▪ Palabra reservada String	<STRING, ->
▪ Palabra reservada Boolean	<BOOLEAN, ->
▪ Palabra reservada Let	<LET, ->
▪ Palabra reservada Alert	<ALERT, ->
▪ Palabra reservada Input	<INPUT, ->
▪ Palabra reservada Function	<FUNCTION, ->
▪ Palabra reservada Return	<RETURN, ->
▪ Palabra reservada If	<IF, ->
▪ Palabra reservada For	<FOR, ->
▪ --	<OPESP, ->
▪ -	<OPARIT, ->
▪ =	<OPASIG, ->
▪ ==	<OPREL, ->
▪ &&	<OPLOG, ->
▪ (<ABPAREN, ->
▪)	<CEAPAREN, ->
▪ {	<ABLLAVE, ->
▪ }	<CELLAVE, ->
▪ ,	<COMA, ->
▪ ;	<PUNTOYCOMA, ->

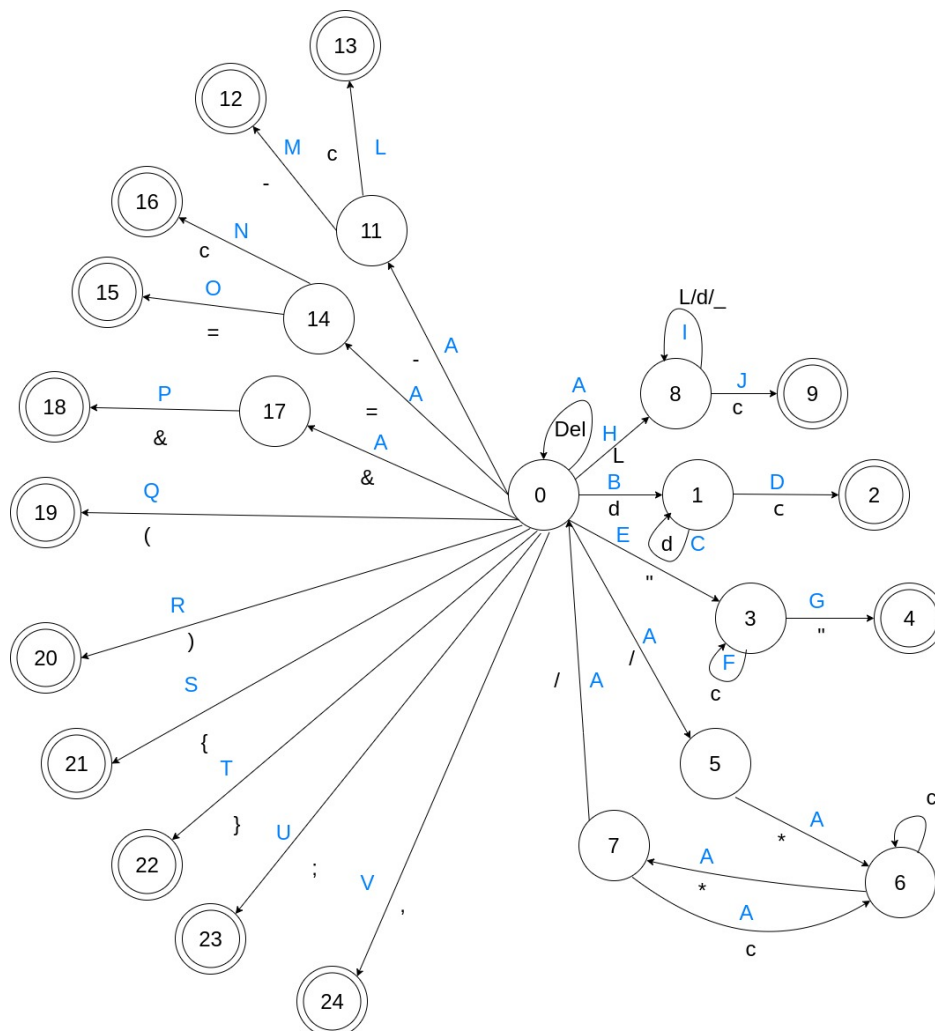
2.2 Gramática Regular

Axioma = A

$$A \rightarrow \text{del } A \mid d D \mid " S \mid / C \mid l I \mid -M \mid = E \mid \& N \mid (\mid) \mid \{ \mid \} \mid ; \mid ,$$
$$D \rightarrow d \quad D \mid \lambda$$
$$S \rightarrow " \mid c \mid S$$
$$C \rightarrow {}^* C',$$
$$C' \rightarrow * C'' \mid c C'$$
$$C'' \rightarrow / A \mid c C'$$
$$I \rightarrow dI \mid lI \mid \underline{} I \mid \lambda$$
$$M \rightarrow - \frac{1}{\lambda}$$
$$E \rightarrow \mathbb{R}^n$$
$$N \rightarrow \&$$

Siendo d un dígito, l una letra, c cualquier otro carácter y del un delimitador.

2.3 Autómata Finito Determinista



2.4 Acciones Semánticas

```
A: leer
B: number = int(d), leer
C: number = number * 10 + int(d), leer
D: if number > 32767
    pError("Número fuera de rango")
    else
        genToken(CTEENTERA, number);
E: string = "", contador = 0, leer
F: string = string + otroCS, contador++, leer
G: if contador > 64
    pError("Cadena demasiado larga")
    else
        genToken(CADENA, string)
    leer
H: string = 1, leer
I: string = string + 1/D/_ , leer
J: if palabrasReservadas.contains(string)
    if string == "number"
        genToken(NUMBER, -)
    elif string == "string"
        genToken(STRING, -)
    elif string == "boolean"
        genToken(BOOLEAN, -)
    elif string == "let"
        genToken(LET, -)
    elif string == "alert"
        genToken(ALERT, -)
    elif string == "input"
        genToken(INPUT, -)
    elif string == "return"
        genToken(RETURN, -)
    elif string == "if"
        genToken(IF, -)
    else
        genToken(FOR, -)
```

```

else    // palabrasReservadas.contains(string) = False
    puntero = TS.get(string)
    if zona_decl == True
        if puntero != None
            pError("Identificador ya declarado")
        else
            TS.update(string)
            puntero = TS.get(string)
            genToken(ID, puntero)
    else
        if puntero == None
            TS.update(string)
            puntero = TS.get(string)
            genToken(ID, puntero)
        else
            genToken(ID, puntero)

L: genToken(OPARIT, -)
M: genToken(OPESP, -), leer
N: genToken(OPASIG, -)
O: genToken(OPREL, -), leer
P: genToken(OPLOG, -), leer
Q: genToken(ABPAREN, -), leer
R: genToken(CEPAREN, -), leer
S: genToken(ABLLAVE, -), leer
T: genToken(CELLAVE, -), leer
U: genToken(COMA, -), leer
V: genToken(PUNTOYCOMA, -), leer
W: genToken(EOF, -), leer

```

2.5 Errores

Error léxico (siempre se lanza cuando el analizador léxico encuentra un error).

1. Cadena con longitud mayor de 64 caracteres.
2. Número fuera de rango (mayor de 32767).
3. Identificador ya declarado.
4. Carácter ilegal.

Todo error va acompañado de la *línea* y *columna* en el que se ha encontrado dicho error.

3 | Diseño Analizador Sintáctico

3.1 Gramática

Axioma = B

No Terminales = { A B C D E F G H I J K L M N O P Q R S T U V W F1 F2 F3 }

Terminales = { && == - -- () = , ; id ent cad log let alert input return for if number
boolean string function }

Producciones = {

B → D

D → F D

D → G D

D → λ

G → if (E) S

G → S

S → H ;

H → id (I)

I → E J

I → λ

J → , E J

J → λ

S → K ;

K → id = E

S → alert (E) ;

S → input (id) ;

S → return L ;

L → E

L → λ

G → let M T id ;

M → λ

T → number

T → boolean

T → string

G → for (N ; E ; O) C

N → K

N → λ

O → K

O → -- id

O → λ

C → G C

C → λ

F → F1 F2 F3

F1 → function P Q id

P → λ

Q → T

Q → λ

F2 → (A)

A → T id AA

A → λ

AA → , T id AA

AA → λ

F3 → C

E → E && R

E → R

R → R == U

R → U

U → U - V

U → V

V → -- id

V → id

V → (E)

V → H

V → ent

V → cad

V → log

}

3.2 Tabla LR(1)

[illegible]

9

[illegible]

4 | Diseño Analizador Semántico

5 | Diseño Tabla de Símbolos

La TS se compone de una lista de tablas, una de ellas es global y se crea al empezar, mientras que el resto son locales y se van creando a medida que avanza la compilación. Están ordenadas en orden de creación. Cada tabla tiene un flag que indica si la tabla existe (esta activa) o se ha eliminado, pero realmente no las eliminamos para posteriormente imprimirlas.

A su vez cada tabla contiene una lista de diccionarios, cada diccionario simboliza una entrada en la tabla de símbolos. Un diccionario es un hashmap que tiene como claves la palabra "lexema" y los tipos de atributos que le corresponda (por ejemplo "Tipo" ó "Despl"), como valores tiene el valor del lexema y de sus atributos en si mismos (por ejemplo "main" ó "entero").

6 | Anexo

6.1 Casos de prueba correctos

6.1.1 Prueba correcta 1

Programa introducido:

```
1 let string texto;  
2 let string textoAux;  
3 textoAux = texto;  
4 alert  
5     (textoAux);
```

Tokens:

<FUNCTION , >
<NUMBER , >
<ID , 0>
<ABPAREN , >
<NUMBER , >
<ID , 0>
<CEPAREN , >
<ABLLAVE , >
<IF , >
<ABPAREN , >
<ID , 0>
<OPREL , >
<CTEENTERA , 0>
<CEPAREN , >
<RETURN , >
<CTEENTERA , 1>
<PUNTOYCOMA , >
<RETURN , >
<ID , 0>
<OPARIT , >
<ID , 0>
<ABPAREN , >
<ID , 0>
<OPARIT , >
<CTEENTERA , 1>
<CEPAREN , >
<PUNTOYCOMA , >
<CELLAVE , >

Tabla de símbolos:

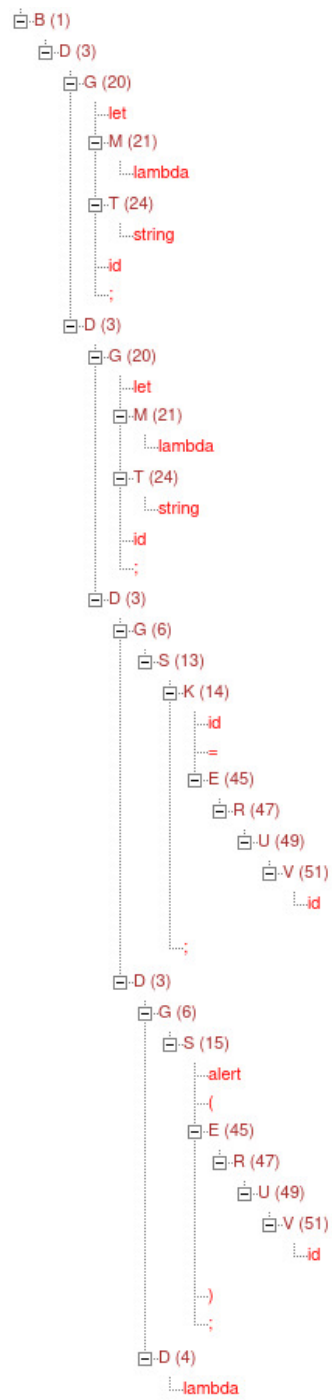
CONTENIDO DE LA TABLA 0 :

* LEXEMA : 'texto'
ATRIBUTOS :
+ Tipo : 'cadena'
+ Despl : '0'

* LEXEMA : 'textoAux'
ATRIBUTOS :

+ Tipo : 'cadena'
+ Despl : '64'

VAST:



6.1.2 Prueba correcta 2

Programa introducido:

```
1 function number Factorial (number n)
2 {
3     if (n == 0) return 1;
4     return n - Factorial (n - 1);
5 }
```

Tokens:

<FUNCTION , >
<NUMBER , >
<ID , 0>
<ABPAREN , >
<NUMBER , >
<ID , 0>
<CEPAREN , >
<ABLLAVE , >
<IF , >
<ABPAREN , >
<ID , 0>
<OPREL , >
<CTEENTERA , 0>
<CEPAREN , >
<RETURN , >
<CTEENTERA , 1>
<PUNTOYCOMA , >
<RETURN , >
<ID , 0>
<OPARIT , >
<ID , 0>
<ABPAREN , >
<ID , 0>
<OPARIT , >
<CTEENTERA , 1>
<CEPAREN , >
<PUNTOYCOMA , >
<CELLAVE , >

Tabla de símbolos:

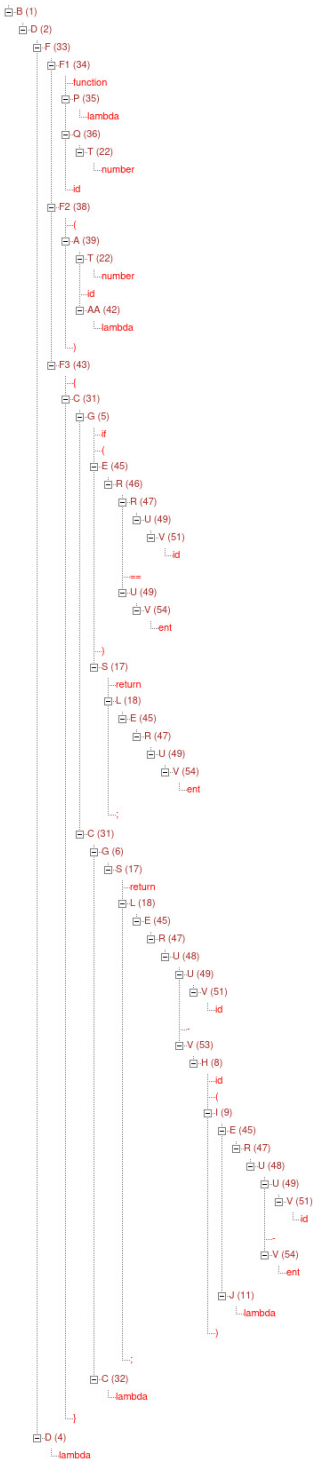
CONTENIDO DE LA TABLA 0 :

* LEXEMA : 'Factorial'
ATRIBUTOS :
+ Tipo : 'funcion'
+ TipoRetorno : 'entero'
+ EtiqFuncion : 'Et_Fun_0'
+ numParam : '1'
+ TipoParam1 : 'entero'

CONTENIDO DE LA TABLA 1 :

- * LEXEMA : 'n'
 - ATRIBUTOS :
 - + Tipo : 'entero'
 - + Despl : '0'
-

VAST:



6.1.3 Prueba correcta 3

Programa introducido:

```
1 let boolean z;  
2 for (b=1;z; )  
3 {  
4 alert (88);  
5 }
```

Tokens:

<LET , >
<BOOLEAN , >
<ID , 0>
<PUNTOYCOMA , >
<FOR , >
<ABPAREN , >
<ID , 1>
<OPASIG , >
<CTEENTERA , 1>
<PUNTOYCOMA , >
<ID , 0>
<PUNTOYCOMA , >
<CEPAREN , >
<ABLLAVE , >
<ALERT , >
<ABPAREN , >
<CTEENTERA , 88>
<CEPAREN , >
<PUNTOYCOMA , >
<CELLAVE , >

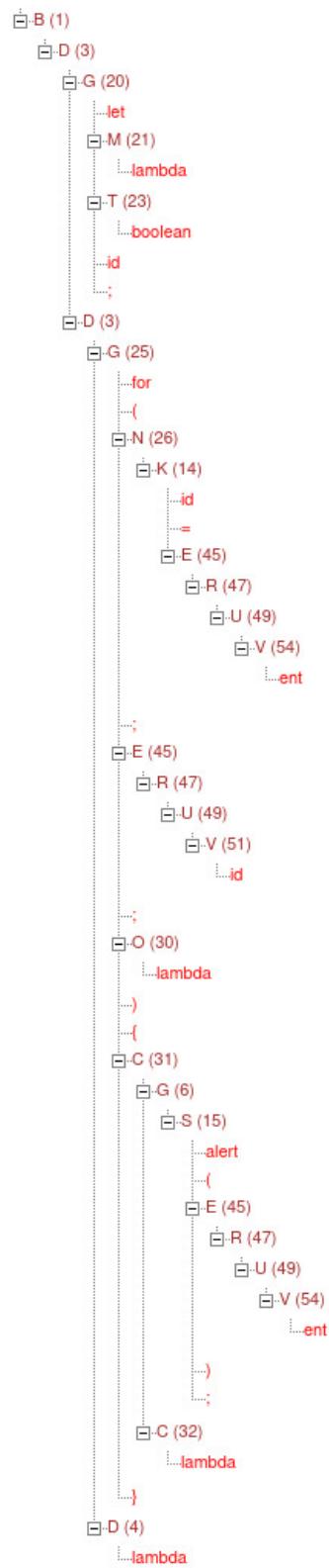
Tabla de símbolos:

CONTENIDO DE LA TABLA 0 :

* LEXEMA : 'z'
ATRIBUTOS :
+ Tipo : 'logico'
+ Despl : '0'

* LEXEMA : 'b'
ATRIBUTOS :
+ Tipo : 'entero'
+ Despl : '1'

VAST:



6.1.4 Prueba correcta 4

Programa introducido:

```
1  /* prueba
2      correcta */
3  let boolean b; let number x;
4  input (z);
5  alert (z);
6  x=z;
7  alert (z-1);
8  b=b&&b;if (b)
9  x =
10     x - 6
11     - z
12     - 1
13     - (2
14     - y
15     - 6);
```

Tokens:

```
<LET , >
<BOOLEAN , >
<ID , 0>
<PUNTOYCOMA , >
<LET , >
<NUMBER , >
<ID , 1>
<PUNTOYCOMA , >
<INPUT , >
<ABPAREN , >
<ID , 2>
<CEPAREN , >
<PUNTOYCOMA , >
<ALERT , >
<ABPAREN , >
<ID , 2>
<CEPAREN , >
<PUNTOYCOMA , >
<ID , 1>
<OPASIG , >
<ID , 2>
<PUNTOYCOMA , >
<ALERT , >
<ABPAREN , >
<ID , 2>
<OPARIT , >
<CTEENTERA , 1>
<CEPAREN , >
<PUNTOYCOMA , >
<ID , 0>
<OPASIG , >
<ID , 0>
<OPLOG , >
```

```

<ID , 0>
<PUNTOYCOMA , >
<IF , >
<ABPAREN , >
<ID , 0>
<CEPAREN , >
<ID , 1>
<OPASIG , >
<ID , 1>
<OPARIT , >
<CTEENTERA , 6>
<OPARIT , >
<ID , 2>
<OPARIT , >
<CTEENTERA , 1>
<OPARIT , >
<ABPAREN , >
<CTEENTERA , 2>
<OPARIT , >
<ID , 3>
<OPARIT , >
<CTEENTERA , 6>
<CEPAREN , >
<PUNTOYCOMA , >

```

Tabla de símbolos:

CONTENIDO DE LA TABLA 0 :

```

*      LEXEMA : 'b'
      ATRIBUTOS :
      + Tipo : 'logico'
      + Despl : '0'
-----
*      LEXEMA : 'x'
      ATRIBUTOS :
      + Tipo : 'entero'
      + Despl : '1'
-----
*      LEXEMA : 'z'
      ATRIBUTOS :
      + Tipo : 'entero'
      + Despl : '2'
-----
*      LEXEMA : 'y'
      ATRIBUTOS :
      + Tipo : 'entero'
      + Despl : '3'
-----

```

VASt:

6.1.5 Prueba correcta 5

Programa introducido:

```
1 function FuncionSentencia (number b, boolean z)
2 {
3     for (b=0;true; --b )
4     {
5         alert ("hola");
6     }
7 }
8 function Funcion (number x, boolean b)
9 {
10     if ( x == 0 ) FuncionSentencia(x,b);
11     alert
12         (cadena); return;
13 }
```

Tokens:

```
<FUNCTION , >
<ID , 0>
<ABPAREN , >
<NUMBER , >
<ID , 0>
<COMA , >
<BOOLEAN , >
<ID , 1>
<CEPAREN , >
<ABLLAVE , >
<FOR , >
<ABPAREN , >
<ID , 0>
<OPASIG , >
<CTEENTERA , 0>
<PUNTOYCOMA , >
<CTELOGICA , 1>
<PUNTOYCOMA , >
<OPESP , >
<ID , 0>
<CEPAREN , >
<ABLLAVE , >
<ALERT , >
<ABPAREN , >
<CADENA , "hola">
<CEPAREN , >
<PUNTOYCOMA , >
<CELLAVE , >
<CELLAVE , >
<FUNCTION , >
<ID , 1>
<ABPAREN , >
<NUMBER , >
<ID , 0>
<COMA , >
<BOOLEAN , >
```



```

<ID , 1>
<CEPAREN , >
<ABLLAVE , >
<IF , >
<ABPAREN , >
<ID , 0>
<OPREL , >
<CTEENTERA , 0>
<CEPAREN , >
<ID , 0>
<ABPAREN , >
<ID , 0>
<COMA , >
<ID , 1>
<CEPAREN , >
<PUNTOYCOMA , >
<ALERT , >
<ABPAREN , >
<ID , 2>
<CEPAREN , >
<PUNTOYCOMA , >
<RETURN , >
<PUNTOYCOMA , >
<CELLAVE , >

```

Tabla de símbolos:

CONTENIDO DE LA TABLA 0 :

```

*      LEXEMA : 'FuncionSentencia'
      ATRIBUTOS :
      + Tipo : 'funcion'
      + TipoRetorno : 'void'
      + EtiqFuncion : 'Et_Fun_0'
      + numParam : '2'
      + TipoParam1 : 'entero'
      + TipoParam2 : 'logico'

```

```

-----
*      LEXEMA : 'Funcion'
      ATRIBUTOS :
      + Tipo : 'funcion'
      + TipoRetorno : 'void'
      + EtiqFuncion : 'Et_Fun_1'
      + numParam : '2'
      + TipoParam1 : 'entero'
      + TipoParam2 : 'logico'

```

```

-----
*      LEXEMA : 'cadena'
      ATRIBUTOS :
      + Tipo : 'entero'
      + Despl : '0'
-----

```

CONTENIDO DE LA TABLA 1 :

* LEXEMA : 'b'
 ATRIBUTOS :
 + Tipo : 'entero'
 + Despl : '0'

* LEXEMA : 'z'
 ATRIBUTOS :
 + Tipo : 'logico'
 + Despl : '1'

CONTENIDO DE LA TABLA 2 :

* LEXEMA : 'x'
 ATRIBUTOS :
 + Tipo : 'entero'
 + Despl : '0'

* LEXEMA : 'b'
 ATRIBUTOS :
 + Tipo : 'logico'
 + Despl : '1'

VASt:

6.2 Casos de prueba erróneos

6.2.1 Prueba errónea 1

Programa introducido:

```
1 let number id;  
2 id(2);
```

Mensaje de error:

Error en la línea 2:

La variable no se puede invocar como una función, con argumentos

6.2.2 Prueba errónea 2

Programa introducido:

```
1 let boolean id;  
2 res = --id ;
```

Mensaje de error:

Error en la línea 2:

El operador especial '--' solo trabaja con tipos de datos enteros

6.2.3 Prueba errónea 3

Programa introducido:

```
1 function Funcion (number a, boolean b, string c){}  
2 Funcion(a,a,a);
```

Mensaje de error:

Error en la línea 2:

El tipo de los parámetros no es el esperado, se esperaban "'entero', 'logico', 'cadena'"

6.2.4 Prueba errónea 4

Programa introducido:

```
1 let string texto;  
2 function pideTexto ()  
3 {  
4   alert ("Introduce un texto");  
5   input (texto);  
6 }  
7  
8   return;
```

Mensaje de error:

Error en la línea 8:

No puede haber una sentencia return fuera de una función

6.2.5 Prueba errónea 5

Programa introducido:

```
1 let boolean bool;  
2 alert(bool);
```

Mensaje de error:

Error en la línea 2:

La expresión introducida no es una cadena o un entero