

---

# Traductores de Lenguajes

## MEMORIA FINAL

---

Grupo 55

Daniel Tomás Sánchez  
Aarón Cabero Blanco  
Alejandro Cuadrón Lafuente

Curso 2020/2021

# Índice

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Diseño Analizador Léxico</b>	<b>3</b>
2.1	Tokens . . . . .	3
2.2	Gramática Regular . . . . .	4
2.3	Autómata Finito Determinista . . . . .	4
2.4	Acciones Semánticas . . . . .	5
2.5	Errores . . . . .	6
<b>3</b>	<b>Diseño Analizador Sintáctico</b>	<b>7</b>
3.1	Gramática . . . . .	7
3.2	Tabla LR(1) . . . . .	8
<b>4</b>	<b>Diseño Analizador Semántico</b>	<b>11</b>
<b>5</b>	<b>Diseño Tabla de Símbolos</b>	<b>12</b>
<b>6</b>	<b>Anexo</b>	<b>13</b>
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
<b>7</b>	<b>Diseño Generador de Código Intermedio</b>	<b>29</b>
<b>8</b>	<b>Diseño Generador de Código Final</b>	<b>33</b>
<b>9</b>	<b>Diseño Registros de Activación</b>	<b>34</b>
<b>10</b>	<b>Referencias</b>	<b>35</b>

# 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"[1].

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

$A \rightarrow \text{del } A \mid d D \mid " S \mid / C \mid l I \mid -M \mid = E \mid \& N \mid ( \mid ) \mid \{ \mid \} \mid ; \mid ,$

$D \rightarrow d D \mid \lambda$

$S \rightarrow " \mid c S$

$C \rightarrow * C'$

$C' \rightarrow * C'' \mid c C'$

$C'' \rightarrow / A \mid c C'$

$I \rightarrow d I \mid l I \mid \_ I \mid \lambda$

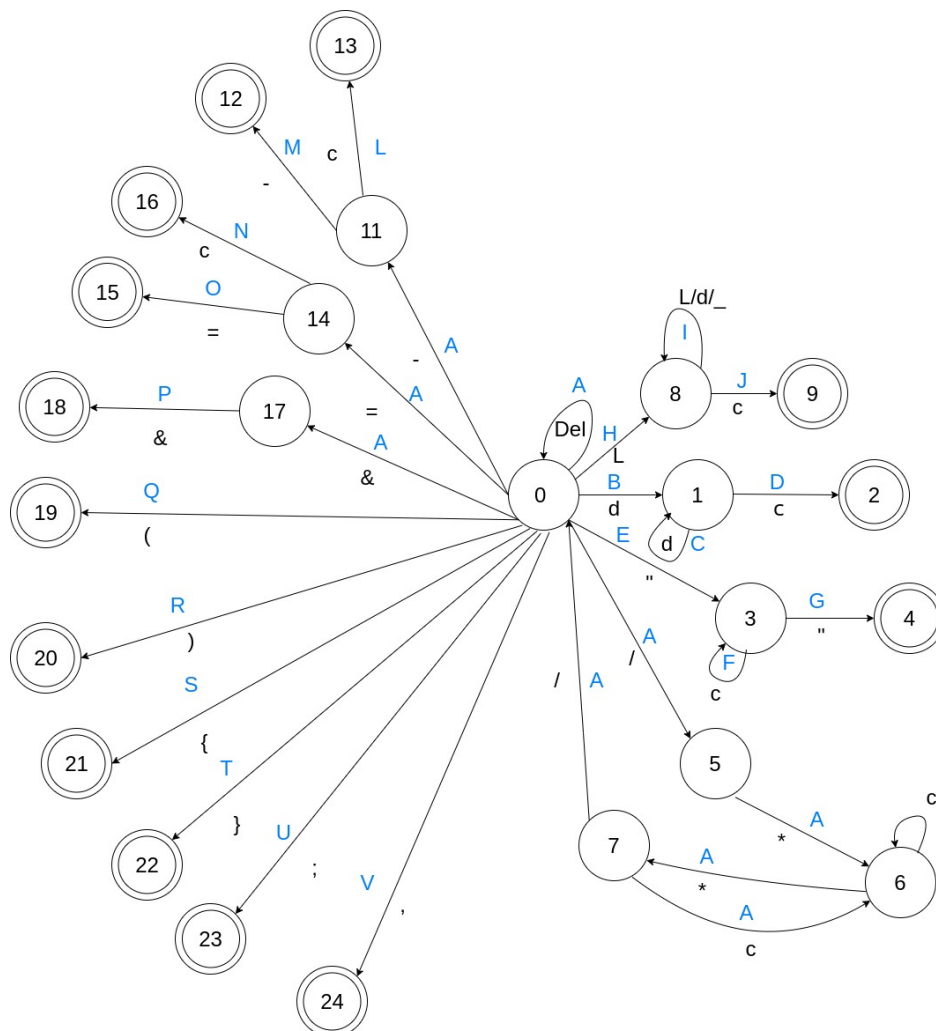
$M \rightarrow - \mid \lambda$

$E \rightarrow = \mid \lambda$

$N \rightarrow \&$

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)

State	ACTION																							
	lambda	if	(	)	:	id	,	=	alert	input	return	let	number	boolean	string	for	(	)	--	function	64	==	-	ent
0	s4	s6				s16			s13	s14	s15	s8				s9				s10				
1																								
2	s4	s6				s16			s13	s14	s15	s8				s9				s10				
3	s4	s6				s16			s13	s14	s15	s8				s9				s10				
4																								
5		s20																						
6		s21																						
7	r5	r5				r5			r5	r5	r5	r5				r5				r5				
8	s23																							
9		s24																						
10	s26																							
11					s27																			
12					s28																			
13					s29																			
14					s30																			
15	s33	s39				s38										s37								
16		s44						s45																
17																								
18																								
19																								
20	s50												s51	s52	s53									
21		s60				s59										s58								
22													s51	s52	s53									
23													r20	r20	r20									
24	s68												s51	s52	s53									
25	s72												s51	s52	s53									
26	r34												r34	r34	r34									
27	r6	r6				r6			r6	r6	r6	r6				r6				r6				
28	r12	r12				r12			r12	r12	r12	r12				r12				r12				
29		s60				s59										s58								
30						s74																		
31					s75																			
32					r17											s76								
33					r18																			
34					r44											r44	s77							
35					r46											r46	r46	s78						
36					r48											r48	r48	r48						
37					s79																			
38		s80				s80										r50	r50	r50						
39		s60				s59										s58								
40					r52											r52	r52	r52						
41					r53											r53	r53	r53						
42					r54											r54	r54	r54						
43					r55											r55	r55	r55						
44	s84	s90				s88										s88								
45		s39				s38										s37								
46	r32	r32				r32			r32	r32	r32	r32				r32								
47	s98	s99				s16			s105	s106	s107	s101				s102								
48																								
49					s108																			
50					s109																			
51					r31																			
52					r32																			
53					r33																			
54					s110																			
55					r44											s111								
56					r46											r46	s112							
57					r48											r48	r48	s113						
58					s114																			
59		s115				r50										r50	r50	r50						
60		s60				s59										s58								
61					r52											r52	r52	r52						
62					r53											r53	r53	r53						
63					r54											r54	r54	r54						
64					r55											r55	r55	r55						
65					s117																			
66					s118																			
67					r28																			
68					r26																			
69								s45																
70					s119																			
71					r35																			
72					r36																			
73					s120																			
74					s121															s111				
75	r16	r16				r16			r16	r16	r16	r16				r16				r16				
76		s39				s38										s37								
77		s39				s38										s37								
78		s39				s38										s37								
79					s49											r49	r49	r49						
80	s84	s90				s88										s88								
81		s126														s111								
82		s127																						

9

[illegible]

## 4 | Diseño Analizador Semántico

asdasdas

asda

## 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 :

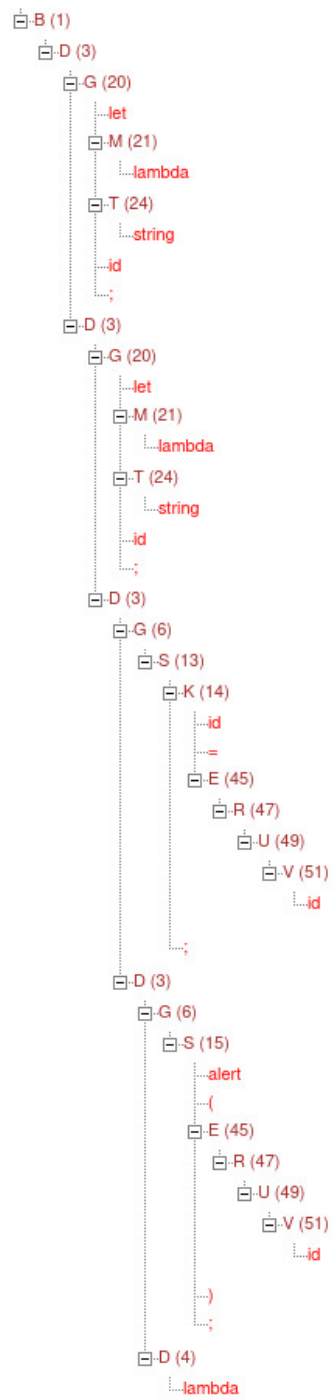
\* LEXEMA : 'texto'  
ATRIBUTOS :  
+ Tipo : 'cadena'  
+ Despl : '0'

-----  
\* 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'

-----

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

```

graph TD
    B1["B (1)"] --> D2["D (2)"]
    D2 --> F33["F (33)"]
    F33 --> F134["F1 (34)"]
    F134 --> function["function"]
    F134 --> P35["P (35)"]
    P35 --> lambda1["lambda"]
    lambda1 --> Q36["Q (36)"]
    Q36 --> T22["T (22)"]
    T22 --> number["number"]
    T22 --> id1["id"]
    T22 --> F238["F2 (38)"]
    F238 --> A39["A (39)"]
    A39 --> T22_2["T (22)"]
    T22_2 --> number_2["number"]
    T22_2 --> id2["id"]
    T22_2 --> AA42["AA (42)"]
    AA42 --> lambda2["lambda"]
    F238 --> F343["F3 (43)"]
    F343 --> C31["C (31)"]
    C31 --> G5["G (5)"]
    G5 --> if["if"]
    if --> E45["E (45)"]
    E45 --> R46["R (46)"]
    R46 --> R47["R (47)"]
    R47 --> U49["U (49)"]
    U49 --> V51["V (51)"]
    V51 --> id3["id"]
    if --> E45_2["E (45)"]
    E45_2 --> R46_2["R (46)"]
    R46_2 --> U49_2["U (49)"]
    U49_2 --> V54["V (54)"]
    V54 --> ent1["ent"]
    if --> S17["S (17)"]
    S17 --> return["return"]
    return --> L18["L (18)"]
    L18 --> E45_3["E (45)"]
    E45_3 --> R47_2["R (47)"]
    R47_2 --> U49_3["U (49)"]
    U49_3 --> V54_2["V (54)"]
    V54_2 --> ent2["ent"]
    if --> C31_2["C (31)"]
    C31_2 --> G6["G (6)"]
    G6 --> S17_2["S (17)"]
    S17_2 --> return_2["return"]
    return_2 --> L18_2["L (18)"]
    L18_2 --> E45_4["E (45)"]
    E45_4 --> R47_3["R (47)"]
    R47_3 --> U48["U (48)"]
    U48 --> U49_4["U (49)"]
    U49_4 --> V51_2["V (51)"]
    V51_2 --> id4["id"]
    U48 --> V53["V (53)"]
    V53 --> H8["H (8)"]
    H8 --> id5["id"]
    U48 --> I9["I (9)"]
    I9 --> E45_5["E (45)"]
    E45_5 --> R47_4["R (47)"]
    R47_4 --> U48_2["U (48)"]
    U48_2 --> U49_5["U (49)"]
    U49_5 --> V51_3["V (51)"]
    V51_3 --> id6["id"]
    U48_2 --> V54_3["V (54)"]
    V54_3 --> ent3["ent"]
    I9 --> J11["J (11)"]
    J11 --> lambda3["lambda"]
    I9 --> C32["C (32)"]
    C32 --> lambda4["lambda"]
    I9 --> D4["D (4)"]
    D4 --> lambda5["lambda"]

```

### 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 , >

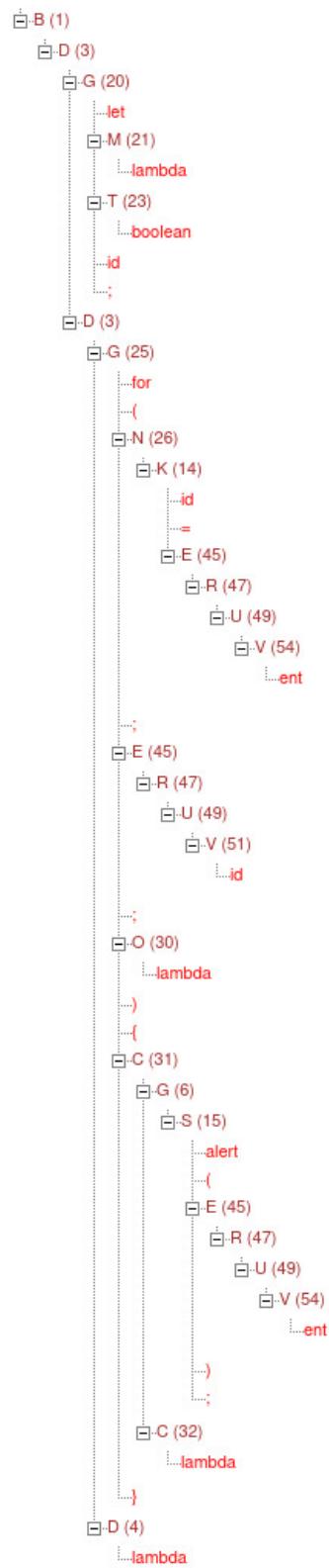
Tabla de símbolos:

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:**

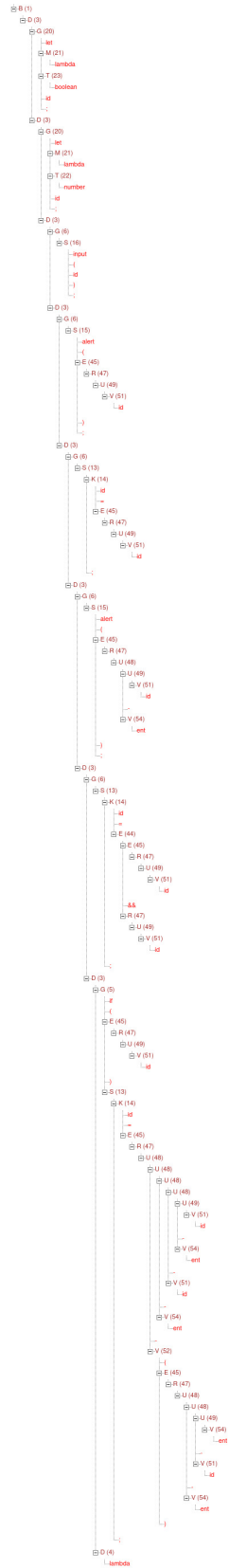
CONTENIDO DE LA TABLA 0 :

```

*      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'

-----

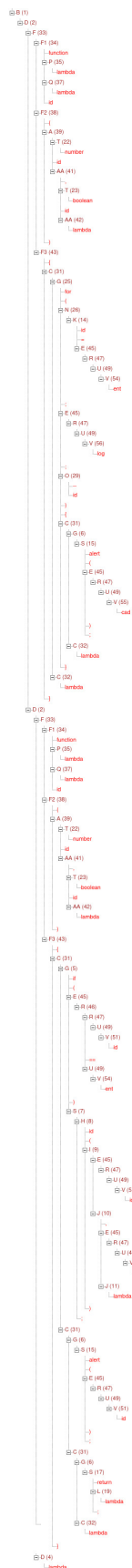
CONTENIDO DE LA TABLA 2 :

\*        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

## 7 | Diseño Generador de Código Intermedio

{ generar etiqueta alert/input ¿? }

Axioma = B

NoTerminales = { A AA B C D E F F1 F2 F3 G H I J K L M N O P Q R S T U V }

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

B → D {

    B.cod = D.cod

}

D → F D {

    D.cod = F.cod || D.cod

}

D → G D {

    D.cod = G.cod || D.cod

}

D → lambda {

    D.cod = vacio

}

G → if ( E ) S {

    G.desp = nuevaetiq()

    G.cod = E.cod || gen(if, E.lugar, =, 0, goto, G.desp) || S.cod || gen(:, G.desp)

}

G → S {

    G.cod = S.cod

}

S → H ; {

    S.cod = H.cod

}

H → id ( I ) {

    H.lugar = nuevatempl()

    H.cod = I.codE || I.codP || gen(H.lugar, =, call, buscaEtiquetaTS(id.pos))

}

I → E J {

    I.codE = E.cod || J.codE

    I.codP = gen(param, E.lugar) || J.codP

}

J → , E J {

    J.codE = E.cod || J1.codE

    J.codP = gen(param, E.lugar) || J1.codP

}

J → lambda {

    L.codE = vacio

    L.codP = vacio

}

I → lambda {

    J.codE = vacio

    J.codP = vacio

}

```

S → K ; {
    S.cod = K.cod
}
K → id = E {
    K.cod = E.cod || gen(buscaLugarTS(id.pos), =, E.lugar)
}

S → alert ( E ) ; {
    S.cod = E.cod || gen(param, E.lugar) || gen(call, alert.etiq) -----¿?
}
S → input ( id ) ; {
    S.cod = gen(param, buscaLugarTS(id.pos)) || gen(call, input.etiq) -----¿?
}

S → return L ; {
    if( L.cod = vacio )
        S.cod = gen(return)
    else
        S.cod = L.cod || gen(return, L.lugar)
}
L → E {
    L.cod = E.cod
    L.lugar = E.lugar
}
L → lambda {
    L.cod = vacio
}

G → let M T id ; {
    B.cod = vacio
}
M → lambda { }
T → number { }
T → boolean { }
T → string { }

G → for ( N ; E ; O ) { C } {
    G.inicio = nuevaetiq()
    G.desp = nuevaetiq()
    G.cod = N.cod || gen(:, G.inicio) || E.cod || gen(if, E.lugar, =, 0, goto, G.desp) || C.cod || O.cod || gen(goto, G
}
N → K {
    N.cod = K.cod
}
N → lambda {
    N.cod = vacio
}

O → K {
    O.cod = K.cod
}
O → -- id {
    O.lugar = nuevatemp()
    id.lugar = buscaLugarTS(id.pos)
}

```

```

    O.cod = gen(id.lugar, =, id.lugar, -, 1) || gen(O.lugar, =, id.lugar)
  }
O → lambda {
  O.cod = vacio
}

```

```

C → G C1 {
  C.cod = G.cod || C1.cod
}
C → lambda {
  C.cod = vacio
}

```

```

F → F1 F2 F3 {
  F.cod = F1.cod || F2.cod || F3.cod || gen(return)
}
F1 → function P Q id {
  F1.cod = gen(:, buscaEtiquetaTS(id.pos))
}
P → lambda { }
Q → T { }
Q → lambda { }

```

```

F2 → ( A ) {
  F2.cod = vacio
}
A → T id AA { }
A → lambda { }
AA → , T id AA { }
AA → lambda { }

```

```

F3 → { C } {
  F3.cod = C.cod
}

```

```

E → E1 && R {
  E.lugar = nuevatemp();
  E.cod = E1.cod || R.cod || gen(E.lugar, =, E1.lugar, AND, R.lugar)
}
E → R {
  E.lugar = R.lugar
  E.cod = R.cod
}

```

```

R → R1 == U {
  R.cod = R1.cod || U.cod || gen(if, R1.lugar, =, U.lugar, goto, esta + 3) || gen(R.lugar, =, 0) || gen(goto, esta -
}
R → U {
  R.cod = U.cod
  R.lugar = U.lugar
}

```

```

U → U1 - V {
  U.lugar = nuevatemp()
}

```



```

    U.cod = gen(U.lugar, =, U1.lugar, -, V.lugar)
  }
U → V {
  U.lugar = V.lugar
  U.cod = V.cod
}

V → -- id {
  V.lugar = nuevatemp()
  id.lugar = buscaLugarTS(id.pos)
  V.cod = gen(id.lugar, =, id.lugar, -, 1) || gen(V.lugar, =, id.lugar)
}
V → id {
  V.lugar = buscaLugarTS(id.pos)
  V.cod = vacio
}
V → ( E ) {
  V.lugar = H.lugar
  V.cod = H.cod
}
V → H {
  V.lugar = H.lugar
  V.cod = H.cod
}

V → ent {
  V.lugar = nuevatemp()
  V.cod = gen(V.lugar, =, ent.valor)
}
V → cad {
  V.lugar = nuevatemp()
  V.cod = gen(V.lugar, =, cad.valor)
}
V → log {
  V-lugar = nuevatmp()
  V.cod = gen(V.lugar, =, log.valor)
}

```

## 8 | Diseño Generador de Código Final

## 9 | Diseño Registros de Activación

## 10 | Referencias

1. *Documentación librería SLY*  
<https://sly.readthedocs.io/en/latest/>
2. *Generador de tabla LR(1)*  
<http://jsmachines.sourceforge.net/machines/lr1.html>