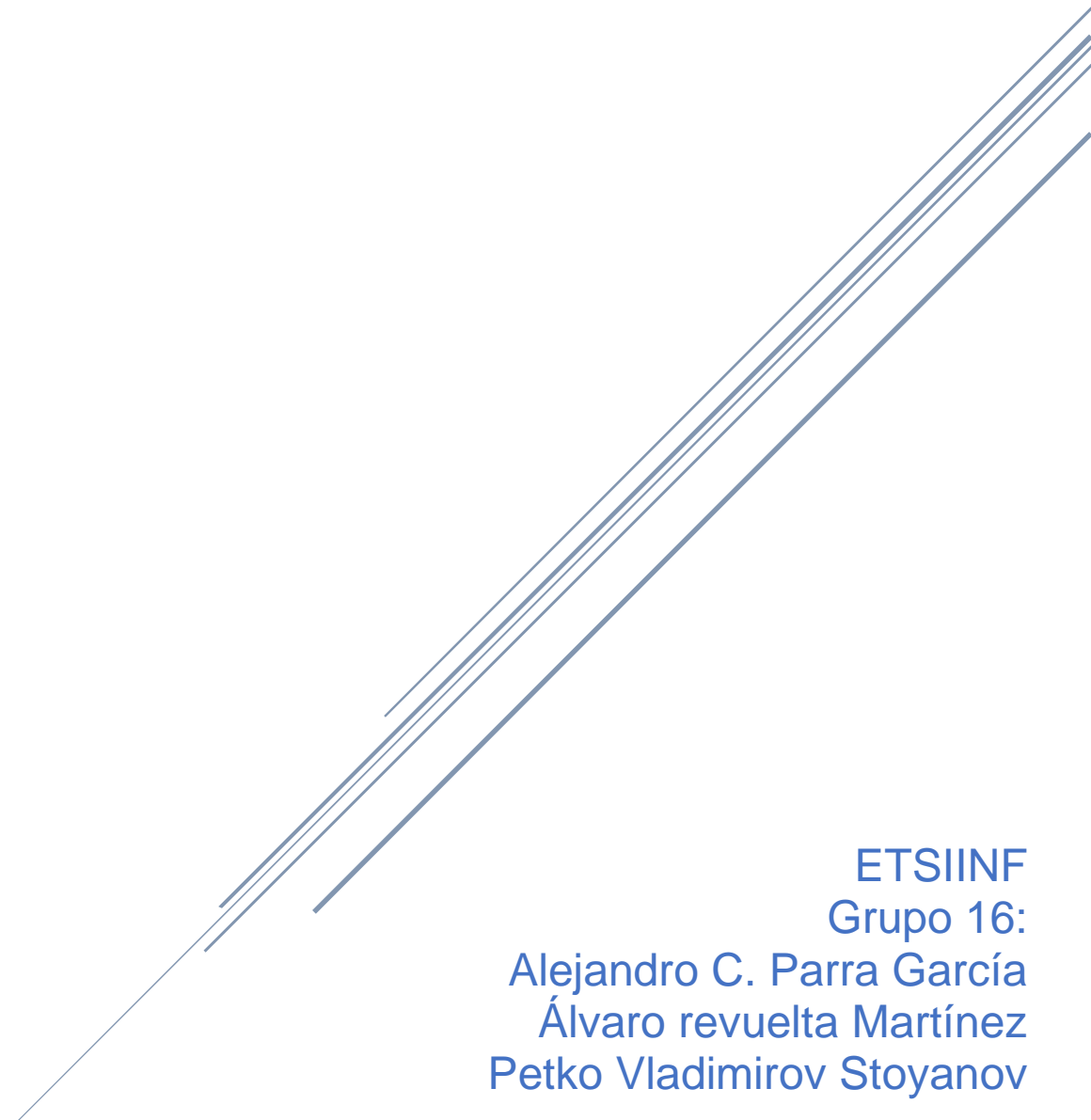


# MEMORIA PDL

Analizador Léxico + Tabla de símbolos + Sintáctico +  
Semántico + Gestor de Errores



ETSIINF  
Grupo 16:  
Alejandro C. Parra García  
Álvaro revuelta Martínez  
Petko Vladimirov Stoyanov

## Contenido

Introducción	2
Analizador Léxico	2
Tokens	2
Gramática regular	3
Autómata finito determinista	4
Acciones Semánticas	5
Errores	7
Tabla de Símbolos	7
Ejemplo:	7
Analizador Sintáctico	8
Gramática	8
First y Follow	9
Comprobación de la condición LL(1)	11
Tabla	14
Analizador Semántico	15
Gramática	15
Tabla de Símbolos Final	19
Ejemplo:	19
Anexo	20
/ Caso N1 /	20
/ Caso N2 /	27
/ Caso N3 /	27
/ Caso N4 /	27
/ Caso N5 /	28
/ Caso N6 /	28
/ Caso N7 /	28
/ Caso N8 /	29
/ Caso N9 /	29
/ Caso N10 /	29

## Introducción

Las opciones específicas que se nos han asignado para el desarrollo de la práctica son las siguientes:

- Sentencias: **Sentencia repetitiva (for)**
- Operadores especiales: **Asignación con o lógico (|=)**
- Técnicas de Análisis Sintáctico: **Descendente con tablas**
- Comentarios: **Comentario de bloque (/\* \*/)**
- Cadenas: **Con comillas dobles (" ")**

Adicionalmente, hemos elegido implementar el operador suma(+) como operador aritmético, la negación(!) como operador lógico y el mayor(>) como operador relacional.

## Analizador Léxico

### Tokens

En este apartado se diseñan los tokens de nuestra práctica y los cuales los vamos a usar. Atendiendo al lenguaje especificado para la práctica, estas serían las **palabras reservadas**:

<var,->  
<int,->  
<boolean,->  
<string,->  
<for,->  
<print,->  
<input,->

<function,->  
<if,->  
<return,->

Para las **variables de usuario** se usaría:

<id,posTS>

Los **números** se representan mediante un entero:

<entero,valor>

Las **cadenas** irían entre comillas dobles (") y se representaría pasando la cadena en el atributo.

<cadena,lexema>

La **asignación** iría representada por el símbolo igual (=)

<asignación,->

Como hemos explicado en la introducción, el **operador Aritmético** es el más (+). Para el **Relacional** la negación (!). Y para el **lógico** el mayor (>).

<opArt,->  
<opRel,->  
<opLog,->

El **operador especial** asignado a nuestro grupo sería la Asignación con o lógico (|=)

<opEsp,->

La **coma** (,) y el **punto y coma** (;)

<coma,->  
<puntoComa,->

Tanto los **paréntesis** como los **corchetes** serían un token cada uno, diferenciando los abiertos de los cerrados en el nombre del código del token.

<parAb,->  
<parCer,->  
<corcheteAb,->  
<corcheteCer,->

Gramática regular

Del: b, <tab>, <eol>    l: a\_z, A\_Z    d: 0\_9    p: car. Imprimible    o: otro car.

$S \rightarrow \text{del}S \mid IA \mid dB \mid |C \mid "D \mid + \mid > \mid ! \mid = \mid , \mid ; \mid ( \mid ) \mid \{ \mid \} \mid /F$

$A \rightarrow IA \mid dA \mid \_A \mid \lambda$

$B \rightarrow dB \mid \lambda$

$C \rightarrow =$

$D \rightarrow pD \mid "$

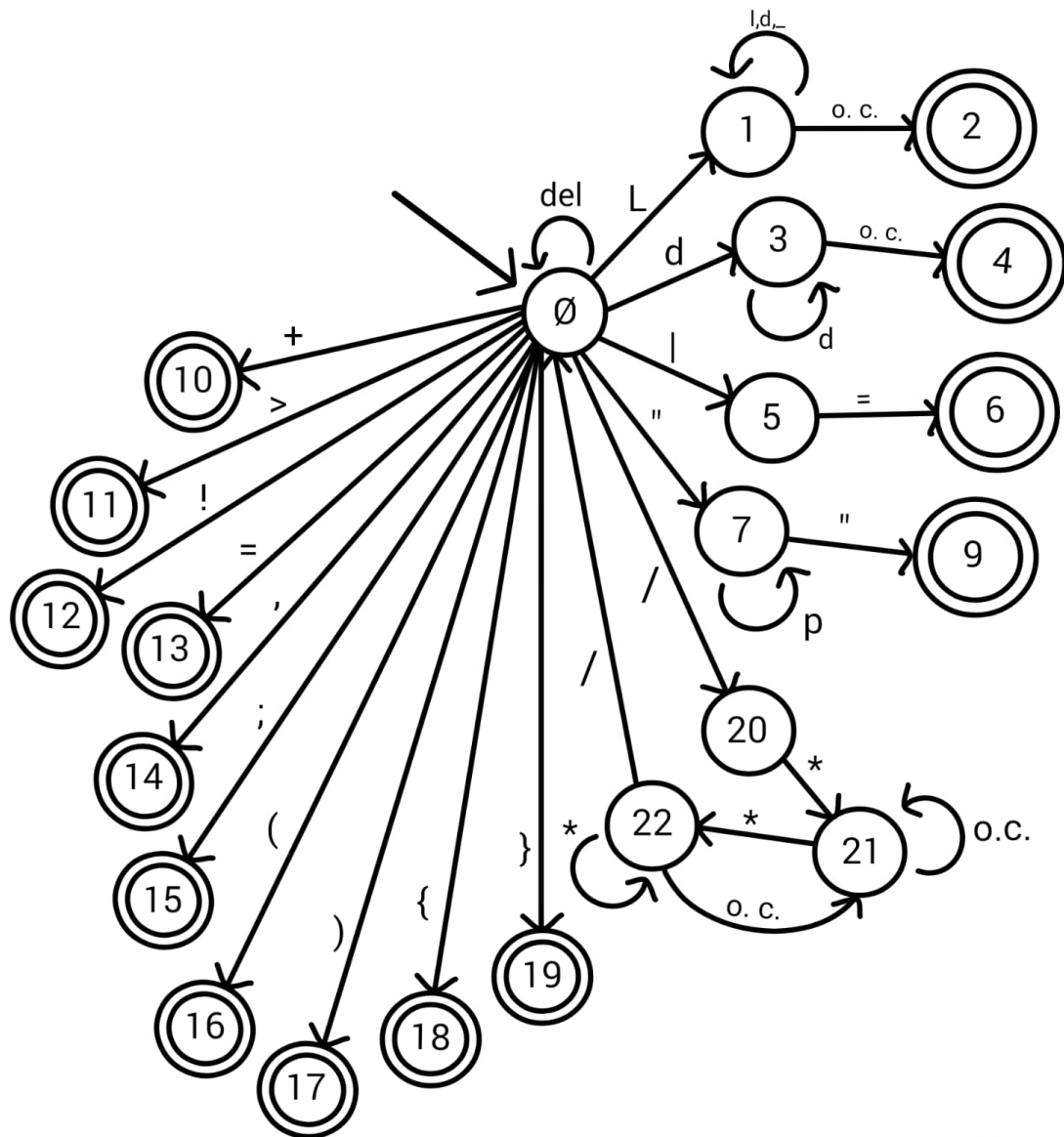
$F \rightarrow *G$

$G \rightarrow oG \mid *H$

$H \rightarrow /S \mid oG \mid *H$

Autómata finito determinista

Del: b, <tab>, <eol>    l: a\_z, A\_Z    d: 0\_9    p: car. Imprimible    o.c.: otro car.



Powered by Skedio

### Acciones Semánticas

Concat(X,Y) ; concatena Y justo después de X.

El uso de [ X | Y ] ; significa que se han leído o una X o una Y.

Gentoken(X,Y;) genera un token con el código X y el atributo Y, el atributo Y puede ir vacío lo que se indicará con un guion (-).

X = buscarTS( Y ) ; busca Y en la tabla de símbolos y devuelve su posición que se guarda en X, si Y no está en la tabla de símbolos devuelve un "null"

X = insertarTS( Y ) ; inserta Y en la tabla de símbolos y devuelve la posición donde se inserta que se guarda en X.

0.0 Leer(car)

0.1 Lexema=l ; Leer(car)

1.1 Lexema = Concat(lexema,[ l | d | \_ ] ) ; Leer(car)

1.2 If Lexema ∈ Tabla = Pal.Clave

Then Gentoken(Lexema,-)

Else q=buscarTS(lexema)

If q=NULL

Then q=insertarTS(lexema)

Gentoken(id,q)

0.3 Valor= ascii(d) ; Leer(car)

3.3 Valor=Valor\*10+ascii(d) ; Leer(car)

3.4 If Valor < 32767

then Gentoken(entero,valor)

else Error("entero mayor o igual a 32767")

0.5 Leer(car)

5.6 Gentoken (opEsp,-) ; Leer(car)

0.7 Lexema= " ; contc=0; Leer(car)

7.7 Lexema = Concat(lexema, p ) ; contc++; Leer(car)

7.9 Lexema=Concat(lexema, " );

if(contc<=64)

Gentoken(cad,lexema) ; Leer(car)

else

Error("cadena de más de 64 caracteres")

0.10 Gentoken(opArit,-) ; Leer(car)

0.11 Gentoken(opRel,-) ; Leer(car)

0.12 Gentoken(opLog,-) ; Leer(car)

0.13 Gentoken(asignación,-) ; Leer(car)

0.14 Gentoken(coma,-) ; Leer(car)

0.15 Gentoken(puntoComa,-) ; Leer(car)

0.16 Gentoken(parAb,-) ; Leer(car)

0.17 Gentoken(parCer,-) ; Leer(car)

0.18 Gentoken(corcheteAb,-) ; Leer(car)

0.19 Gentoken(corcheteCer,-) ; Leer(car)

0.20 Leer(car)

20.21 Leer(car)

21.21 Leer(car)

21.22 Leer(car)

22.21 Leer(car)

22.0 Leer(car)

## Errores

Los mensajes de error se crean y envían desde el módulo de control de errores. Un analizador léxico tendría que enviar a dicho módulo un código de error para cada operación que no contemple el autómata finito, y también de aquellos errores que hemos definido en las acciones semánticas.

## Tabla de Símbolos

La tabla de símbolos se usa para almacenar la información de los identificadores. Se usa para almacenar y consultar el contenido.

En la tabla se almacena múltiple información:

Lexema: Caracteres que representan el lexema del código fuente.

Tipo de dato

Desplazamiento

N argumentos, en el caso de que sea una función

Tipo de los argumentos, en el caso de que tenga parámetros.

Tipo retorno, en el caso que sea una función y que devuelva algún dato

La tabla de símbolos se estructura en una tabla global y en locales. Las locales se usan para funciones.

### Ejemplo:

```
Var a124 int;  
var boolean b;  
function int SumaAlCuadrado (int ca5e7, int ee78) {  
  ...  
  ...  
}
```

Tabla Global

Iden.	Lexema	Tipo	Desp	Nº args	Tipos args	Tipo retorno
1	A124	entero	0			
2	B	boolean	2			
3	SumaAlCuadrado	Función		2	entero X entero	entero

Tabla local

Iden.	Lexema	Tipo	Desp	Nº args	Tipos args	Tipo retorno
-------	--------	------	------	---------	------------	--------------



1	ca5e7	entero	0			
2	ee78	entero	2			

## Analizador Sintáctico

### Gramática

La gramática inicial posee errores de recursividad y de factorización. Estos errores están marcados en **rojo** y después ellos indicados los cambios necesarios para arreglarlos.

Gramática inicial.

$P \rightarrow DP \mid FP \mid SP \mid \underline{SC} P \mid \lambda$

$D \rightarrow \text{var } T \text{ id};$

$T \rightarrow \text{int} \mid \text{string} \mid \text{boolean}$

$F \rightarrow \text{function } T' \text{ id } (A) \{ C \}$

$T' \rightarrow T \mid \lambda$

$A \rightarrow \lambda \mid T \text{ id } A'$

$A' \rightarrow \lambda \mid , T \text{ id } A'$

$C \rightarrow DC \mid SC \mid \lambda \mid \underline{SC} C$

$S \rightarrow \text{if}(E) S \mid \text{id} = E; \mid \text{id} \mid = E; \mid \text{print}(E); \mid \text{input}(E); \mid \text{id}(L); \mid \text{return } X;$

$S \rightarrow \text{if}(E) S \mid \text{idO} \mid \text{print}(E); \mid \text{input}(E); \mid \text{return } X;$

$O \rightarrow = E; \mid \mid = E; \mid (L);$

$L \rightarrow \lambda \mid E L'$

$L' \rightarrow \lambda \mid , EL'$

$X \rightarrow E \mid \lambda$

$\underline{SC} \rightarrow \text{for}(B; E; B') \{ C \}$

$B \rightarrow \text{id} = E \mid \lambda$

$B' \rightarrow \text{id} = E \mid \text{id} \mid = E \mid \lambda$

$B' \rightarrow \text{id } B'' \mid \lambda$

$B'' \rightarrow = E \mid \mid = E$

//Antes de quitar la recursividad

$E \rightarrow E > K$

$E \rightarrow K$

$K \rightarrow ! K$

$K \rightarrow U$

$U \rightarrow U + V$

$U \rightarrow V$

$V \rightarrow \text{id} \mid \text{cte-entera} \mid \text{cadena} \mid (E) \mid \text{id}(L)$

//Una vez quitada la recursividad

$E \rightarrow KE'$

$E' \rightarrow \lambda \mid >KE'$   
 $K \rightarrow !K$   
 $K \rightarrow U$   
 $U \rightarrow VU'$   
 $U' \rightarrow \lambda \mid +VU'$

$V \rightarrow id \mid \text{cte-entera} \mid \text{cadena} \mid (E) \mid id(L)$   
 $V \rightarrow idJ \mid \text{cte-entera} \mid \text{cadena} \mid (E)$   
 $J \rightarrow (L) \mid \lambda$

Una vez aplicado los cambios la gramática resultante es la siguiente:

#### GRAMÁTICA DEFINITIVA

$P \rightarrow DP \mid FP \mid SP \mid \underline{SC} P \mid \lambda$   
 $D \rightarrow \text{var } T \text{ id};$   
 $T \rightarrow \text{int} \mid \text{string} \mid \text{boolean}$   
 $F \rightarrow \text{function } T' \text{ id } (A) \{ C \}$   
 $T' \rightarrow T \mid \lambda$   
 $A \rightarrow \lambda \mid T \text{ id } A'$   
 $A' \rightarrow \lambda \mid ,T \text{ id } A'$   
 $C \rightarrow DC \mid SC \mid \lambda \mid \underline{SC} C$   
 $S \rightarrow \text{if}(E) S \mid \text{idO} \mid \text{print}(E); \mid \text{input}(E); \mid \text{return } X;$   
 $O \rightarrow = E; \mid \mid = E; \mid (L);$

$L \rightarrow \lambda \mid E L'$   
 $L' \rightarrow \lambda \mid ,EL'$

$X \rightarrow E \mid \lambda$   
 $\underline{SC} \rightarrow \text{for}(B; E; B') \{ C \}$   
 $B \rightarrow \text{id} = E \mid \lambda$   
 $B' \rightarrow \text{id } B'' \mid \lambda$   
 $B'' \rightarrow = E \mid \mid = E$   
 $E \rightarrow KE'$   
 $E' \rightarrow \lambda \mid >KE'$   
 $K \rightarrow !K$   
 $K \rightarrow U$   
 $U \rightarrow VU'$   
 $U' \rightarrow \lambda \mid +VU'$   
 $V \rightarrow idJ \mid \text{cte-entera} \mid \text{cadena} \mid (E)$   
 $J \rightarrow (L) \mid \lambda$

First y Follow

$\text{First}(P): \{ \text{var}, \text{function}, \text{if}, \text{id}, \text{print}, \text{input}, \text{return}, \text{for}, \lambda \}$   
 $\text{First}(D): \{ \text{var} \}$   
 $\text{First}(T): \{ \text{int}, \text{string}, \text{boolean} \}$

First(F): { function }  
 First(T'): { int, string, boolean,  $\lambda$  }  
 First(A): { int, string, boolean,  $\lambda$  }  
 First(A'): { ,,  $\lambda$  }  
 First(C): { var, if, id, print, input, return, for,  $\lambda$  }  
 First(S): { if, id, print, input, return }  
 First(O): { =, |=, ( }  
  
 First(L): {  $\lambda$ , id, cte-entera, cadena, (, ! }  
 First(L'): {  $\lambda$ , , }  
  
 First(X): {  $\lambda$ , id, cte-entera, cadena, (, ! }  
 First(SC): { for }  
 First(B): { id,  $\lambda$  }  
 First(B'): { id,  $\lambda$  }  
 First(B''): { =, |= }  
 First(E): { id, cte-entera, cadena, (, ! }  
 First(E'): {  $\lambda$ , > }  
 First(K): { id, cte-entera, cadena, (, ! }  
 First(U): { id, cte-entera, cadena, ( }  
 First(U'): {  $\lambda$ , + }  
 First(V): { id, cte-entera, cadena, ( }  
 First(J): { (,  $\lambda$  }  
  
 Follow(P): { \$ }  
 Follow(D): { var, function, if, id, print, input, return, for, }, \$ }  
 Follow(T): { id }  
 Follow(F): { var, function, if, id, print, input, return, for, \$ }  
 Follow(T'): { id }  
 Follow(A): { ) }  
 Follow(A'): { ) }  
 Follow(C): { } }  
 Follow(S): { var, function, if, id, print, input, return, for, }, \$ }  
 Follow(O): { var, function, if, id, print, input, return, for, }, \$ }  
  
 Follow(L): { ) }  
 Follow(L'): { ) }  
  
 Follow(X): { ; }  
 Follow(SC): { var, function, if, id, print, input, return, for, }, \$ }  
 Follow(B): { ; }  
 Follow(B'): { ) }  
 Follow(B''): { ) }  
 Follow(E): { ), ; }  
 Follow(E'): { ), ; }  
 Follow(K): { >, ), ; }  
 Follow(U): { >, ), ; }  
 Follow(U'): { >, ), ; }  
 Follow(V): { +, >, ), ; }  
 Follow(J): { +, >, ), ; }

## Comprobación de la condición LL(1)

Una gramática es LL(1) si:

- Para cada no terminal A para el que haya más de una producción en G:
  - Para cada par de producciones  $A \rightarrow \alpha \mid \beta$  se cumplen las siguientes condiciones:
    - i. No existe ningún terminal  $t$  tal que tanto  $\alpha$  como  $\beta$  deriven de cadenas que empiezan por el mismo terminal, es decir,  
$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \text{VACÍO}$$
  - A lo sumo uno puede derivar de la cadena lambda pero no ambos.  
Supongamos que  $\beta$  puede derivarla. En este caso:  
$$\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \text{VACÍO}$$

### 1) $P \rightarrow DP \mid FP \mid SP \mid \underline{SC} P \mid \lambda$

- a)  $\text{FIRST}(DP) \cap \text{FIRST}(FP) = \{\text{var}\} \cap \{\text{function}\} = \emptyset$
- b)  $\text{FIRST}(DP) \cap \text{FIRST}(SP) = \{\text{var}\} \cap \{\text{if, id, print, input, return}\} = \emptyset$
- c)  $\text{FIRST}(DP) \cap \text{FIRST}(\underline{SC} P) = \{\text{var}\} \cap \{\text{for}\} = \emptyset$
- d)  $\text{FIRST}(DP) \cap \text{FIRST}(\lambda) = \{\text{var}\} \cap \{\lambda\} = \emptyset$   
Como hay lambda  $\Rightarrow$ 
  - i)  $\text{FIRST}(DP) \cap \text{FOLLOW}(P) = \{\text{var}\} \cap \{\$ \} = \emptyset$
- e)  $\text{FIRST}(FP) \cap \text{FIRST}(SP) = \{\text{function}\} \cap \{\text{if, id, print, input, return}\} = \emptyset$
- f)  $\text{FIRST}(FP) \cap \text{FIRST}(\underline{SC} P) = \{\text{function}\} \cap \{\text{for}\} = \emptyset$
- g)  $\text{FIRST}(FP) \cap \text{FIRST}(\lambda) = \{\text{function}\} \cap \{\lambda\} = \text{VACÍO}$   
Como hay lambda  $\Rightarrow$ 
  - i)  $\text{FIRST}(FP) \cap \text{FOLLOW}(P) = \{\text{function}\} \cap \{\$ \} = \emptyset$
- h)  $\text{FIRST}(SP) \cap \text{FIRST}(\underline{SC} P) = \{\text{if, id, print, input, return}\} \cap \{\text{for}\} = \emptyset$
- i)  $\text{FIRST}(SP) \cap \text{FIRST}(\lambda) = \{\text{if, id, print, input, return}\} \cap \{\lambda\} = \emptyset$   
Como hay lambda  $\Rightarrow$ 
  - i)  $\text{FIRST}(SP) \cap \text{FOLLOW}(P) = \{\text{if, id, print, input, return}\} \cap \{\$ \} = \emptyset$
- j)  $\text{FIRST}(\underline{SC} P) \cap \text{FIRST}(\lambda) = \{\text{for}\} \cap \{\lambda\} = \emptyset$   
Como hay lambda  $\Rightarrow$ 
  - i)  $\text{FIRST}(\underline{SC} P) \cap \text{FOLLOW}(P) = \{\text{for}\} \cap \{\$ \} = \emptyset$

### 2) $T \rightarrow \text{int} \mid \text{string} \mid \text{boolean}$ $\text{FIRST}(\text{int}) \cap \text{FIRST}(\text{string}) = \{\text{int}\} \cap \{\text{string}\} = \emptyset$

- a)  $\text{FIRST}(\text{int}) \cap \text{FIRST}(\text{boolean}) = \{\text{int}\} \cap \{\text{boolean}\} = \emptyset$
- b)  $\text{FIRST}(\text{string}) \cap \text{FIRST}(\text{boolean}) = \{\text{string}\} \cap \{\text{boolean}\} = \emptyset$

### 3) $T' \rightarrow T \mid \lambda$

- a)  $\text{FIRST}(T) \cap \text{FIRST}(\lambda) = \{\text{int, string, boolean}\} \cap \{\lambda\} = \emptyset$   
Como hay lambda  $\Rightarrow$ 
  - i)  $\text{FIRST}(T) \cap \text{FOLLOW}(T') = \{\text{int, string, boolean}\} \cap \{\text{id}'\} = \emptyset$

### 4) $A \rightarrow \lambda \mid T \text{ id } A'$

- a)  $\text{FIRST}(\lambda) \cap \text{FIRST}(T \text{ id } A') = \{\lambda\} \cap \{\text{int, string, boolean}\} = \emptyset$   
Como hay lambda  $\Rightarrow$ 
  - i)  $\text{FIRST}(T \text{ id } A') \cap \text{FOLLOW}(A) = \{\text{int, string, boolean}\} \cap \{\} = \emptyset$

**5)  $A' \rightarrow \lambda \mid , T \text{ id } A'$**

- a)  $\text{FIRST}(\lambda) \wedge \text{FIRST}(, T \text{ id } A') = \{\lambda\} \wedge \{, \} = \emptyset$   
 Como hay lambda =>  
 i)  $\text{FIRST}(, T \text{ id } A') \wedge \text{FOLLOW}(A') = \{, \} \wedge \{ \} = \emptyset$

**6)  $C \rightarrow DC \mid SC \mid \lambda \mid \underline{SC} C$**

- a)  $\text{FIRST}(DC) \wedge \text{FIRST}(SC) = \{\text{var}\} \wedge \{\text{for}\} = \emptyset$   
 b)  $\text{FIRST}(DC) \wedge \text{FIRST}(\underline{SC} C) = \{\text{var}\} \wedge \{\text{for}\} = \emptyset$   
 c)  $\text{FIRST}(DC) \wedge \text{FIRST}(\lambda) = \{\text{var}\} \wedge \{\lambda\} = \emptyset$   
 como hay lambda =>  
 i)  $\text{FIRST}(DC) \wedge \text{FOLLOW}(C) = \{\text{var}\} \wedge \{ \} = \emptyset$   
 d)  $\text{FIRST}(SC) \wedge \text{FIRST}(\underline{SC} C) = \{\text{if}, \text{id}, \text{print}, \text{input}, \text{return}\} \wedge \{\text{for}\} = \emptyset$   
 e)  $\text{FIRST}(SC) \wedge \text{FIRST}(\lambda) = \{\text{for}\} \wedge \{\lambda\} = \emptyset$   
 Como hay lambda =>  
 i)  $\text{FIRST}(SC) \wedge \text{FOLLOW}(C) = \{\text{for}\} \wedge \{ \} = \emptyset$   
 f)  $\text{FIRST}(\underline{SC} C) \wedge \text{FIRST}(\lambda) = \{\text{for}\} \wedge \{\lambda\} = \emptyset$   
 Como hay labda =>  
 i)  $\text{FIRST}(\underline{SC} C) \wedge \text{FOLLOW}(C) = \{\text{for}\} \wedge \{ \} = \emptyset$

**7)  $S \rightarrow \text{if}(E) S \mid \text{id } O \mid \text{print}(E); \mid \text{input}(E); \mid \text{return } X;$**

- a)  $\text{FIRST}(\text{if}(E) S) \wedge \text{FIRST}(\text{id } O) = \{\text{if}\} \wedge \{\text{id}\} = \emptyset$   
 b)  $\text{FIRST}(\text{if}(E) S) \wedge \text{FIRST}(\text{print}(E);) = \{\text{if}\} \wedge \{\text{print}\} = \emptyset$   
 c)  $\text{FIRST}(\text{if}(E) S) \wedge \text{FIRST}(\text{input}(E);) = \{\text{if}\} \wedge \{\text{input}\} = \emptyset$   
 d)  $\text{FIRST}(\text{if}(E) S) \wedge \text{FIRST}(\text{return } X) = \{\text{if}\} \wedge \{\text{return}\} = \emptyset$   
 e)  $\text{FIRST}(\text{id } O) \wedge \text{FIRST}(\text{print}(E)) = \{\text{id}\} \wedge \{\text{print}\} = \emptyset$   
 f)  $\text{FIRST}(\text{id } O) \wedge \text{FIRST}(\text{input}(E)) = \{\text{id}\} \wedge \{\text{input}\} = \emptyset$   
 g)  $\text{FIRST}(\text{id } O) \wedge \text{FIRST}(\text{return } X) = \{\text{id}\} \wedge \{\text{return}\} = \emptyset$   
 h)  $\text{FIRST}(\text{print}(E)) \wedge \text{FIRST}(\text{input}(E)) = \{\text{print}\} \wedge \{\text{input}\} = \emptyset$   
 i)  $\text{FIRST}(\text{print}(E)) \wedge \text{FIRST}(\text{return } X) = \{\text{print}\} \wedge \{\text{return}\} = \emptyset$   
 j)  $\text{FIRST}(\text{input}(E)) \wedge \text{FIRST}(\text{return } X) = \{\text{input}\} \wedge \{\text{return}\} = \emptyset$

**8)  $O \rightarrow = E; \mid \mid = E; \mid (L);$**

- a)  $\text{FIRST}(= E;) \wedge \text{FIRST}(\mid = E;) = \{=\} \wedge \{\mid =\} = \emptyset$   
 b)  $\text{FIRST}(= E;) \wedge \text{FIRST}((L)) = \{=\} \wedge \{( \} = \emptyset$   
 c)  $\text{FIRST}(\mid = E;) \wedge \text{FIRST}((L)) = \{\mid =\} \wedge \{( \} = \emptyset$

**9)  $L \rightarrow \lambda \mid EL'$**

- a)  $\text{First}(\lambda) \wedge \text{First}(EL') = \{\lambda\} \wedge \{\text{id}, \text{cte-entera}, \text{cadena}, (, !\} = \emptyset$   
 Como hay lambda =>  
 i)  $\text{Follow}(L) \wedge \text{First}(EL') = \{ \} \wedge \{\text{id}, \text{cte-entera}, \text{cadena}, (, !\} = \emptyset$

**10)  $L' \rightarrow \lambda \mid , E L'$**

- a)  $\text{FIRST}(, E L') \wedge \text{FIRST}(\lambda) = \{, \} \wedge \{\lambda\} = \emptyset$   
 Como hay lambda =>  
 i)  $\text{FIRST}(, E L') \wedge \text{FOLLOW}(L') = \{, \} \wedge \{ \} = \emptyset$

**11)  $X \rightarrow E \mid \lambda$**

- a)  $\text{FIRST}(E) \wedge \text{FIRST}(\lambda) = \{\text{id}, \text{cte-entera}, \text{cadena}, (, !\} \wedge \{\lambda\} = \emptyset$   
 Como hay lambda =>  
 i)  $\text{FIRST}(E) \wedge \text{FOLLOW}(X) = \{\text{id}, \text{cte-entera}, \text{cadena}, (, !\} \wedge \{ ; \} = \emptyset$

**12)  $B \rightarrow id = E \mid \lambda$**

- a)  $FIRST(id = E) \wedge FIRST(\lambda) = \{id\} \wedge \{\lambda\} = \emptyset$   
Como hay lambda =>  
i)  $FIRST(id = E) \wedge FOLLOW(B) = \{id\} \wedge \{ ; \} = \emptyset$

**13)  $B' \rightarrow id B'' \mid \lambda$**

- a)  $FIRST(id B'') \wedge FIRST(\lambda) = \{id\} \wedge \{\lambda\} = \emptyset$   
Como hay lambda =>  
i)  $FIRST(id B'') \wedge FOLLOW(B') = \{id\} \wedge \{ \} = \emptyset$

**14)  $B'' \rightarrow = E \mid |=$**

- a)  $FIRST(= E) \wedge FIRST(|=) = \{=\} \wedge \{ |= \} = \emptyset$

**15)  $E' \rightarrow \lambda \mid >KE'$**

- a)  $FIRST(>KE') \wedge FIRST(\lambda) = \{>\} \wedge \{\lambda\} = \emptyset$   
Como hay lambda =>  
i)  $FIRST(>KE') \wedge FOLLOW(E') = \{>\} \wedge \{ , , ; \} = \emptyset$

**16)  $U' \rightarrow \lambda \mid +VU'$**

- a)  $FIRST(\lambda) \wedge FIRST(+VU') = \{\lambda\} \wedge \{+\} = \emptyset$   
Como hay lambda =>  
i)  $FIRST(+VU') \wedge FOLLOW(U') = \{+\} \wedge \{ > , , ; \} = \emptyset$

**17)  $V \rightarrow idJ \mid \text{cte-entera} \mid \text{cadena} \mid (E)$**

- a)  $FIRST(idJ) \wedge FIRST(\text{cte-entera}) = \{id\} \wedge \{\text{cte-entera}\} = \emptyset$   
b)  $FIRST(id) \wedge FIRST(\text{cadena}) = \{id\} \wedge \{\text{cadena}\} = \emptyset$   
c)  $FIRST(\text{cte-entera}) \wedge FIRST(\text{cadena}) = \emptyset$

**18)  $J \rightarrow (L) \mid \lambda$**

- a)  $FIRST((L)) \wedge FIRST(\lambda) = \{ ( \} \wedge \{\lambda\} = \emptyset$   
Como hay lambda =>  
i)  $FIRST((L)) \wedge FOLLOW(J) = \{ ( \} \wedge \{ + , > , , ; \} = \emptyset$

Tabla

	var	id	;	int	string	boolean	function	(	)	{	}
P	P -> DP	P -> FP					P -> FP				
D	D -> var T id ;										
T				T -> int	T -> string	T -> boolean					
F							F -> function T' id( A ) { C }				
T'		T' -> $\Lambda$		T' -> T	T' -> T	T' -> T					
A				A -> T id A'	A -> T id A'	A -> T id A'				A -> $\Lambda$	
A'										A' -> $\Lambda$	
C	C -> DC	C -> S C									C -> $\Lambda$
S		S -> id O									
O								O -> ( L ) ;			
L		L -> E L'						L -> E L'		L -> $\Lambda$	
L'										L' -> $\Lambda$	
X		X -> E	X -> $\Lambda$				X -> E				
SC											
B		B -> id=E	B -> $\Lambda$								
B'		B' -> idB''								B' -> $\Lambda$	
B''											
E		E -> KE'						E -> KE'			
E'			E' -> $\Lambda$							E' -> $\Lambda$	
K		K -> U					K -> U				
U		U -> VU'					U -> VU'				
U'			U' -> $\Lambda$							U' -> $\Lambda$	
V		V -> idJ					V -> (E)				
J			J -> $\Lambda$				J -> (L)			J -> $\Lambda$	

	.	if	print	input	return	=	=	for	>	!	+	cte_entera	cadena	\$
P		P -> FP	P -> FP	P -> FP	P -> FP			P -> SC F						P -> $\Lambda$
D														
T														
F														
T'														
A														
A'	A' -> , T id A'													
C		C -> S C	C -> S C	C -> S C	C -> S C			C -> SC						
S		S -> if ( E ) S	S -> print ( E ) ;	S -> input ( E ) ;	S -> return X ;									
O						O -> =E ;	O ->  = E ;							
L										L -> E L'		L -> E L'	L -> E L'	
L'	L' -> , id L'													
X										X -> E		X -> E	X -> E	
SC								SC -> For(B;E;B')(C)						
B														
B'														
B''						B'' -> =E	B'' ->  = E							
E										E -> KE'		E -> KE'	E -> KE'	
E'									E' -> >KE'					
K										K -> IK		K -> U	K -> U	
U												U -> VU'	U -> VU'	
U'									U' -> $\Lambda$		U' -> +VU'			
V												V -> cte_entera	V -> cte_entera	
J									J -> $\Lambda$		J -> $\Lambda$			

# Analizador Semántico

## Gramática

$P' \rightarrow P$   
 $P \rightarrow DP$   
 $P \rightarrow FP$   
 $P \rightarrow SP$   
 $P \rightarrow \underline{SC} P$   
 $P \rightarrow \lambda$   
 $D \rightarrow \text{var } T \text{ id } ;$   
 $T \rightarrow \text{int}$   
 $T \rightarrow \text{string}$   
 $T \rightarrow \text{boolean}$   
 $F \rightarrow \text{function } T' \text{ id } ( A ) \{ C \}$   
 $T' \rightarrow T$   
 $T' \rightarrow \lambda$   
 $A \rightarrow \lambda$   
 $A \rightarrow T \text{ id } A'$   
 $A' \rightarrow \lambda$   
 $A' \rightarrow , T \text{ id } A'$   
 $C \rightarrow DC$   
 $C \rightarrow SC$   
 $C \rightarrow \lambda$   
 $C \rightarrow \underline{SC} C$   
 $S \rightarrow \text{if}(E) S$   
 $S \rightarrow \text{idO}$   
 $S \rightarrow \text{print}(E);$   
 $S \rightarrow \text{input}(E);$   
 $S \rightarrow \text{return } X;$   
 $O \rightarrow = E;$   
 $O \rightarrow |= E;$   
 $O \rightarrow (L);$   
 $L \rightarrow \lambda$   
 $L \rightarrow E L'$   
 $L' \rightarrow \lambda$   
 $L' \rightarrow , E L'$   
 $X \rightarrow E$   
 $X \rightarrow \lambda$   
 $\underline{SC} \rightarrow \text{for}( B ; E ; B' ) \{ C \}$   
 $B \rightarrow \text{id} = E$   
 $B \rightarrow \lambda$   
 $B' \rightarrow \text{id } B''$   
 $B' \rightarrow \lambda$   
 $B'' \rightarrow = E$   
 $B'' \rightarrow |= E$   
 $E \rightarrow KE'$   
 $E' \rightarrow \lambda$   
 $E' \rightarrow > KE'$   
 $K \rightarrow ! K$   
 $K \rightarrow U$   
 $U \rightarrow VU'$   
 $U' \rightarrow \lambda$



$U' \rightarrow +VU'$   
 $V \rightarrow idJ$   
 $V \rightarrow cte-entera$   
 $V \rightarrow cadena$   
 $V \rightarrow (E)$   
 $J \rightarrow (L)$   
 $J \rightarrow \lambda$

TSG = Tabla de símbolos Global

TSL = Tabla de símbolos Local

Desp = desplazamiento

Desp\_G = Desplazamiento Global

Desp\_L = desplazamiento Local

Crear\_TS() = Crea una Tabla de Símbolos

Borra\_TS() = Borra la Tabla de Símbolos indicada

Insertar\_TS() = añade una entrada a la Tabla de símbolos

busca\_TS() = busca una entrada en la tabla de símbolos

Decl = variable binaria que indica si se esta o no en zona de declaración

Tabla\_actual = indica cual es la tabla que se esta usando en este instante

```

P' -> {TSG=crear_TS(); desp_G = Ø; T_actual=TSG; desp=desp_G } P { Borra_TS(TSG)}
P -> DP
P -> FP
P -> SP
P -> SC P
P -> λ
D -> {Decl=true} var T id {insertar_TS(id.posTS, T.tipo, desp) desp= desp+T.ancho, Decl=false};
T -> int      {T.tipo=ent      T.ancho=1}
T -> string   {T.tipo=cadena  T.ancho=64}
T -> Boolean  {T.tipo=log      T.ancho=1}

F -> function T' {Decl=true} id {TSL=crearTS(); desp_L=Ø; Tabla_actual=TSL; desp=despl_L } ( A
{decl=false} ) {inserer_tipo_TS(id.posTS,<tipo>=function, <tiporetorno>=T'.tipo, A.TIPO-
>T'.tipo, -) inserer_etiqueta(id.posTS,netiq())} { C } {If C.tipo ≠ tipo-error Then
      {If (C.tiporet==ok || C.tiporet==vacio) && T'.tipo==vacio Then
        Tabla_actual = TSG; desp= desp_G; Borra_TS(TSL)
      Else If C.tiporet ≠ T'.tipo then error("Tipo erroneo en return")
      Else Tabla_actual = TSG; desp= desp_G; Borra_TS(TSL)}
    Else error("Error en el cuerpo de funcion")}

T' -> T {T'.tipo =T.tipo, T'.ancho=T.ancho}
T' -> λ {T'.tipo = vacio}
A -> λ {A.tipo = vacio}

```

```

A -> T id A' {inserer_tipo_TS(id.posTS, T.tipo, desp); desp=desp+T.ancho; A.tipo=T.tipo x
A'.tipo}
A' -> λ {A'.tipo = vacio}
A -> ,T id A' {inserer_tipo_TS(id.posTS, T.tipo, desp); desp=desp+T.ancho; A.tipo=T.tipo x
A'.tipo}

C -> DC1 {If C1.tipo ≠ tipo-error Then C.tiporet=C1.tiporet; C.tipo=C1.tipo}
C -> SC1 {If (C1.tipo && S.tipo) ≠ tipo-error Then
    {If S.tiporet == vacio Then C.tiporet = C1.tiporet; C.tipo=ok
    Else if C1.tiporet == vacio Then C.tiporet = S.tiporet; C.tipo=ok
    Else if S.tiporet == C1.tiporet Then C.tiporet = S.tiporet; C.tipo=ok
    Else C.tiporet = error("Returns con tipo diferente"); C.tipo=error("Returns con
    tipo diferente")}}
    Else C.tipo=error("Error en cuerpo"); C.tiporet=error("Error en cuerpo")}}
C -> λ {C.tiporet=vacio; C.tipo=ok}
C -> SC C1 { If (C1.tipo && SC.tipo) ≠ tipo-error Then
    {If SC.tiporet == vacio Then C.tiporet = C1.tiporet; C.tipo=ok
    Else if C1.tiporet == vacio Then C.tiporet = SC.tiporet; C.tipo=ok
    Else if SC.tiporet == C1.tiporet Then C.tiporet = SC.tiporet; C.tipo=ok
    Else SC.tiporet = error("Returns con tipo diferente") ; C.tipo=error("Returns
    con tipo diferente")}}
    Else C.tipo=error("Error en cuerpo"); C.tiporet=error("Error en cuerpo")}}

S -> if(E) S1 {If S1.tipo==ok && E.tipo==log Then S.tipo = ok
    Else S.tipo=error("Error en el if");
    S.tiporet=S1.tiporet}
S -> id O {If busca_Tipo_TS(id.pos)== O.tipo Then S.tipo = ok
    Else busca_TS(id.pos).tipoParam[]==O.tipo Then S.tipo = ok
    Else S.tipo=error("Tipo incorrecto en asignacion");
    S.tiporet=ok }
S -> print(E); {If E.tipo==log Then S.tipo=error("Variable logica en print")
    Else S.tipo = ok;
    S.tiporet=ok}
S -> input(E); { If E.tipo==log Then S.tipo=error("Variable logica en input")
    Else S.tipo = ok;
    S.tiporet=ok}
S -> return X; {If X.tipo ≠ tipo-error Then S.tipo=ok
    Else S.tipo=error("Tipo no valido para el return");
    S.tiporet=X.tipo }

O -> = E; {O.tipo=E.tipo}
O -> |= E; {If E.tipo == log Then O.tipo = log
    Else O.tipo = error("Tipo no lógico en asignación con or logico") }

O -> (L); {O.tipo=L.tipo}

L -> λ {L.tipo=vacio}
L -> E L' { If E.tipo== {ent,cadena,log} Then L.tipo= E.tipo x L'.tipo
    Else L.tipo=error("...") }
L' -> λ {L'.tipo=vacio}
L' -> ,E L'1 { If E.tipo== {ent,cadena,log} Then L'.tipo= E.tipo x L'1.tipo

```

```

Else L'.tipo=error("...")

X -> E {If E.tipo==ent || E.tipo==cadena || E.tipo==log Then X.tipo=E.tipo
      Else X.tipo=Error("Tipo no entero o cadena o lógico") }

X -> λ {X.tipo = vacio}

SC -> for( B; E; B' ) { C } {If (B.tipo ≠ tipo-error && E.tipo == log && B'.tipo ≠ tipo-error &&
SC.tipo ≠ tipo-error) Then SC.tiporet = C.tiporet; SC.tipo=C.tipo
Else SC.tipo=error("Error en el for") }

B-> id = E { If busca_Tipo_TS(id.pos) == E.tipo Then B.tipo=ok
            Else B.tipo=error("Asignación de tipos distintos en la iniciacion del for")}
B -> λ {B.tipo=ok}
B' -> id B'' {If busca_Tipo_TS(id.pos) == B''.tipo Then B'.tipo=ok
            Else B'.tipo=error("Asignación de tipos distintos en la actualización del for")}
B' -> λ {B'.tipo=vacio}
B'' -> = E {B''.tipo=E.tipo}
B'' -> | = E {If E.tipo == log Then B''.tipo = log
            Else B''.tipo = error("Tipo no lógico en la actualización del for") }
E -> KE' {If E' == ok Then {If K.tipo=={ent,log,cadena} Then E.tipo = K.tipo
                        Else E.tipo=error("Tipo no valido")}
        Else If K.tipo == E'.tipo== ent Then E.tipo=ent
        Else E.tipo=error("Tipo no valido")}
E' -> λ {E'.tipo = ok}
E' -> >KE'₁ {If E'₁== ok Then {If K.tipo==ent Then E'.tipo = K.tipo
                        Else E'.tipo=error("Tipo no valido")}
        Else If K.tipo == E'₁.tipo==ent Then E'.tipo=ent
        Else E'.tipo=error("Tipo no valido")}

K -> ! K₁ { If K₁.tipo==log Then K.tipo=log
            Else K.tipo=Error("Tipo no logico")}
K -> U {K.tipo=U.tipo}
U -> VU' { If U'== ok Then {If V.tipo == {ent,cadena,log} Then U.tipo = V.tipo
                        Else U.tipo=error("Tipo no valido")}
        Else If V.tipo == U'.tipo == ent Then U.tipo=ent
        Else U.tipo=error("Tipo no valido")}
U' -> λ {U'.tipo=ok}
U' -> +VU'₁ {If U'₁== ok Then {If V.tipo=ent Then U'.tipo = V.tipo
                        Else U'.tipo = error("Suma con tipo distinto a entero")}
        Else If V.tipo == U'₁.tipo == ent Then U'.tipo=ent
        Else U'.tipo=error("Suma con tipo distinto a entero")}

V -> idJ {If busca_TS(id.pos).tiporetorno ==vacio Then V.tipo=error("La función no puede
devolver vacio")
        Else If busca_TS(id.pos).N_param ≠ J.Tipo.Tam Then V.tipo=error("Numero de
parámetro equivocados")
        Else If Busca_TS(id.pos).tipoParam[] ≠ J.tipo Then V.tipo=error("Tipo de atributos
equivocados")
        Else V.tipo= busca_TS(id.pos).tiporetorno}
V -> cte-entera {T.tipo=ent}

```

$V \rightarrow \text{cadena} \quad \{T.tipo=cadena\}$   
 $V \rightarrow (E) \quad \{V.tipo= E.tipo\}$   
 $J \rightarrow (L) \quad \{J.tipo= L.tipo\}$   
 $J \rightarrow \lambda \quad \{J.tipo=vacio\}$

## Tabla de Símbolos Final

La tabla de símbolos se usa para almacenar la información de los identificadores. Se usa para almacenar y consultar el contenido.

En la tabla se almacena múltiple información:

Lexema: Caracteres que representan el lexema del código fuente.

Tipo de dato: puede ser ent, log, cadena o funcion

Desplazamiento

N argumentos, en el caso de que sea una función

Tipo de los argumentos, en el caso de que tenga parámetros.

Tipo retorno, en el caso que sea una función y que devuelva algún dato

Etiqueta: <<et>>+<<Nombre\_Funcion>>

La tabla de símbolos se estructura en una tabla global y en locales. Las locales se usan para funciones.

Con respecto al primer diseño que hicimos en el analizador léxico, se añadiría el dato de etiqueta que se usaría en las funciones.

Además el dato entero pasaría a llamarse ent, el boolean seria log y el string seria cadena

Ejemplo:

```

Var a124 int;
var boolean b;
function int SumaAlCuadrado (int ca5e7, int ee78) {
...
...
}

```

Tabla Global

Iden.	Lexema	Tipo	Desp	Nº args	Tipos args	Tipo retorno	Etiqueta
1	A124	ent	0				
2	B	log	1				
3	SumaAlCuadrado	funcion		2	ent X ent	ent	etSumaAlCuadrado

Tabla local

Iden.	Lexema	Tipo	Desp	Nº args	Tipos args	Tipo retorno	Etiqueta
1	ca5e7	ent	0				
2	ee78	ent	1				

## Anexo

Casos de prueba:

Primeros 3 casos son los que no generan errores y los tres últimos los incorrectos.

/ Caso N1 /

```

var int z;
var string a;
var boolean b;
function test(int i){
    for(i = 0; i > 10; ){
        if( (i+5) > 10)
            print("es mayor de 5");
        i = i + 1;
    }
}
z=0;
test(z);

```

**Tokens:**

```

<var, >
<int, >
<id,1>
<puntoComa, >
<function, >
<id,2>
<parAb, >
<int, >
<id,3>
<parCerr, >
<corchAb, >
<for, >
<parAb, >
<id,3>
<asignacion, >
<cte_entera,0>

```

```

<puntoComa, >
<id,3>
<opRel, >
<cte_entera,10>
<puntoComa, >
<parCerr, >
<corchAb, >
<if, >
<parAb, >
<parAb, >
<id,3>
<opArt, >
<cte_entera,5>
<parCerr, >
<opRel, >
<cte_entera,10>
<parCerr, >
<print, >
<parAb, >
<cadena,"es mayor de 5">
<parCerr, >
<puntoComa, >
<id,3>
<asignacion, >
<id,3>
<opArt, >
<cte_entera,1>
<puntoComa, >
<corchCerr, >
<corchCerr, >
<id,1>
<asignacion, >
<cte_entera,0>
<puntoComa, >
<id,2>
<parAb, >
<id,1>
<parCerr, >
<puntoComa, >

```

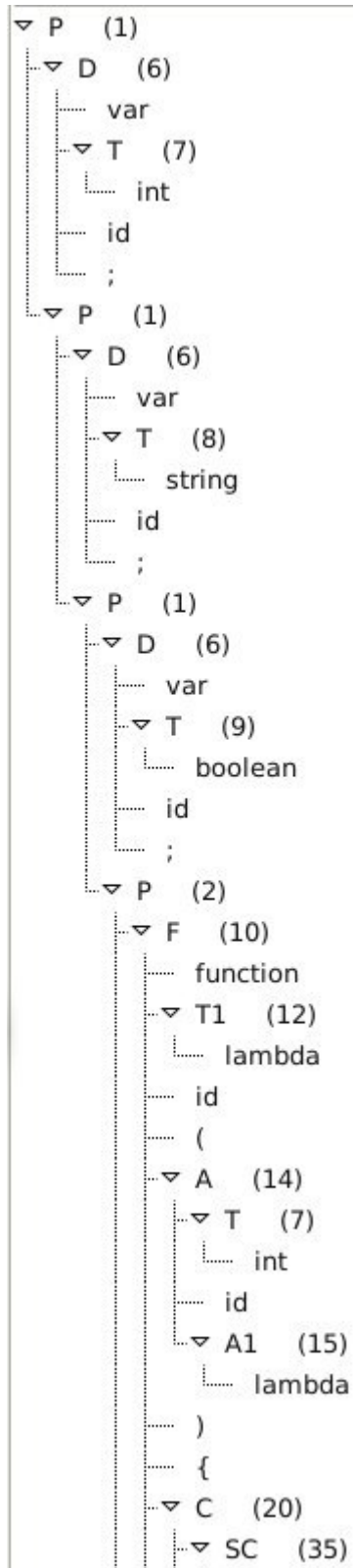
### Parser

```

D 1 6 7 2 10 12 14 7 15 20 35 36 42 46 47 51 48 43 42 46 47 50 55 48 44 46 47 51 48 43 39 18
21 42 46 47 53 42 46 47 50 55 49 51 48 43 48 44 46 47 51 48 43 23 42 46 47 52 48 43 18 22 26
42 46 47 50 55 49 51 48 43 19 19 3 22 26 42 46 47 51 48 43 3 22 28 30 31 5

```

## Árbol Sintáctico



```

for
(
  ▾ B (36)
    id
    =
    ▾ E (42)
      ▾ K (46)
        ▾ U (47)
          ▾ V (51)
            cte_entera
          ▾ U1 (48)
            lambda
        ▾ E1 (43)
          lambda
      ;
    ▾ E (42)
      ▾ K (46)
        ▾ U (47)
          ▾ V (50)
            id
            ▾ J (55)
              lambda
          ▾ U1 (48)
            lambda
        ▾ E1 (44)
          >
          ▾ K (46)
            ▾ U (47)
              ▾ V (51)
                cte_entera
              ▾ U1 (48)
                lambda
          ▾ E1 (43)
            lambda

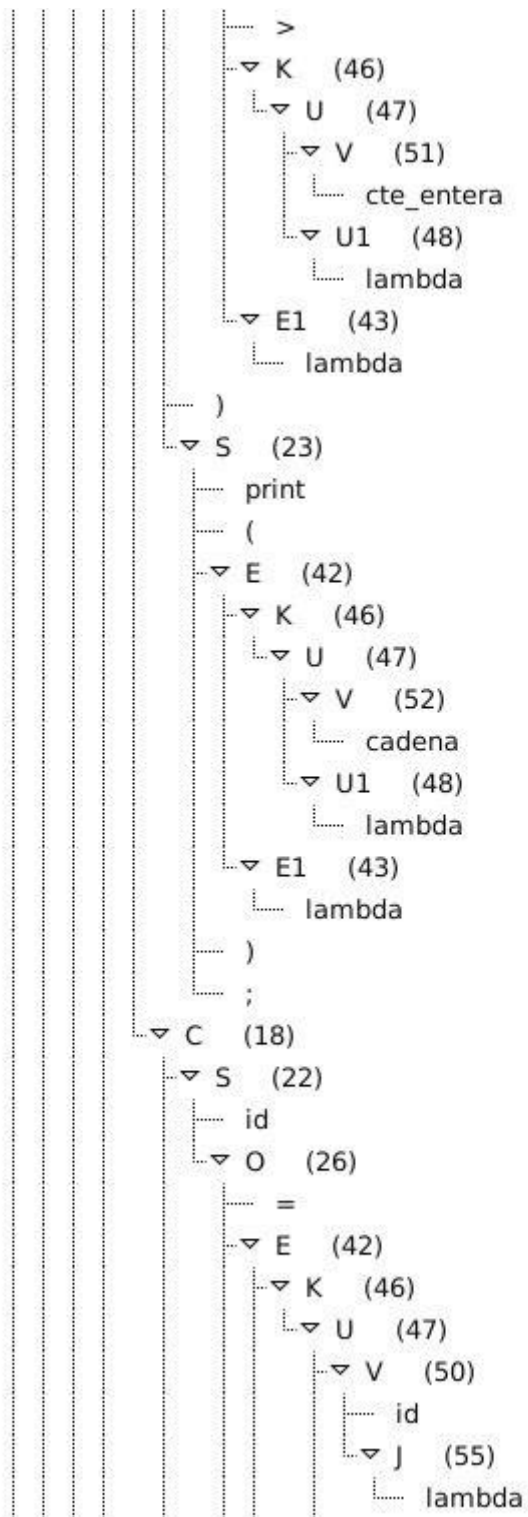
```



```

;
└─ B1 (39)
   └─ lambda
      )
      {
      └─ C (18)
         └─ S (21)
            └─ if
               (
               └─ E (42)
                  └─ K (46)
                     └─ U (47)
                        └─ V (53)
                           (
                           └─ E (42)
                              └─ K (46)
                                 └─ U (47)
                                    └─ V (50)
                                       └─ id
                                          └─ J (55)
                                             lambda
                                             └─ U1 (49)
                                                +
                                                └─ V (51)
                                                   cte_entera
                                                   └─ U1 (48)
                                                      lambda
                                                      └─ E1 (43)
                                                         lambda
                                                         )
                                                         └─ U1 (48)
                                                            lambda
                                                            └─ E1 (44)

```





## Tabla de símbolos

#1:

\*'z'

+ tipo:'ent'

+ despl:0

\*'test'

+ tipo:'funcion'

+ numParam:1

+ TipoParam1:'ent'

+ TipoRetorno:

+ EtiqFuncion:'ETtest'

#2:

\*'i'

+ tipo:'ent'

+ despl:0

/ Caso N2 /

```
var boolean acierto;
```

```
var int intro;
```

```
print("Elige un numero del 0 al 9:");
```

```
function boolean sorpresa(boolean aux){
```

```
    print("Has acertado!");
```

```
    aux |= acierto;
```

```
    return aux;
```

```
}
```

/ Caso N3 /

```
print("Introducir número:");
```

```
var int a;
```

```
var string cad;
```

```
cad = "Enhorabuena! Tu número es mayor que 5!";
```

```
input(a);
```

```
if (a>5)
```

```
    print(cad);
```

/ Caso N4 /

```
var string a;
```

```
print("Introduce tu nombre: ");
```

```

input(a);
var int b;
print("Introduce tu edad");
input(b);
if(b>18)
    print("Acceso permitido");

```

/ Caso N5 /

```

var int a;
print("Introduce un numero menor de 50");
input(a);
for( ;50>a;a=a+1){
    print(a);
}

```

/ Caso N6 /

Suma de tipos erroneos

```

var string a;
a = "hola"
var int b;
b = 5;
return a + b;

```

Mensaje de error:

Exception: La expresion escrita no tiene los tipos correctos

/ Caso N7 /

Numero de parámetros de función erroneo

```

var int a;
a = 2;
var int b;
b = 3;
function int suma (int num1, int num2) {
    return num1+num2;
}
var int d;
d = 3;
var int c;
c= suma(a,b,d);

```

Mensaje de error:

Exception: Numero de parametros equivocado

/ Caso N8 /

Tipos de parámetros en función erróneos

```
var string a;  
a = "dos";  
var int b;  
b = 3;  
function int suma (int num1, int num2) {  
    return num1+num2;  
}  
var int c;  
c= suma(a,b);
```

Mensaje de error:

Exception: Tipos de los atributos equivocado

/ Caso N9 /

Inicialización del For erróneos

```
Var int argc;  
argc = 3;  
var int res;  
res =0;  
var int a;  
a = 3;  
var int i;  
for(i = "hola"; i>argc;){  
    res = res + a;  
    i = i + 1;  
}
```

Mensaje de error:

Exception: Asignacion de tipos distintos en el for

/ Caso N10 /

El error se produce al intentar asignar un 2 a un string

```
var string a;  
a = 2;  
var string b;  
b = 3;
```

```
function int suma (int num1, int num2) {  
    return num1+num2;  
}  
var int c;  
c= suma(a,b);
```

Mensaje de error:

Exception: Tipo incorrecto en asignacion