




8 DE ENERO DE 2024

*PRÁCTICA DE PROCESADORES DE
LENGUAJES*

GRUPO 53

*MANUEL PÉREZ REDONDO, C200382
JUAN FRANCISCO CASANOVA FERRER, B190340
ANDREA LOZANO OROZCO, Y160260*



Índice

| | |
|----------------------------------|---|
| Diseño del Analizador Léxico | 3 |
| Tokens | 3 |
| Gramática | 4 |
| Autómata Finito Determinista | 5 |
| Acciones Semánticas y Errores | 6 |
| Diseño de la Tabla de Símbolos | 7 |
| Matriz de Transición | 7 |
| Diseño Analizador Sintáctico | |
| Gramática | |
| Tablas de Acción y Goto | |
| Diseño Analizador Semántico | |
| Índice de Anexos | |

Diseño del Analizador Léxico

TOKENS

1. <CteEntera, valor>
2. <Suma, - >
3. <Negación, - >
4. <Comparador, - >
5. <palRes, lexema>
 - 'boolean'
 - 'function'
 - 'get'
 - 'if'
 - 'int'
 - 'let'
 - 'put'
 - 'return'
 - 'string'
 - 'void'
 - 'while'
6. <Asignación, ->
7. <AsignaciónSuma, - >
8. <Cadena, lexema>
9. <id, posTS>
10. <coma, ->
11. <puntocomma, ->
12. <abroparentesis, ->
13. <cierroparentesis, ->
14. <abrocorchetes, ->
15. <cierrocorchetes, ->

En un primer intento decidimos que las palabras reservadas nos devolvieran un código numérico representando la posición en la tabla de palabras reservadas. Pero por comodidad en nuestro código, al final decidimos que devolvieran un lexema con la propia palabra reservada.

Además, para los símbolos decidimos en un inicio porque, devolvieran su valor hash, tras las correcciones del documento cada símbolo tiene su propio token

asignado.

Al igual que para los operadores y las asignaciones, decidimos finalmente crear un token para cada uno.

GRAMÁTICA

- del: {espacio, tabulador}
- l: {a, b, ..., z}
- a: {A, B, ..., Z}
- d: {0, 1, ..., 9}
- f: {eof, eol}

Además los siguientes símbolos

- ,
- ;
- (
-)
- {
- }
- –
- +
- =
- !
- /

- cd: cualquier carácter – d
- cc: cualquier carácter – f
- oc: cualquier otro carácter menos los usados en el estado anterior

$S \rightarrow \text{del } S \mid l A \mid a A \mid d B \mid / C \mid " D \mid , \mid ; \mid (\mid) \mid \{ \mid \} \mid + E \mid = F \mid ! \mid _ A$

$A \rightarrow l A \mid d A \mid a A \mid _ A \mid \text{lambda}$

$B \rightarrow d B$

$C \rightarrow / C'$

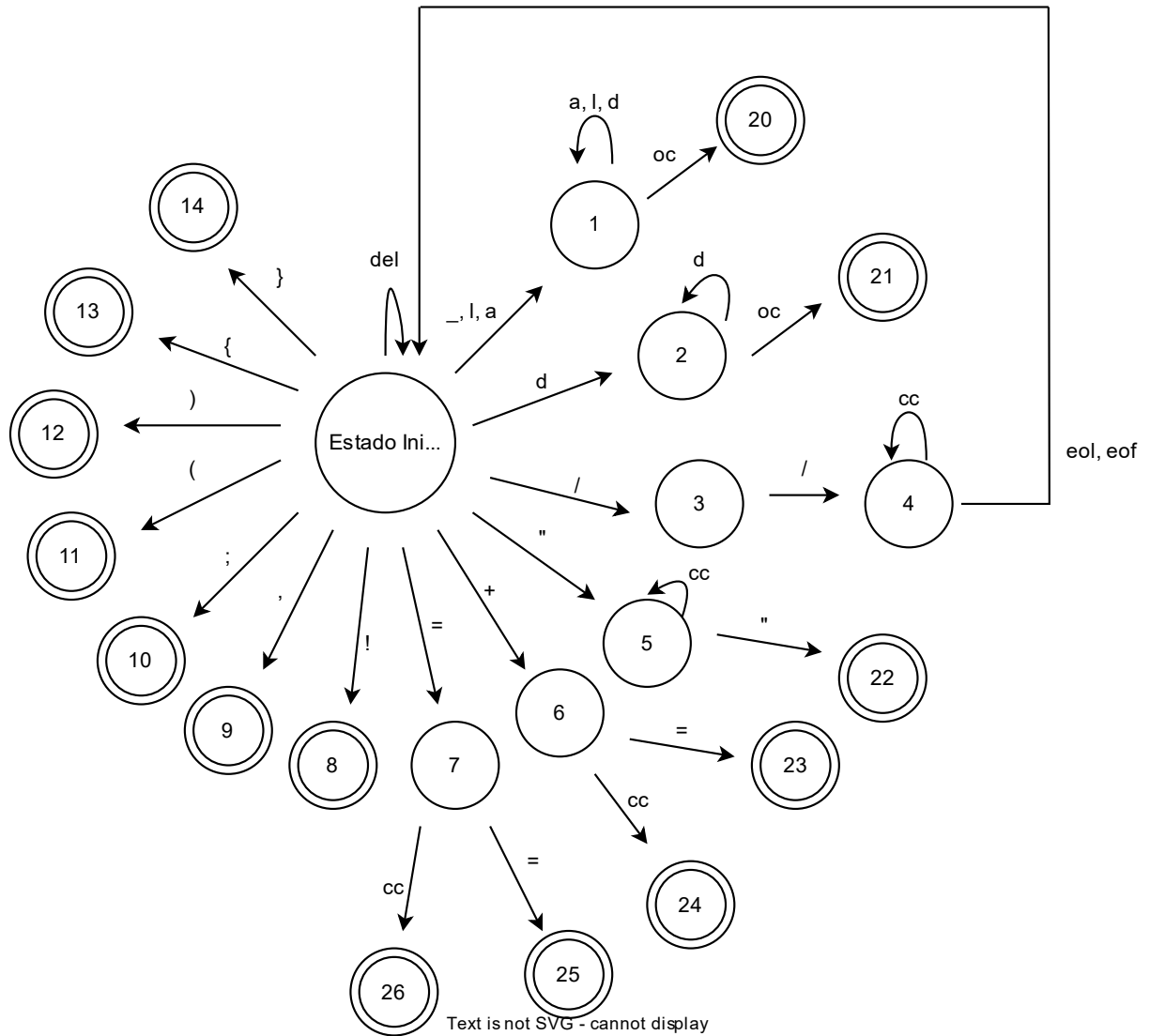
$C' \rightarrow (\text{todo}) C' \mid \text{eof } S$

$D \rightarrow \text{cc } D \mid "$

$E \rightarrow = \mid \text{cc}$

$F \rightarrow = \mid \text{cc}$

AUTÓMATA FINITO DETERMINISTA



ACCIONES SEMÁNTICAS Y ERRORES

Todos los caracteres no contemplados en los estados son errores.

0:0; Leer; cont1 = 0 → A

0:1; lexema = l; Leer → B

1:1; lexema = lexema \oplus l; Leer

IF cont1 > 64 THEN error ("Se ha superado la longitud de la cadena",
línea, lexema) → C

1:20; IF lexema ∈ TSactiva | TSglobal THEN error ("El lexema pertenece a la
TS", línea,
lexema)

ELSE IF lexema ∈ Tabla_palRes THEN GenToken(palRes, lexema)

ELSE GenToken(id, posTS) → D

0:2; valor = valor_ascii(d); Leer → E

2:2; valor = valor*10 + valor_ascii(d); Leer

IF valor > 32767 THEN error ("Supera el límite máximo de
representación de
números enteros") → F

2:21; GenToken(CteEntera, valor) → G

0:3; Leer → H

3:4; Leer

4:4; Leer

4:0; Leer

0:5; lexema = l; Leer; cont2 = 0 → I

5:5; lexema = lexema \oplus l; Leer; cont2++

IF cont2 > 64 THEN error ("Supera la longitud máxima de una cadena",
línea,
lexema) → J

5:22; lexema = lexema \oplus l; GenToken(cadena, lexema); Leer → K

0:6; Leer → L

6:23 lexema = lexema \oplus l; GenToken(asignaciónSuma, lexema); Leer
→ P

6:24; GenToken(suma, lexema) → Q

0:7; Leer

7:25; GenToken(==, -); Leer

7:26;; GenToken(asignación, lexema) → O

0:8; GenToken(negación, lexema); Leer → M

0:9; GenToken(coma, -); Leer → N

0:10; GenToke(puntocomma, -); Leer → Ñ

0:11; GenToken(abroparentesis,-); Leer→P
0:12; GenToken(cierroparentesis,-); Leer→R
0:13; GenToken(abrocorchete,-); Leer→S
0:14; GenToken(cierrocorchete,-); Leer→T

Diseño de la Tabla de Símbolos

MATRIZ DE TRANSICIÓN

| | I | a | d | del | , | ; | (|) | { | } | c | - | ! | = | + | / | " | eol | | | | | | | | | | | | | | | | | | |
|------|-----|----|-----|-----|-----|---|---|-----|-----|---|-----|---|-----|---|-----|---|-----|-----|-----|----|----|---|-----|----|-----|----|-----|---|-----|---|-----|----|-----|----|-----|-----|
| -->0 | 1 | B | 1 | B | 2 | E | 0 | A | 13 | L | 20 | R | 21 | S | 22 | T | 23 | U | 24 | V | 30 | 1 | B | 14 | M | 7 | B | 6 | B | 3 | H | 5 | I | 31 | | |
| 1 | 1 | C | 1 | C | 1 | C | | 32 | 33 | | 34 | | 35 | | 36 | | 37 | | 38 | 11 | D | 1 | C | | 39 | | 40 | | 41 | | 42 | | 43 | 44 | | |
| 2 | | 45 | 46 | 2 | F | | | 47 | 48 | | 49 | | 50 | | 51 | | 52 | | 53 | 12 | G | | 54 | | 55 | | 56 | | 57 | | 58 | | 59 | 60 | | |
| 3 | | 61 | 62 | | 63 | | | 64 | 65 | | 66 | | 67 | | 68 | | 69 | | 70 | | 71 | | 72 | | 73 | | 74 | | 75 | 4 | H | | 76 | 77 | | |
| 4 | 78 | | 79 | | 80 | | | 81 | 82 | | 83 | | 84 | | 85 | | 86 | | 87 | 4 | H | | 88 | | 89 | | 90 | | 91 | | 92 | | 93 | 0 | H | 109 |
| 5 | 94 | | 95 | | 96 | | | 97 | 98 | | 99 | | 100 | | 101 | | 102 | | 103 | 5 | J | | 104 | | 105 | | 106 | | 107 | | 108 | 15 | K | | 109 | |
| 6 | 110 | | 111 | | 112 | | | 113 | 114 | | 115 | | 116 | | 117 | | 118 | | 119 | 17 | Q | | 120 | | 121 | 16 | P | | 122 | | 123 | | 124 | | 125 | |
| 7 | 126 | | 127 | | 128 | | | 129 | 130 | | 131 | | 132 | | 133 | | 134 | | 135 | 18 | O | | 136 | | 137 | 19 | N | | 138 | | 139 | | 140 | | 141 | |
| 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 25 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 27 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 28 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 29 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Acciones semánticas y su código de letra:

- Leer; cont1 = 0 → A
- Lexema = l; Leer → B
- Lexema = lexema \oplus l; Leer
If cont1 > 64 then error (50) → C
- if lexema ∈ TSactiva | TSglobal then error (51)
else if lexema ∈ Tabla_palRes then GenToken(palRes, lexema)
else GenToken(id, posTS) → D
- valor = valor_ascii(d); Leer → E
- valor = valor*10 + valor_ascii(d); Leer
if valor > 32767 then error (52) → F
- GenToken(CteEntera, valor) → G
- Leer → H
- Lexema = l; Leer, cont2 = 0 → I
- Lexema = lexema \oplus l; Leer; ccont2++
If cont2 > 64 then error (53) → J
- lexema = lexema \oplus l; GenToken(cadena, lexema); Leer → K
- GenToken(símbolo, hash(s)); Leer → L
- GenToken(negación, lexema); Leer → M
- lexema = lexema \oplus l; GenToken(comparador, lexema); Leer → N
- GenToken(asignación, lexema) → O
- lexema = lexema \oplus l; GenToken(asignaciónSuma, lexema); Leer → P
- GenToken(suma, lexema) → Q

Leyenda numérica de errores:

- 50 → Se ha superado la longitud de la cadena
- 51 → El lexema pertenece a la TS
- 52 → Supera el límite máximo de representación de números enteros
- 53 → Supera la longitud máxima de una cadena
- El resto de los errores (30-108) son errores debido a la acción de que todos los caracteres no contemplados en los estados son errores.

Para la implementación de la Tabla de Símbolos hemos usado una hashtable. Los principales motivos han sido, entre otros, la facilidad de uso y la rápida recuperación de los valores dado el atributo. En un futuro haremos una estructura que nos permita tener la Tabla de Símbolos global y la activa de forma simultánea para su perfecto funcionamiento.

En la Tabla de Símbolos, las claves son la propia posición en la tabla y, como valor, es el lexema a introducir. Cabe decir que en futuras entregas se realizarán cambios para que se puedan consultar los atributos de un lexema determinado y no solo su nombre.

Diseño del Analizador Sintáctico Ascendente LR

GRAMÁTICA

En nuestro caso nos ha tocado el Analizador Sintáctico Ascendente, por lo que no nos afecta que la gramática sea recursiva por la izquierda. Además, no tenemos que comprobar la condición LL1.

En el primer paso para la gramática de nuestro Analizador Sintáctico debemos aumentar la gramática. Como nuestro axioma es P, vamos a añadir la regla $P' \rightarrow P$.

Por lo que la gramática quedaría de la siguiente manera:

$P' \rightarrow P$
 $P \rightarrow BP \mid FP \mid \lambda$
 $E \rightarrow E == U \mid U$
 $U \rightarrow U + V \mid V$
 $V \rightarrow ! X \mid X$
 $X \rightarrow id \mid (E) \mid id (L) \mid ent \mid cad$
 $S \rightarrow id = E; \mid id (L); \mid put E; \mid get id; \mid return Z; \mid id += E;$
 $L \rightarrow EQ \mid \lambda$
 $Q \rightarrow , EQ \mid \lambda$
 $Z \rightarrow E \mid \lambda$
 $B \rightarrow if (E) S \mid if (E) \{ S \} \mid while (E) S \mid while (E) \{ S \} \mid let id T; \mid let id T = E; \mid S$
 $T \rightarrow int \mid boolean \mid string$
 $F \rightarrow F' \{ C \}$
 $F' \rightarrow F'' (A)$
 $F'' \rightarrow function id H$
 $H \rightarrow T \mid void$
 $A \rightarrow T id K \mid void$
 $K \rightarrow , T id K \mid \lambda$
 $C \rightarrow BC \mid \lambda$

Para la tabla de ACCIÓN vamos a asignar a cada regla un número para que sea más fácil a la hora de representarlo.

$P' \rightarrow P \Rightarrow 1$
 $P \rightarrow BP \Rightarrow 2$
 $P \rightarrow FP \Rightarrow 3$
 $P \rightarrow \lambda \Rightarrow 4$
 $E \rightarrow E == U \Rightarrow 5$
 $E \rightarrow U \Rightarrow 6$
 $U \rightarrow U + V \Rightarrow 7$
 $U \rightarrow V \Rightarrow 8$
 $V \rightarrow ! X \Rightarrow 9$
 $V \rightarrow X \Rightarrow 10$

$X \rightarrow \text{id} \Rightarrow 11$
 $X \rightarrow (E) \Rightarrow 12$
 $X \rightarrow \text{id} (L) \Rightarrow 13$
 $X \rightarrow \text{ent} \Rightarrow 14$
 $X \rightarrow \text{cad} \Rightarrow 15$
 $S \rightarrow \text{id} = E; \Rightarrow 16$
 $S \rightarrow \text{id} (L); \Rightarrow 17$
 $S \rightarrow \text{put } E; \Rightarrow 18$
 $S \rightarrow \text{get id}; \Rightarrow 19$
 $S \rightarrow \text{return } Z; \Rightarrow 20$
 $S \rightarrow \text{id} += E; \Rightarrow 21$
 $L \rightarrow \text{EQ} \Rightarrow 22$
 $L \rightarrow \lambda \Rightarrow 23$
 $Q \rightarrow , \text{EQ} \Rightarrow 24$
 $Q \rightarrow \lambda \Rightarrow 25$
 $Z \rightarrow E \Rightarrow 26$
 $Z \rightarrow \lambda \Rightarrow 27$
 $B \rightarrow \text{if} (E) S \Rightarrow 28$
 $B \rightarrow \text{if} (E) \{ S \} \Rightarrow 29$
 $B \rightarrow \text{while} (E) S \Rightarrow 30$
 $B \rightarrow \text{while} (E) \{ S \} \Rightarrow 31$
 $B \rightarrow \text{let id } T; \Rightarrow 32$
 $B \rightarrow \text{let id } T = E; \Rightarrow 33$
 $B \rightarrow S \Rightarrow 34$
 $T \rightarrow \text{int} \Rightarrow 35$
 $T \rightarrow \text{boolean} \Rightarrow 36$
 $T \rightarrow \text{string} \Rightarrow 37$
 $F \rightarrow F' \{ C \} \Rightarrow 38$
 $F' \rightarrow F'' (A) \Rightarrow 39$
 $F'' \rightarrow \text{function id } H \Rightarrow 40$
 $H \rightarrow T \Rightarrow 41$
 $H \rightarrow \text{void} \Rightarrow 42$
 $A \rightarrow T \text{ id } K \Rightarrow 43$
 $A \rightarrow \text{void} \Rightarrow 44$
 $K \rightarrow , T \text{ id } K \Rightarrow 45$
 $K \rightarrow \lambda \Rightarrow 46$
 $C \rightarrow BC \Rightarrow 47$
 $C \rightarrow \lambda \Rightarrow 48$

TABLAS LR DE ACCIÓN Y GOTO

[illegible]

Para las tablas de ACCIÓN y GOTO hemos utilizado los estados que nos ha proporcionado la herramienta Bison para nuestra gramática.

La gramática es LR1, solo hay una regla en cada celda y no hay filas vacías.

Diseño del Analizador Semántico

Reglas Semánticas

$P' \rightarrow P \Rightarrow \{ \text{TS_Global} = \text{Crear_TS}(), \text{TS_Global.desp} = 0, \text{TS_Activa} = \text{TS_Global}(), \text{Destruir_TS}() \}$

$P \rightarrow BP \Rightarrow \{ \}$

$P \rightarrow FP \Rightarrow \{ \}$

$P \rightarrow \lambda \Rightarrow \{ \}$

$E \rightarrow E == U \Rightarrow \{ \text{E.tipo} := \text{if}(\text{E1.tipo} == \text{U.tipo} = \text{entero}) \text{ then}$
 Tipo.logico
 else
 Error(No coinciden los tipos en la expresión $E == U$)
 }

$E \rightarrow U \Rightarrow \{ \}$

$U \rightarrow U + V \Rightarrow \{ \text{U.tipo} := \text{if}(\text{U1.tipo} == \text{V.tipo} = \text{entero}) \text{ then}$
 Tipo.logico
 else
 Error(No coinciden los tipos de la expresión $U + V$)
 }

$U \rightarrow V \Rightarrow \{ \}$

$V \rightarrow ! W \Rightarrow \{ \text{V.tipo} := \text{if}(\text{W.tipo} == \text{boolean}) \text{ then}$
 W.logico
 Else
 Error(No coinciden los tpos de la expresión $!W$)
 }

$V \rightarrow W \Rightarrow \{ \text{V.tipo} := \text{W.tipo} \}$

$W \rightarrow \text{id} \Rightarrow \{ \text{W.tipo} := \text{buscatipoTS}(\text{id.pos}) \}$

$W \rightarrow (E) \Rightarrow \{ \text{W.tipo} := \text{if}(\text{E.tipo} == \{\text{entero}, \text{boolean}\}) \text{ then}$
 E.tipo
 else
 Error(Tipo de la expresión $W(E)$ incorrecto)
 }

$W \rightarrow \text{id} (L) \Rightarrow \{ \text{W.tipo} := \text{buscatipoTS}(\text{id.pos}) \}$

$W \rightarrow \text{ent} \Rightarrow \{ \text{W.tipo} = \text{entero}, \text{W.ancho} = 2 \}$

$W \rightarrow \text{cad} \Rightarrow \{ \text{W.tipo} = \text{string}, \text{W.ancho} = 128 \}$

$S \rightarrow \text{id} = E; \Rightarrow \{ \text{S.tipo} := \text{if}(\text{buscatipoTS}(\text{id.pos}) == \text{E.tipo}) \text{ then}$
 Tipo.OK
 else
 Error(El tipo del identificador no coincide con la expresion)
 }

```

S → id ( L ); => (S.tipo:= buscatipoTS(id.pos))
S → put E; => { S.tipo:= if(E.tipo =={entero, cadena}) then
                    E.tipo
                else
                    Error(Tipo de la expresión es incorrecto para mostrar)
            }

S → get id; => { S.tipo:=if(buscatipoTS(id.pos)={entero, cadena}) then
                    BuscatipoTS(id.pos)
                else
                    Error(No se puede obtener el identificador, tipo erroneo)
            }

S → return Z; => { S.tipoRetorno:=Z.tipo }

S → id += E; => { S.tipo:=if(buscatipoTS(id.pos) == E.tipo) then
                    Tipo.OK
                else
                    Error(El tipo del identificador no coincide con la expresion)
            }

L → EQ => { }
L → λ => { }
Q → , EQ => { }
Q → λ => { }
Z → E => { }
Z → λ => { }
B → if ( E ) S => { B.tipo:=if(E.tipo=logico) then
                    S.tipo
                Else
                    Error(El tipo de la expresion  evaluada es erroneo) }
B → if ( E ) { S } => => { B.tipo:=if(E.tipo=logico) then
                    S.tipo
                Else
                    Error(El tipo de la expresion  evaluada es erroneo) }

B → while ( E ) S => => { B.tipo:=if(E.tipo=logico) then
                    S.tipo
                Else
                    Error(El tipo de la expresion  evaluada es erroneo) }

B → while ( E ) { S } => => { B.tipo:=if(E.tipo=logico) then
                    S.tipo
                Else
                    Error(El tipo de la expresion  evaluada es erroneo) }

B → let id T; => { B.tipo:=if(asignartipoTS(id.pos, T.tipo), aumentarDespla(T.anchos)) then
                    Tipo.OK }

```



```

B → let id T = E; => { B.tipo:=if(T.tipo == E.tipo)then
                        asignartipoTS(id.pos, T.tipo), aumentarDespla(T.ancho),
                        Tipo.OK
                      else
                        Error(No coinciden los tipos de T y la expresion)
                      }

B → S => { B.tipoRetorno:=S.TipoRetorno }
T → int => => { T.tipo=entero, T.ancho=2}
T → boolean => { T.tipo=boolean, T.ancho=2}
T → string => { T.tipo=string, T.ancho=128}
F → F' { C } => { if(F'.tipo != C.tipoRetorno) then
                  Error(No coinciden los tipos de retorno)
                }
F' → F'' ( A ) => { zona_dec = false, rellenarAtributosid(listaparam), F'.tipo:=F''.tipo }
F'' → function id H => { F''.tipo:=H.tipo, insertarTS(id.pos, Tipo.function), zona_dec = true}
H → T => { H.tipo:=T.tipo }
H → void => { T.tipo=void}

A → T id K => { if(buscatipoTS(id.pos)==T.tipo) then
                insertatipoTS(id.pos, T.tipo), A.tipo=T.tipo
              else
                Error(El tipo del identificador no se correspnde con ninguno que exista)
              }

A → void => { }
K → , T id K => => { if(buscatipoTS(id.pos)==T.tipo) then
                    insertatipoTS(id.pos, T.tipo), A.tipo=T.tipo
                  else
                    Error(El tipo del identificador no se correspnde con ninguno que exista)
                  }

K → λ => { }
C → BC => { }
C → λ => { C.tipo = Tipo.OK}

```

Índice de Anexos

| | |
|---|-------|
| Anexo 1: caso de prueba 1 (correcto) | _____ |
| Anexo 2: caso de prueba 2 (correcto) | _____ |
| Anexo 3: caso de prueba 3 (correcto) | _____ |
| Anexo 4: caso de prueba 4 (correcto) | _____ |
| Anexo 5: caso de prueba 5 (correcto) | _____ |
| Anexo 6: caso de prueba 6 (incorrecto) | _____ |
| Anexo 6: caso de prueba 7 (incorrecto) | _____ |
| Anexo 6: caso de prueba 8 (incorrecto) | _____ |
| Anexo 6: caso de prueba 9 (incorrecto) | _____ |
| Anexo 6: caso de prueba 10 (incorrecto) | _____ |

CASO 1 -> Correcto

```
let x string = lexema;
function nombre1 boolean( string cadena ) {

    let y int= 2;
    if ( y == 4 ) {
        put cadena;
    }
    return (y == 4);
}
```

Fichero de tokens:

```
<PALABRARESERVADA, let>
<ID, 0>
<PALABRARESERVADA, string>
<ASIGNACION, >
<CADENA, "lexema">
<PUNTOCOMA, >
<PALABRARESERVADA, function>
<ID, 1>
<PALABRARESERVADA, boolean>
<ABREPARENTESIS, >
<PALABRARESERVADA, string>
<ID, 0>
<CIERRAPARENTESIS, >
<ABRECORCHETE, >
<PALABRARESERVADA, let>
<ID, 1>
<PALABRARESERVADA, int>
<ASIGNACION, >
<ENTERO, 2>
<PUNTOCOMA, >
<PALABRARESERVADA, if>
<ABREPARENTESIS, >
<ID, 1>
<COMPARADOR, >
<ENTERO, 4>
<CIERRAPARENTESIS, >
<ABRECORCHETE, >
<PALABRARESERVADA, put>
<ID, 0>
<PUNTOCOMA, >
<CIERRACORCHETE, >
<PALABRARESERVADA, return>
<ABREPARENTESIS, >
<ID, 1>
<COMPARADOR, >
```

<ENTERO, 4>
 <CIERRAPARENTESIS, >
 <PUNTOCOMA, >
 <CIERRACORCHETE, >
 <FINDEFICHERO, >

Tabla de Símbolos:

CONTENIDOS DE LA TABLA #1:

* LEXEMA : 'cadena'

Atributos:

+ tipo: 'STRING'

+ despl: 0

* LEXEMA : 'y'

Atributos:

+ tipo: 'INT'

+ despl: 128

CONTENIDOS DE LA TABLA #0:

* LEXEMA : 'x'

Atributos:

+ tipo: 'STRING'

+ despl: 0

* LEXEMA : 'nombre1'

Atributos:

+ tipo: 'FUNCTION'

+ numParam: '1'

+ TipoParam0: 'STRING'

+ ModoParam0: 'VALOR'

+ TipoRetorno: 'BOOLEAN'

+ EtiqFuncion: 'nombre1'

Árbol Sintáctico:

CASO 2 -> Correcto

let x int = 0;

let y int = 3;

```

while (x < y) {
  x+=1;
  put x;
}
  
```

Fichero de tokens:

<PALABRARESERVADA, let>

<ID, 0>

<PALABRARESERVADA, int>

<ASIGNACION, >

```

<ENTERO, 0>
<PUNTOCOMA, >
<PALABRARESERVADA, let>
<ID, 1>
<PALABRARESERVADA, int>
<ASIGNACION, >
<ENTERO, 3>
<PUNTOCOMA, >
<PALABRARESERVADA, while>
<ABREPARENTESIS, >
<ID, 0>
<COMPARADOR, >
<ID, 1>
<CIERRAPARENTESIS, >
<ABRECORCHETE, >
<PALABRARESERVADA, put>
<ID, 0>
<PUNTOCOMA, >
<CIERRACORCHETE, >
<FINDEFICHERO, >

```

Tabla de Símbolos:

CONTENIDOS DE LA TABLA #0:

```

*      LEXEMA      :      'x'
Atributos:
+ tipo: 'INT'
+ despl: 0
*      LEXEMA      :      'y'
Atributos:
+ tipo: 'INT'
+ despl: 2

```

Árbol Sintáctico:

CASO 3 -> Correcto

```

let id boolean=(6==2);
function fun boolean( boolean ejemplo ) {

let aux boolean=ejemplo;
return aux;

}

```

Fichero de tokens:

```

<PALABRARESERVADA, let>
<ID, 0>
<PALABRARESERVADA, boolean>
<ASIGNACION, >
<ABREPARENTESIS, >

```

<ENTERO, 6>
 <COMPARADOR, >
 <ENTERO, 2>
 <CIERRAPARENTESIS, >
 <PUNTOCOMA, >
 <PALABRARESERVADA, function>
 <ID, 1>
 <PALABRARESERVADA, boolean>
 <ABREPARENTESIS, >
 <PALABRARESERVADA, boolean>
 <ID, 0>
 <CIERRAPARENTESIS, >
 <ABRECORCHETE, >
 <PALABRARESERVADA, let>
 <ID, 1>
 <PALABRARESERVADA, boolean>
 <ASIGNACION, >
 <ID, 0>
 <PUNTOCOMA, >
 <PALABRARESERVADA, return>
 <ID, 1>
 <PUNTOCOMA, >
 <CIERRACORCHETE, >
 <FINDEFICHERO, >

Tabla de Símbolos:

CONTENIDOS DE LA TABLA #1:

* LEXEMA : 'ejemplo'

Atributos:

+ tipo: 'BOOLEAN'

+ despl: 0

* LEXEMA : 'aux'

Atributos:

+ tipo: 'BOOLEAN'

+ despl: 2

CONTENIDOS DE LA TABLA #0:

* LEXEMA : 'id'

Atributos:

+ tipo: 'BOOLEAN'

+ despl: 0

* LEXEMA : 'fun'

Atributos:

+ tipo: 'FUNCTION'

+ numParam: '1'

+ TipoParam0: 'BOOLEAN'

+ ModoParam0: 'VALOR'

+ TipoRetorno: 'BOOLEAN'

+ EtiquetaFuncion: 'fun'

Árbol Sintáctico:

CASO 4 -> Correcto

```
let id int;  
let id2 string;
```

```
if(id == 1)  
put id2;
```

```
get id2;
```

Fichero de tokens:

```
<PALABRARESERVADA, let>  
<ID, 0>  
<PALABRARESERVADA, int>  
<PUNTOCOMA, >  
<PALABRARESERVADA, let>  
<ID, 1>  
<PALABRARESERVADA, string>  
<PUNTOCOMA, >  
<PALABRARESERVADA, if>  
<ABREPARENTESIS, >  
<ID, 0>  
<COMPARADOR, >  
<ENTERO, 1>  
<CIERRAPARENTESIS, >  
<PALABRARESERVADA, put>  
<ID, 1>  
<PUNTOCOMA, >  
<PALABRARESERVADA, get>  
<ID, 1>  
<PUNTOCOMA, >  
<FINDEFICHERO, >
```

Tabla de Símbolos:

CONTENIDOS DE LA TABLA #0:

* LEXEMA : 'id'

Atributos:

+ tipo: 'INT'

+ despl: 0

* LEXEMA : 'id2'

Atributos:

+ tipo: 'STRING'

+ despl: 2

Árbol Sintáctico:

CASO 5 -> Correcto

```
function nombre1 int( string cadena ) {  
  
    let y int= 2;  
    if ( y == 4 ) {  
        put cadena;  
    }  
  
    return y;  
  
}
```

Fichero de tokens:

```
<PALABRARESERVADA, function>  
<ID, 0>  
<PALABRARESERVADA, int>  
<ABREPARENTESIS, >  
<PALABRARESERVADA, string>  
<ID, 0>  
<CIERRAPARENTESIS, >  
<ABRECORCHETE, >  
<PALABRARESERVADA, let>  
<ID, 1>  
<PALABRARESERVADA, int>  
<ASIGNACION, >  
<ENTERO, 2>  
<PUNTOCOMA, >  
<PALABRARESERVADA, if>  
<ABREPARENTESIS, >  
<ID, 1>  
<COMPARADOR, >  
<ENTERO, 4>  
<CIERRAPARENTESIS, >  
<ABRECORCHETE, >  
<PALABRARESERVADA, put>  
<ID, 0>  
<PUNTOCOMA, >  
<CIERRACORCHETE, >  
<PALABRARESERVADA, return>  
<ID, 1>  
<PUNTOCOMA, >  
<CIERRACORCHETE, >  
<FINDEFICHERO, >
```

Tabla de Símbolos:

CONTENIDOS DE LA TABLA #1:

```
*      LEXEMA      :      'cadena'  
Atributos:  
+ tipo: 'STRING'  
+ despl: 0  
*      LEXEMA      :      'y'
```


Atributos:
+ tipo: 'INT'
+ despl: 128

CONTENIDOS DE LA TABLA #0:

* LEXEMA : 'nombre1'
Atributos:
+ tipo: 'FUNCTION'
+ numParam: '1'
+ TipoParam0: 'STRING'
+ ModoParam0: 'VALOR'
+ TipoRetorno: 'INT'
+ EtiqFuncion: 'nombre1'
Árbol Sintáctico:

Los arboles vast se pasan en el fichero como adjuntos para su visualización.

CASO 6 -> Incorrecto

```
let id int = 6;
```

```
while (id == (6+1))  
return %;
```

```
put id;
```

Error:

Error en el análisis léxico: Error al procesar el caracter actual: % en línea 7

CASO 7 -> Incorrecto

```
let id int = 7;
```

```
if(id == (6+1)) {  
id+=5}  
}
```

```
put id int;
```

Error:

Error en el análisis sintáctico: Error de análisis sintáctico en el token ABRECORCHETE en línea 5

CASO 8 -> Incorrecto

```
let id int;
```

```
function string nombrefun () {  
let dev string;  
return dev;  
}
```

Error:

Error en el análisis sintáctico: Error de análisis sintáctico en el token STRING en línea 4

CASO 9 -> Incorrecto

```
let id int;  
let id2 string = "Hola";
```

```
    if(id == id2)  
return id2;
```

Error:

Error en el análisis semántico: los tipos en la expresión E: E == U no coinciden en línea 6

CASO 10 -> Incorrecto

```
let id int;
```

```
function nombrefun int (int param) {  
    if(param == 0)  
    param+=5;
```

```
    let error string;  
    return error;  
}
```

Error:

Error en el análisis semántico: Hay un error con el tipo de retorno de la función en línea 10