Proyecto de Procesadores de Lenguajes

Participantes:

- Villca Salguero, Evert
- Hashemian, Seyed Mostafa
- García González, Javier

Índice

1. Diseño de Analizador Léxico

1.1. Definición de tokens

ELEMENTO	CÓDIGO	ATRIBUTO
boolean	1	
function	2	
if	3	
input	4	
int	5	
output	6	
return	7	
string	8	
var	9	
void	10	
while	11	
postdecremento (--)	12	
entero	13	número
cadena (“)	14	cadena (“)
identificador	15	número
=	16	
,	17	
;	18	
(	19	
)	20	
{	21	
}	22	
suma (+)	23	
negación (!)	24	
distinto (!=)	25	
eof	26	

1.2. Definición de la gramática

S -> - P | + | ! D | = | , | ; | (|) | { | } | l A | d B | del S | / C | “ G | eof

P -> -

D -> = | lambda

A -> l A | d A | _A | lambda

B -> d B | lambda

C -> * E

E -> c2 E | * F

F -> / S | * F | c3 E

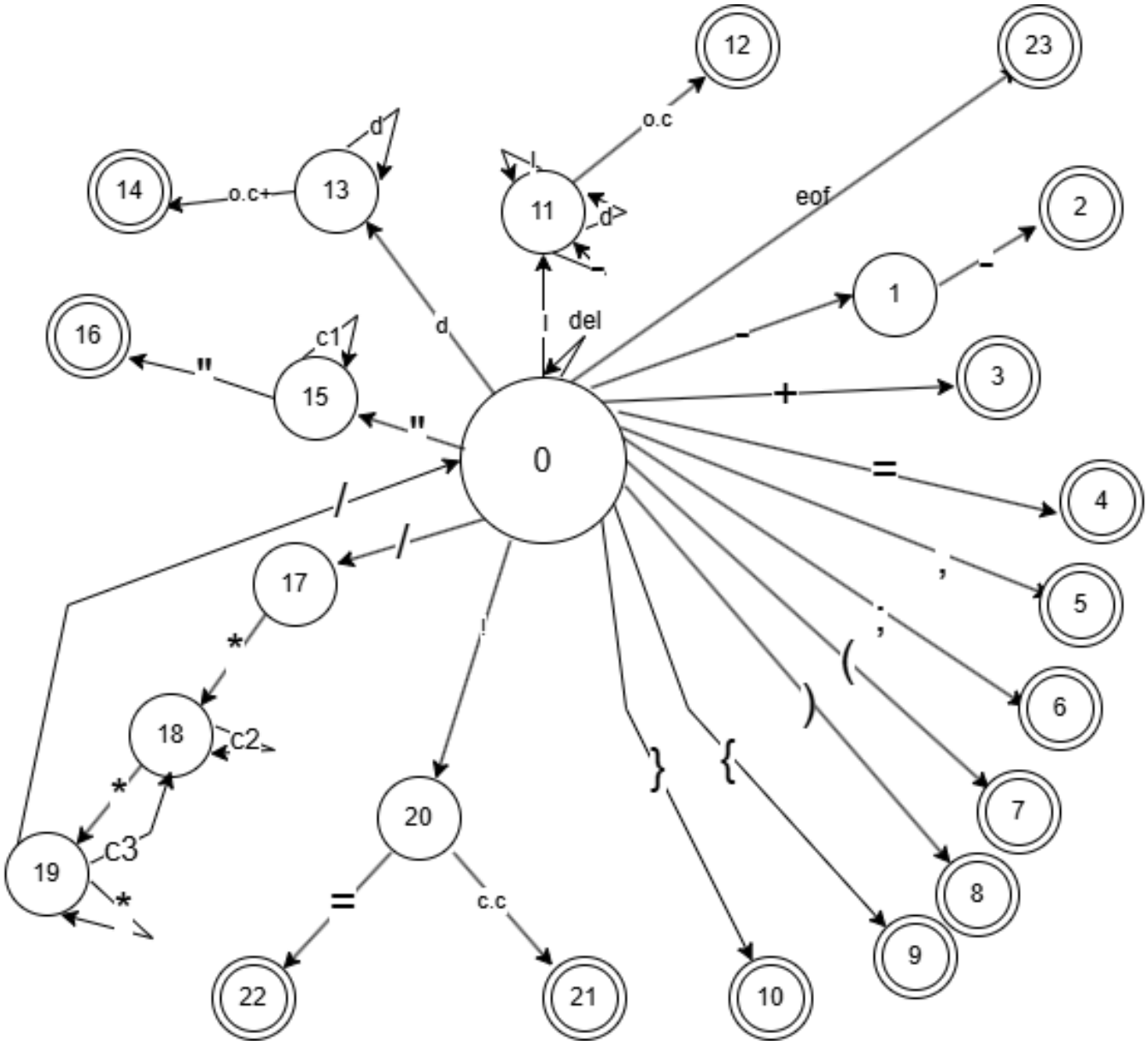
G -> c1 G | “

c1 -> char - “

c2 -> char - *

c3 -> char - * y /

1.3. AFD



1.4. Acciones semánticas

0-15 lex = Ø; leer()

15-15 lex = lex ⊕ c1; leer()

15-16 If(lex.length <= 64) {
leer();

```
            Gen_Token(cadena, lexema)
        }
        else error("Cadena fuera de rango")
```

```
0-13    num = valor(car);  leer()
13-13   num = num * 10 + valor(car);  leer()
13-14   if( num <= 32767) Gen_Token(entero, num)
        Else error( "Número fuera de rango")
```

```
0-11    lex =car; leer ()
11-11   lex= lex ⊕ car;  leer()
11-12   if ( lex pertenece a PalabrasReservadas) Gen_Token( )
        else {
            if ( zonaDec ) {
                p = buscaID(lex);
                if (p=null) {
                    p = insertaID(lex);
                    Gen_Token(15, p)
                }
                else {
                    p1 = buscaID(lex);
                    if( p1 = null ) {
                        p1 =  insertaID(lex);
                        Gen_Token(15, p1);
                    }
                    else error("Identificador ya declarado");
                }
            }
            else { p = buscaID(lex);
                if (p=null) p =insertaIDglobal(lex);
                Gen_Token(15, p);
            }
        }
```

```
0-1     leer()
1-2     leer();  Gen_Token(12, 0)
```

```
0-20    leer()
20-21   Gen_token(24, 0)
20-22   leer(); Gen_token(25, 0)
```

```
0-3     leer(); Gen_Token(23, 0)
```

0-4 leer(); Gen_Token(16, 0)
0-5 leer(); Gen_Token(17, 0)
0-6 leer(); Gen_Token(18, 0)
0-7 leer(); Gen_Token(19, 0)
0-8 leer(); Gen_Token(20, 0)
0-9 leer(); Gen_Token(21, 0)
0-10 leer(); Gen_Token(22, 0)
0-23 leer(); Gen_Token(26, 0)

0-0 leer()
0-17 leer()
17-18 leer()
18-18 leer()
18-19 leer()
19-18 leer()
19-19 leer()
19-0 leer()

1.5. Errores

Todo símbolo o secuencia de caracteres que no sea reconocido por el Autómata Finito Determinista (*AFD*) generará un error léxico. Entre los posibles errores léxicos se encuentran los siguientes:

- **Cadena fuera de rango:** Cuando una cadena de caracteres excede el límite permitido.
- **Número fuera de rango:** Cuando un valor numérico supera los límites establecidos.
- **Identificador ya declarado:** Cuando se intenta declarar un identificador que ya existe en el mismo ámbito.
- **Símbolo desconocido que no pertenece al lenguaje:** Cuando aparece un carácter o símbolo no válido dentro del lenguaje.
- **Comentario sin cerrar:** Cuando un comentario se deja abierto.

En el caso de que se produzca cualquier error, se indicará el número de línea junto al tipo de error.

2. Diseño de Analizador Sintáctico

2.1. Gramática LL(1)

Axioma = P
NoTerminales = { P B F A S Y E L X Q C T H K R G U O V Z }
Terminales = { boolean break function if input int output return string while var void entero cadena id -- = , ; : ( ) { } + ! != eof }

Producciones = {
P -> B P
P -> F P
P -> eof
S -> id Y
Y -> = E ;
Y -> (L) ;
Y -> -- ;
S -> output E ;
S -> input id ;
S -> return X ;
X -> E
X -> lambda
L -> E Q
L -> lambda
Q -> , E Q
Q -> lambda
B -> if (E) S
B -> while (E) { C }
B -> var T id ;
B -> S
F -> function H id (A) { C }
H -> T
H -> void
A -> void
A -> T id K
K -> , T id K
K -> lambda
C -> B C
C -> lambda
T -> int
T -> boolean
T -> string

E -> R G
G -> != R G
G -> lambda
R -> U O
O -> + U O
O -> lambda
U -> ! V
U -> V
V -> id Z
V -> entero
V -> cadena
V -> (E)
Z -> (L)
Z -> lambda
Z -> --
}

2.2. Demostración de que la gramática es LL(1)

Para desmostrar que la gramática es LL(1) utilizaremos la herramienta proporcionada SGDLL(1).

Análisis LL1 de gll1.txt

Analizando s□mbolo A
Analizando producci□n A -> void
FIRST de A -> void = { void }
Analizando producci□n A -> T id K
Analizando s□mbolo T
Analizando producci□n T -> int
FIRST de T -> int = { int }
Analizando producci□n T -> boolean
FIRST de T -> boolean = { boolean }
Analizando producci□n T -> string
FIRST de T -> string = { string }
FIRST de T = { boolean int string }
FIRST de A -> T id K = { boolean int string }
FIRST de A = { boolean int string void }
Analizando s□mbolo B
Analizando producci□n B -> if (E) S
FIRST de B -> if (E) S = { if }

Analizando producci□n B -> while (E) { C }
FIRST de B -> while (E) { C } = { while }
Analizando producci□n B -> var T id ;
FIRST de B -> var T id ; = { var }
Analizando producci□n B -> S
Analizando s□mbolo S
Analizando producci□n S -> id Y
FIRST de S -> id Y = { id }
Analizando producci□n S -> output E ;
FIRST de S -> output E ; = { output }
Analizando producci□n S -> input id ;
FIRST de S -> input id ; = { input }
Analizando producci□n S -> return X ;
FIRST de S -> return X ; = { return }
FIRST de S = { id input output return }
FIRST de B -> S = { id input output return }
FIRST de B = { id if input output return var while }
Analizando s□mbolo C
Analizando producci□n C -> B C
FIRST de C -> B C = { id if input output return var while }
Analizando producci□n C -> lambda
FIRST de C -> lambda = { lambda }
FIRST de C = { id if input output return var while lambda }
Calculando FOLLOW de C
FOLLOW de C = { } }
Analizando s□mbolo E
Analizando producci□n E -> R G
Analizando s□mbolo R
Analizando producci□n R -> U O
Analizando s□mbolo U
Analizando producci□n U -> ! V
FIRST de U -> ! V = { ! }
Analizando producci□n U -> V
Analizando s□mbolo V
Analizando producci□n V -> id Z
FIRST de V -> id Z = { id }
Analizando producci□n V -> entero
FIRST de V -> entero = { entero }
Analizando producci□n V -> cadena
FIRST de V -> cadena = { cadena }
Analizando producci□n V -> (E)

FIRST de V -> (E) = { (}
FIRST de V = { (cadena entero id }
FIRST de U -> V = { (cadena entero id }
FIRST de U = { ! (cadena entero id }
FIRST de R -> U O = { ! (cadena entero id }
FIRST de R = { ! (cadena entero id }
FIRST de E -> R G = { ! (cadena entero id }
FIRST de E = { ! (cadena entero id }
Analizando s□mbolo F
Analizando producci□n F -> function H id (A) { C }
FIRST de F -> function H id (A) { C } = { function }
FIRST de F = { function }
Analizando s□mbolo G
Analizando producci□n G -> != R G
FIRST de G -> != R G = { != }
Analizando producci□n G -> lambda
FIRST de G -> lambda = { lambda }
FIRST de G = { != lambda }
Calculando FOLLOW de G
Calculando FOLLOW de E
Calculando FOLLOW de X
FOLLOW de X = { ; }
Analizando s□mbolo Q
Analizando producci□n Q -> , E Q
FIRST de Q -> , E Q = { , }
Analizando producci□n Q -> lambda
FIRST de Q -> lambda = { lambda }
FIRST de Q = { , lambda }
Calculando FOLLOW de Q
Calculando FOLLOW de L
FOLLOW de L = {) }
FOLLOW de Q = {) }
FOLLOW de E = {) , ; }
FOLLOW de G = {) , ; }
Analizando s□mbolo H
Analizando producci□n H -> T
FIRST de H -> T = { boolean int string }
Analizando producci□n H -> void
FIRST de H -> void = { void }
FIRST de H = { boolean int string void }
Analizando s□mbolo K

Analizando producci□n K -> , T id K
FIRST de K -> , T id K = { , }
Analizando producci□n K -> lambda
FIRST de K -> lambda = { lambda }
FIRST de K = { , lambda }
Calculando FOLLOW de K
Calculando FOLLOW de A
FOLLOW de A = {) }
FOLLOW de K = {) }
Analizando s□mbolo L
Analizando producci□n L -> E Q
FIRST de L -> E Q = { ! (cadena entero id }
Analizando producci□n L -> lambda
FIRST de L -> lambda = { lambda }
FIRST de L = { ! (cadena entero id lambda }
Analizando s□mbolo O
Analizando producci□n O -> + U O
FIRST de O -> + U O = { + }
Analizando producci□n O -> lambda
FIRST de O -> lambda = { lambda }
FIRST de O = { + lambda }
Calculando FOLLOW de O
Calculando FOLLOW de R
FOLLOW de R = { !=) , ; }
FOLLOW de O = { !=) , ; }
Analizando s□mbolo P
Analizando producci□n P -> B P
FIRST de P -> B P = { id if input output return var while }
Analizando producci□n P -> F P
FIRST de P -> F P = { function }
Analizando producci□n P -> eof
FIRST de P -> eof = { eof }
FIRST de P = { eof function id if input output return var while }
Analizando s□mbolo X
Analizando producci□n X -> E
FIRST de X -> E = { ! (cadena entero id }
Analizando producci□n X -> lambda
FIRST de X -> lambda = { lambda }
FIRST de X = { ! (cadena entero id lambda }
Analizando s□mbolo Y
Analizando producci□n Y -> = E ;

```
FIRST de Y -> = E ;  =  { = }
Analizando producci□n Y -> ( L ) ;
FIRST de Y -> ( L ) ;  =  { ( }
Analizando producci□n Y -> -- ;
FIRST de Y -> -- ;  =  { -- }
FIRST de Y  =  { ( -- = }
Analizando s□mbolo Z
Analizando producci□n Z -> ( L )
FIRST de Z -> ( L )  =  { ( }
Analizando producci□n Z -> lambda
FIRST de Z -> lambda  =  { lambda }
Analizando producci□n Z -> --
FIRST de Z -> --  =  { -- }
FIRST de Z  =  { ( -- lambda }
Calculando FOLLOW de Z
Calculando FOLLOW de V
Calculando FOLLOW de U
FOLLOW de U  =  { != ) + , ; }
FOLLOW de V  =  { != ) + , ; }
FOLLOW de Z  =  { != ) + , ; }

An□lisis concluido satisfactoriamente
```

2.3. Procedimientos

```
function P()
   if sig_tok.id pertenece a first(B) then
      parse += "1 "
      B()
      P()
   else if sig_tok.id pertenece a first(F) then
      parse += "2 "
      F()
      P()
```

```
    end if
end function

function B()
    if sig_tok.id == 3 then
        parse += "17 "
        equipara(3)
        equipara(19)
        E()
        equipara(20)
        S()
    else if sig_tok.id == 11 then
        parse += "18 "
        equipara(11)
        equipara(19)
        E()
        equipara(20)
        equipara(21)
        C()
        equipara(22)
    else if sig_tok.id == 9 then
        parse += "19 "
        equipara(9)
        T()
        equipara(15)
        equipara(18)
    else if sig_tok.id pertenece a first(S) then
        parse += "20 "
        S()
    end if
end function

function F()
    if sig_tok.id == 2 then
        parse += "21 "
        equipara(2)
        H()
        equipara(15)
        equipara(19)
        A()
        equipara(20)
```

```
        equipara(21)
        C()
        equipara(22)
    else
        print("Se esperaba una funcion")
    end if
end function

function A()
    if sig_tok.id == 10 then
        parse += "24 "
        equipara(10)
    else if sig_tok.id pertenece a first(T) then
        parse += "25 "
        T()
        equipara(15)
        K()
    else
        print("Se esperaba la declaracion de los parametros de la funcion, pero se obtuvo
" + Token.cod.get(sig_tok.id))
    end if
end function

function S()
    if sig_tok.id == 15 then
        parse += "4 "
        equipara(15)
        Y()
    else if sig_tok.id == 6 then
        parse += "8 "
        equipara(6)
        E()
        equipara(18)
    else if sig_tok.id == 4 then
        parse += "9 "
        equipara(4)
        equipara(15)
        equipara(18)
    else if sig_tok.id == 7 then
        parse += "10 "
        equipara(7)
```

```
        X()
        equipara(18)
    else
        print("Se esperaba alguna sentencia simple " + Token.cod.get(sig_tok.id))
    end if
end function

function Y()
    if sig_tok.id == 16 then
        parse += "5 "
        equipara(16)
        E()
        equipara(18)
    else if sig_tok.id == 19 then
        parse += "6 "
        equipara(19)
        L()
        equipara(20)
        equipara(18)
    else if sig_tok.id == 12 then
        parse += "7 "
        equipara(12)
        equipara(18)
    else
        print("No se puede poner " + Token.cod.get(sig_tok.id) + " despues de una
variable")
    end if
end function

function E()
    if sig_tok.id pertenece a first(R) then
        parse += "33 "
        R()
        G()
    end if
end function

function L()
    if sig_tok.id pertenece a first(L) then
        parse += "13 "
        E()
```

```
        Q()
    else if sig_tok.id pertenece a follow(L) then
        parse += "14 "
    end if
end function

function X()
    if sig_tok.id pertenece a first(E) then
        parse += "11 "
        E()
    else if sig_tok.id pertenece a follow(X) then
        parse += "12 "
    else
        print("No se puede devolver " + Token.cod.get(sig_tok.id))
    end if
end function

function Q()
    if sig_tok.id == 17 then
        parse += "15 "
        equipara(17)
        E()
        Q()
    else if sig_tok.id pertenece a follow(Q) then
        parse += "16 "
    end if
end function

function C()
    if sig_tok.id pertenece a first(B) then
        parse += "28 "
        B()
        C()
    else if sig_tok.id pertenece a follow(C) then
        parse += "29 "
    end if
end function

function T()
    if sig_tok.id == 5 then
        parse += "30 "
```

```
        equipara(5)
    else if sig_tok.id == 1 then
        parse += "31 "
        equipara(1)
    else if sig_tok.id == 8 then
        parse += "32 "
        equipara(8)
    else
        print(Token.cod.get(sig_tok.id) + " no es un tipo de datos valido")
    end if
end function

function H()
    if sig_tok.id pertenece a first(T) then
        parse += "22 "
        T()
    else if sig_tok.id == 10 then
        parse += "23 "
        equipara(10)
    end if
end function

function K()
    if sig_tok.id == 17 then
        parse += "26 "
        equipara(17)
        T()
        equipara(15)
        K()
    else if sig_tok.id pertenece a follow(K) then
        parse += "27 "
    end if
end function

function R()
    if sig_tok.id pertenece a first(U) then
        parse += "36 "
        U()
        O()
    end if
end function
```

```
function G()
    if sig_tok.id == 25 then
        parse += "34 "
        equipara(25)
        R()
        G()
    else if sig_tok.id pertenece a follow(G) then
        parse += "35 "
    end if
end function

function U()
    if sig_tok.id == 24 then
        parse += "39 "
        equipara(24)
        V()
    else if sig_tok.id pertenece a first(V) then
        parse += "40 "
        V()
    end if
end function

function O()
    if sig_tok.id == 23 then
        parse += "37 "
        equipara(23)
        U()
        O()
    else if sig_tok.id pertenece a follow(O) then
        parse += "38 "
    end if
end function

function V()
    if sig_tok.id == 15 then
        parse += "41 "
        equipara(15)
        Z()
    else if sig_tok.id == 13 then
        parse += "42 "
```

```
        equipara(13)
    else if sig_tok.id == 14 then
        parse += "43 "
        equipara(14)
    else if sig_tok.id == 19 then
        parse += "44 "
        equipara(19)
        E()
        equipara(20)
    end if
end function

function Z()
    if sig_tok.id == 19 then
        parse += "45 "
        equipara(19)
        L()
        equipara(20)
    else if sig_tok.id == 12 then
        parse += "47 "
        equipara(12)
    else if sig_tok.id pertenece a follow(Z) then
        parse += "46 "
    else
        e    end if
end function

function equipara(i)
    if sig_tok.id == i then
        sig_tok = a.ALexico()
    else
        print("Error en la linea " + a.linea)
        print("Se esperaba " + Token.cod.get(i) + " pero se encontro " +
Token.cod.get(sig_tok.id))
        return
    end if
end function
```

3. Diseño de Analizador Semántico

```
P' -> { TSG = crearTS(); despTSG=0 } P { destruirTS(TSG) }
P -> B P
P -> F P
P -> eof

S -> id
        { id.tipo = buscaTipoTS(id.pos);
          if(id.tipo = function) Y.tipoParam = buscaTipoParam(id.pos);
          Y.tipo = id.tipo;
          if(id.tipo = null ) { insertarTipoTS(id.pos, entero);
                insertarDespTS(id.pos, desp);
                despTS(desp+1);
                id.tipo = int;
          }
        }
        Y

Y -> = E ;
        { if(Y.tipo != E.tipo)
                error("A la variable se le debe asignar un valor del mismo tipo");
        }

Y -> ( L ) ;
        { if(Y.tipoParam != L.tipo)
                error("Llamada a funcion con parametros incorrectos");
        }
Y -> -- ;
        { if(Y.tipo != int)
                error("Solo se puede hacer postdecremento de un entero");
        }

S -> output E ;
        { if(E.tipo == string || E.tipo == int ) S.tipo = tipo_ok
          else S.tipo = error("Solo se puede hacer output de enteros o cadenas")
        }

S -> input id ;
        { id.tipo = buscaTipoTS(id.pos)
          if( id.tipo == string || id.tipo == int) S.tipo=tipo_ok
          else S.tipo = error("Solo se puede hacer input de cadenas o enteros")
```

```
        }

S -> return X ;
        { if(S.tipoRet = null) S.tipo = error("Return no debe ir fuera de funcion);
          else if(S.tipoRet = X.tipo) S.tipo = ok;
          else S.tipo = error("Retorno de tipo incorrecto);
        }

X -> E
        { X.tipo = E.tipo;
        }

X -> lambda
        { X.tipo = void;
        }
L -> E Q
        { L.tipo = E.tipo x Q.tipo;
        }

L -> lambda
        { L.tipo = null;
        }

Q -> , E Q1
        { Q.tipo = E.tipo x Q1.tipo;
        }

Q -> lambda
        { Q.tipo = null
        }

B -> if ( E ) S
        { if(E.tipo = boolean) B.tipo = S.tipo
          else error("Condicion dentro del if debe de booleana);
        }

B -> while ( E ) { C }
        { if(E.tipo = logico) B.tipo = C.tipo;
          else B.tipo = error("Condicion del while debe ser logica");
        }
```

```
B -> {zonaDec = true}
          var T id ;
          { insertarTipoTS(id.pos, T.tipo);
            insertarDespTS(id.pos, desp);
            zonaDec = false;
          }

B -> { S.tipoRet = B.tipoRet }
          S
          { B.tipo = S.tipo;
          }

F -> function H {zonaDec = true} id
          { TSL = crearTS(); TSactual = TSL;
            insertarTipoTS(id.pos, funcion);
            insertaEtTS(id.pos, etiqueta);
            zonaDec = false;
            despL = 0;
          }
          ( A )
          { insertarNumParam(id.pos, A.numParam);
     insertarTipoParam(id.pos, A.tipo);
     C.tipoRet = H.tipo
   }
   { C }
   {     destruirTS();     }

H -> T
          { H.tipo = T.tipo;
          }

H -> void
          { H.tipo = void;
          }

A -> void
          { A.tipo = void;
          }

A -> T { zonaDec = true } id K
          { insertarTipoTS(id.pos, T.tipo);
```

```
            insertarDespTS(id.pos, desp);
            desp = desp + T.ancho;
            zonaDec = false;
            A.numParam = 1 + K.numParam;
            A.tipo = T.tipo x K.tipo;
            zonaDec = false;
          }

K -> , T id K1
          { zonaDec = true;
      insertarTipoTS(id.pos, T.tipo);
            insertarDespTS(id.pos, desp);
            desp = desp + T.ancho;
      zonaDec = false;

      K.tipo = T.tipo x K1.tipo
            K.numParam = 1 + K1.numParam;
          }

K -> lambda
          { K.tipo = null;
          }

C -> { B.tipoRet, C1.tipoRet = C.tipoRet }
          B C1
          { if(C.tipo = vacio)
          }

C -> lambda

T -> int
T -> boolean
T -> string

E -> R G
          { if( G.tipo = vacio ) E.tipo = R.tipo;
            else if( R.tipo = int && G.tipo = int ) E.tipo = boolean;
          }

G -> != R G1
          { if( R.tipo = int && G1.tipo = int ) G.tipo = int;
```

```
    if( R.tipo = int && G1.tipo = vacio ) G.tipo = int;
        else G.tipo = error("La operacion != solo se puede realizar en enteros");
      }

G -> lambda

R -> U O
      { if(O.tipo = vacio) R.tipo = U.tipo;
        else if(U.tipo, O.tipo = int) R.tipo = int;
        else R.tipo = error("Tipos deben ser enteros en una suma");
      }

O -> + U O1
      { if( U.tipo = int && O1.tipo = vacio ) O.tipo = int;
    else if( U.tipo = int && O1.tipo = int ) O.tipo = int;
        else O.tipo = error("Tipos deben ser enteros");
      }

O -> lambda
      { O.tipo = vacio }

U -> ! V
      { if(V.tipo = boolean) U.tipo = boolean
        else U.tipo = error("Debe se de tipo logico");
      }

U -> V
      { U.tipo = V.tipo }

V -> id
      { id.tipo = buscaTipoTS(id.pos);
        if(id.tipo = funcion)
            Z.tipoParam = buscaTipoParam(id.pos);
        Z.tipo = id.tipo;
      }
      Z
      { if(id.tipo = function)
            V.tipo = buscaTipoRet(id.pos);
        else V.tipo = id.tipo;
      }
```

```
V -> entero
        { V.tipo = int }

V -> cadena
        { V.tipo = string }

V -> ( E )
        { V.tipo = E.tipo }

Z -> ( L )
        { if(Z.idTipo = null){ error(“Funcion no declarada”) }
          if( Z.tipoParam != L.tipo )
                error("Llamada a funcion con parametros incorrectos");
        }

Z -> lambda
        { if( Z.idTipo=null){
                 insertarTipoTS(id.pos, entero);
                insertarDespTS(id.pos, desp);
                despTS(desp+1);
                id.tipo = int;
                Z.dev = int;
           }
           else Z.dev = Z.idTipo;
        }

Z -> --
        { if( Z.idTipo != int)
                error("Solo se puede hacer postdescremento de variables enteras");
          Z.dev = Z.idTipo;
        }
```

3.2. Diseño de la tabla de símbolos

Contamos con una clase TS encargada de gestionar las tablas de símbolos. Contamos con 3 tablas: Global, Local y Actual. Internamente las tablas se manejan con un HashMap donde la key es la posición del identificador y el value es una “fila” que contiene el lexema, tipo, desplazamiento, etc. Para mayor comodidad se han usado posiciones a partir de la 400000 para las tablas locales y para las globales a partir del 0.

4. Anexo: Casos de prueba

4.1. Casos de prueba con error

Función declarada dentro de otra función

```
function int Func2 (int z, int n)
{
  n = 13;
  z = y1 + Func2(4, global);
  function void hola (void)
  {
    output"Hello!";
  }
  return 5;
}
```

```
Error en la linea 5: No se permite la definicion de funciones anidadas
```

Función devuelve tipo incorrecto de datos

```
function int Func1 (int x, boolean y1)
{
  var boolean w;
  x = 11;


  return y1;
}
```

```
Error en la linea 6: Devuelve un tipo incorrecto 'boolean'. Deberia devolver 'int'
```

Cadena sin cerrar

```
var int y1;
var string z;
function int Func2 (int a, int n)
{
  n = 13;
  return 5;
}

z="hola;
```

```
Error en linea 10: Cadena sin cerrar
```

Llamada a función con parámetros incorrectos

```
function int Func1 (int x, boolean y1)
{
  return x;
}


z = Func1(6, 7, 7);
```

```
Error en la linea 6: Llamada a funcion con parametros incorrectos
```

Llamada a función no declarada

```
var int a;
var string b;

function int Func1 (int x, boolean y1)
{
  x = 11;
  return x;
}

resultado = FuncNoDeclarada(1, 2);
```

```
Error en la linea 10: Función 'FuncNoDeclarada' no declarada
```

4.2. Casos de prueba sin error

Ejemplo 1: Uso de while, operadores post-drecemento y distinto

```
var int i;
i = 5;

while (i != 0)
{
  output(i);
  i--;
}
```

Tokens:
<9, >
<5, >

<15, 0>
<18, >
<15, 0>
<16, >
<13, 5>
<18, >
<11, >
<19, >
<15, 0>
<25, >
<13, 0>
<20, >
<21, >
<6, >
<19, >
<15, 0>
<20, >
<18, >
<15, 0>
<12, >
<18, >
<22, >
<26, >

Árbol sintáctico:

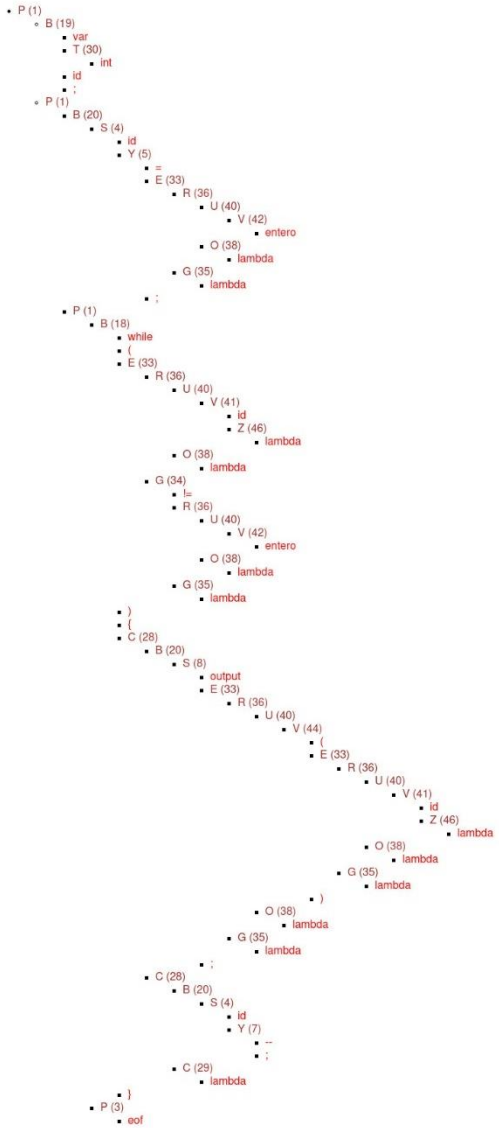


Tabla de símbolos:
TABLA GLOBAL #0:

* LEXEMA : 'i'
ATRIBUTOS:
+ tipo : 'int'
+ despl : 0

Ejemplo 2: Uso del if, negación y variable no declarada

```
var boolean b;
if (!b)
   c = 7;
output(c);
```

Tokens:
<9, >
<1, >
<15, 0>
<18, >
<3, >
<19, >
<24, >
<15, 0>
<20, >
<15, 1>
<16, >
<13, 7>
<18, >
<6, >
<19, >
<15, 1>
<20, >
<18, >
<26, >

Tabla de Símbolos:
TABLA GLOBAL #0:
* LEXEMA : 'b'
ATRIBUTOS:
+ tipo : 'boolean'
+ despl : 0
---
* LEXEMA : 'c'
ATRIBUTOS:
+ tipo : 'int'
+ despl : 1

Árbol sintáctico:

Ejemplo 3: Función void y uso de comentarios

```
function void saludo(void)
{
  output("Bienvenido");
  /* comentario en bloque */
}
saludo()
```

Tokens:

<2, >
<10, >
<15, 0>
<19, >
<10, >
<20, >
<21, >
<6, >
<19, >
<14, "Bienvenido">
<20, >
<18, >
<22, >
<15, 0>
<19, >
<20, >
<18, >
<26, >

Árbol sintáctico:

Tabla de Símbolos:

TABLA LOCAL saludo #1:

TABLA GLOBAL #0:

* LEXEMA : 'saludo'
ATRIBUTOS:
+ tipo : 'function'
+ numParam : 0
+ TipoRetorno : 'void'
+ EtiqFuncion : 'Et_saludo'

Ejemplo 4: Declaración y uso de variables. Función sin error

```
var int a;
var boolean b;

function int suma(int x, int y)
{
   return x + y;
}

a = 10;
output(a);
```

Tokens:
<9, >
<5, >
<15, 0>
<18, >
<9, >
<1, >
<15, 1>
<18, >
<2, >
<5, >
<15, 2>
<19, >
<5, >
<15, 400000>
<17, >
<5, >
<15, 400001>
<20, >
<21, >
<7, >
<15, 400000>
<23, >
<15, 400001>
<18, >

<22, >
<15, 0>
<16, >
<13, 10>
<18, >
<6, >
<19, >
<15, 0>
<20, >
<18, >
<26, >

Tabla de Símbolos:

TABLA LOCAL suma #1:

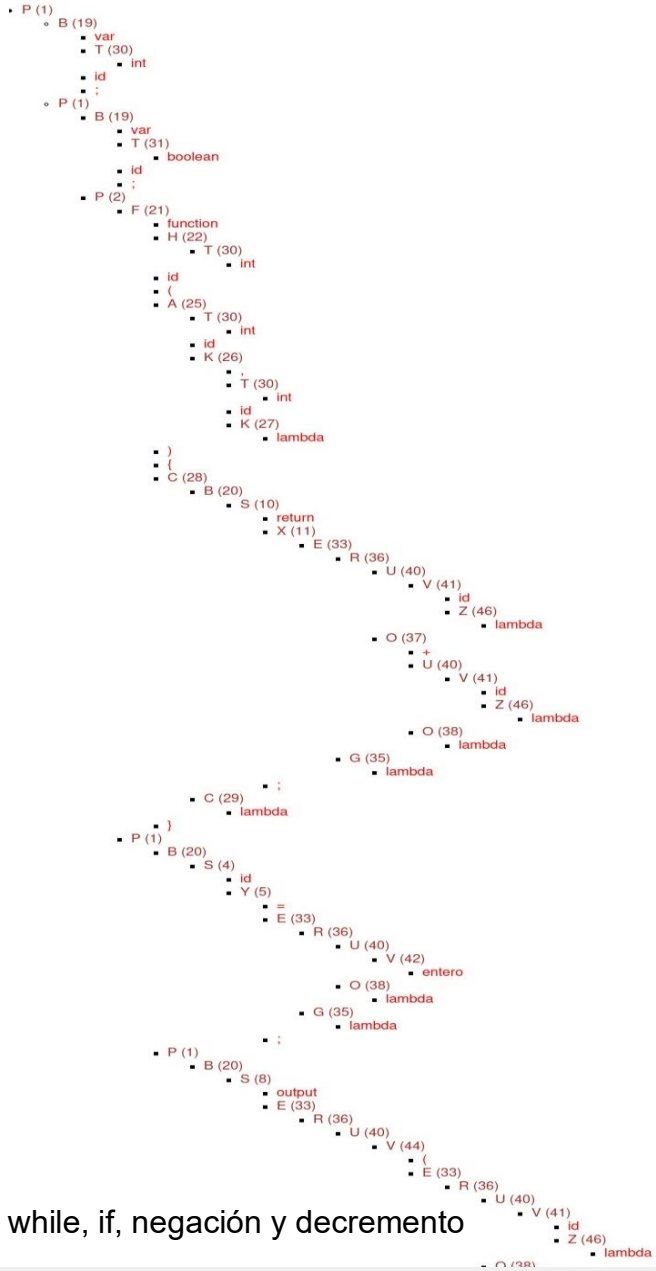
* LEXEMA : 'x'
ATRIBUTOS:
+ tipo : 'int'
+ despl : 0
---
* LEXEMA : 'y'
ATRIBUTOS:
+ tipo : 'int'
+ despl : 1

TABLA GLOBAL #0:

* LEXEMA : 'a'
ATRIBUTOS:
+ tipo : 'int'
+ despl : 0
---
* LEXEMA : 'b'
ATRIBUTOS:
+ tipo : 'boolean'
+ despl : 1
---
* LEXEMA : 'suma'
ATRIBUTOS:
+ tipo : 'function'
+ numParam : 2
+ TipoParam01 : 'int'
+ TipoParam02 : 'int'

+ TipoRetorno : 'int'
+ EtiqFuncion : 'Et_suma'

Árbol sintáctico:



Ejemplo 5: Uso de while, if, negación y decremento

```
var int contador;
contador = 3;

while (contador != 0)
{
  if (!(contador != 2))
```

```
    output(contador);
  contador--;
}
```

Tokens:

<9, >
<5, >
<15, 0>
<18, >
<15, 0>
<16, >
<13, 3>
<18, >
<11, >
<19, >
<15, 0>
<25, >
<13, 0>
<20, >
<21, >
<3, >
<19, >
<24, >
<19, >
<15, 0>
<25, >
<13, 2>
<20, >
<20, >
<6, >
<19, >
<15, 0>
<20, >
<18, >
<15, 0>
<12, >
<18, >
<22, >
<26, >

Árbol sintáctico:

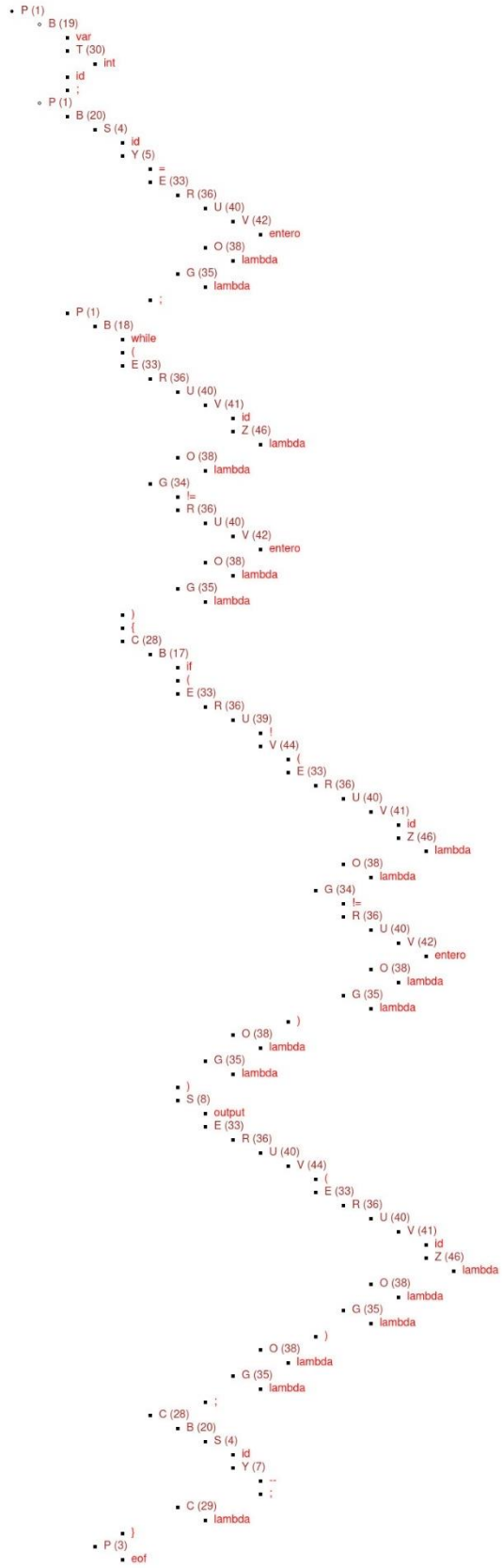


Tabla de Símbolos:
TABLA GLOBAL #0:

* LEXEMA : 'contador'
ATRIBUTOS:
+ tipo : 'int'
+ despl : 0
