MEMORIA PROYECTO PROCESADORES DE LENGUAJES

GRUPO 109

Sergio Zaballos Herrera nº matrícula: a180185

INDICE:

0. Características específicas del lenguaje

1. Diseño del Analizador Léxico

1.1 --- Definición de tokens
1.2 --- Definición de la gramática
1.3 --- Autómata finito determinista
1.4 --- Acciones semánticas
1.5 --- Errores
1.6--- Explicación del funcionamiento

2. Diseño del Analizador Sintáctico

2.1 --- Gramática LL(1)
2.2 --- Demostración de la gramática LL(1)
2.3 --- Tabla sintáctica
2.4 --- Explicación del funcionamiento

3.Diseño del Analizador Semántico

4. Diseño de la Tabla de Símbolos

5. Diseño del Gestor de Errores

6. Anexo: Casos de Prueba

6.1 --- Casos de Prueba sin errores (tokens sacados, árbol sintáctico, tab.simbolos)
6.2 --- Casos de prueba con errores (mensaje de error)

0. Características específicas del lenguaje

Las Características elegidas e impuestas para el lenguaje son:
- Operadores:
    - Aritmético (+)
    - Relacional (>)
    - Lógico (&&)
- Comentarios (/* */)
- Cadenas (' ')
- Sentencia de bucle (for)
- Asignación con multiplicación (*=)
- Técnica de análisis Descendente con tablas

1. Diseño del Analizador Léxico

1.1 Definición de tokens

Para realizar el Analizador Léxico, primero tenemos que identificar qué vamos a considerar como 'tokens'. Vamos a considerar los siguientes 'tokens':

ELEMENTO	**CÓDIGO**	**ATRIBUTO**
boolean	boolean	-
for	for	-
function	function	-
get	get	-
if	if	-
int	int	-
let	let	-
put	put	-
return	return	-
string	string	-
void	void	-
constanteEntera	constEnt	número
cadenaString	cadena	lexema
id	ID	posTS
*=	opMult	-
=	igual	-
,	coma	-
;	puntoycoma	-
(	abrirParentesis	-
)	cerrarParentesis	-
{	abrirLlave	-
}	cerrarLlave	-
+	mas	-
&&	and	-
>	mayorQ	-

1.2 Definición de la gramática

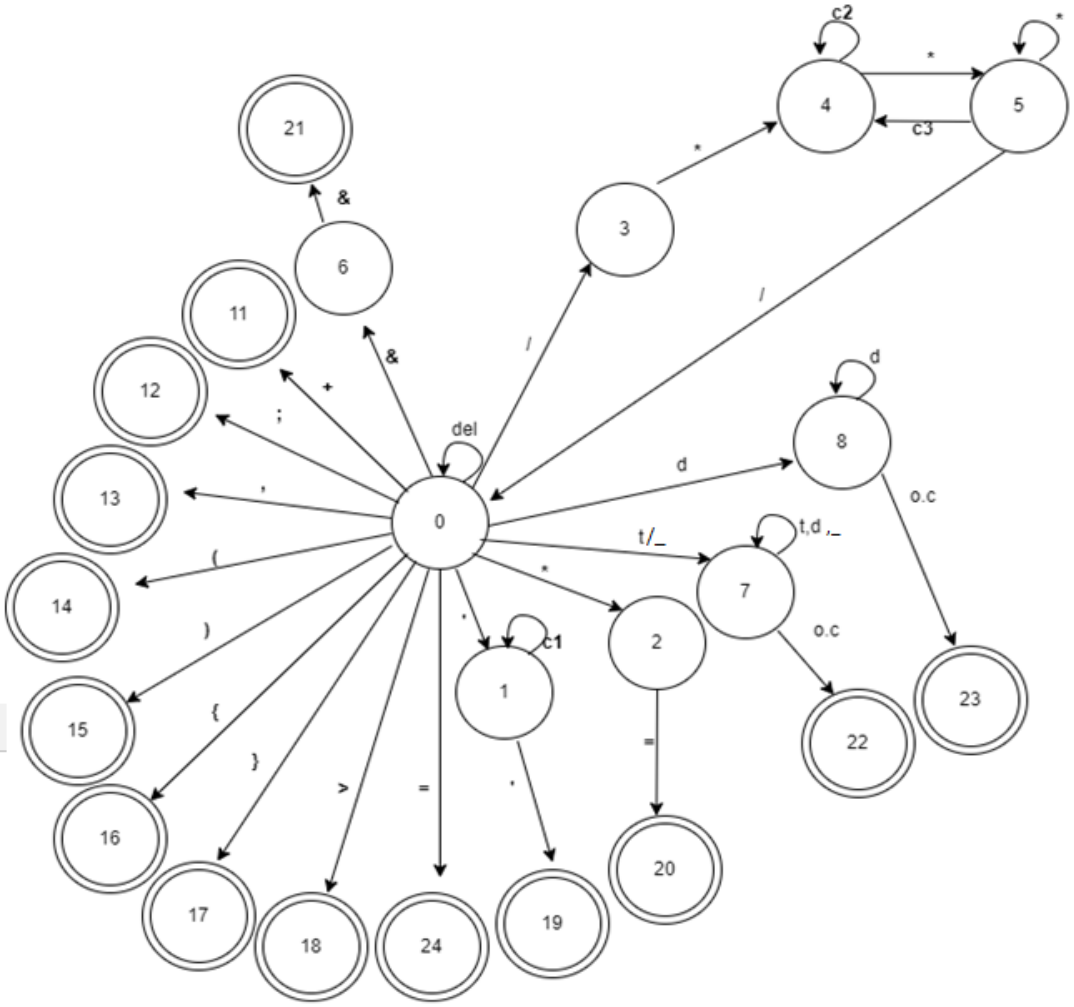
Lo siguiente a realizar es la construcción de la gramática del Analizador Léxico, quedando:

t = cualquier letra {a-z , A-Z} d = cualquier dígito {0-9}
c1 = cualquier carácter - { ‘ } c2= cualquier carácter - {*} c3= cualquier carácter - {*,/}

1. S -> delS | + | ; | , | (|) | { | } | > | = | ‘A | *C | /D | &E | dB | tG | _G
2. A-> c1A| ’
3. C-> =
4. D-> *D’
5. D’-> c2D’ | *D’’
6. D’’-> *D’’ | c3D’ | /S
7. E-> &
8. B-> dB | lambda
9. G-> tG | dG |_G | lambda

1.3 Autómata Finito Determinista

Posteriormente construimos la gramática del Analizador Léxico, quedando de esta manera:



1.4 Acciones Semánticas

El realizar las acciones semánticas acaba resultando en:

LEER(): todas las transiciones menos o.c

0-11 GENTOKEN(suma,) 0-12 GENTOKEN(puntoycoma,)
0-13 GENTOKEN(coma ,) 0-14 GENTOKEN(abrirParentesis,)
0-15 GENTOKEN(cerrarParentesis,) 0-16 GENTOKEN(abrirLlave,)
0-17 GENTOKEN(cerrarLlave,) 0-18 GENTOKEN(mayorQ,)
0-24 GENTOKEN(igual,)
0-2 /lexema=* 2-20 GENTOKEN(opMult,)
0-6 lexema=& 6-7 GENTOKEN(and,)
0-1 lexema =’ 1-1 lexema = lexema + c1 1-19 lexema=lexema+’
GENTOKEN(cadena,lexema)
0-8 numero = d
8-8 numero = numero*10 + d
8-23 GENTOKEN(constEnt,numero)
0-7 lexema = t/_
7-7 lexema = lexema + t/d/_
7-22 if(lexema ∈ PalabrasReservadas) then GENTOKEN(lexema,)

```
Else
    If(ZonaDec=true)
        p = buscarTS(lexema)
        if(p=null) then p = insertarTS(lexema);  GENTOKEN(ID,p)
        else ERROR(“ID YA DECLARADO”)
    else //ZonaDec=false
        p = buscarTS(lexema)
        if(p!=null) then GENTOKEN(ID,p);
        else if(buscarTSGlobal(lexema)!=null) then GENTOKEN(ID,p);
        else//no esta ni en la local ni en la global
            insertarTSGlobal(lexema);      pos = TSGlobal.getPos(lexema);
            TSGlobal.setTipo(pos,”entero”);   TSGLOBAL.añadirDespl(pos,TSGlobal.despl);
            TSGlobal.setDespl(1);
            GENTOKEN(ID,pos);
```

1.5 Errores

Cualquier transición que no aparezca en el Autómata corresponde a errores.

1.6 Explicación del funcionamiento

Al Analizador Léxico se le inserta el texto que se quiere procesar para ir sacando los tokens según lo vaya requiriendo el Analizador Sintáctico. Cuando se le llama, se pone a procesar el texto desde donde lo dejó la iteración anterior, eliminando las partes del texto que ya han sido procesadas. Para ir procesando los caracteres del texto se genera un objeto 'Automata' que contiene la matriz de transiciones equivalente al autómata de nuestra gramática, siendo cada estado de esta un objeto 'estado'. Se compone principalmente de 2 métodos:

getNextChar(cadena,pos): Devuelve el carácter correspondiente a la posición dada de la cadena. Si hay un salto de línea aumenta el número de línea en 1 y hace 'reset' a 'pos' para devolver el primer carácter de la siguiente línea.

getToken(tablas): Según el carácter encontrado, a partir de ese carácter y el estado en el que se encuentre del autómata, cambiará el 'estadoActual' por el siguiente estado del Autómata y volverá a repetir el proceso hasta que salte un error léxico(cualquier estado no correspondiente al autómata, devolviendo el error y su línea al Analizador Sintáctico) o llegue a un estado final, donde generará el token correspondiente a ese estado final y se lo pasará al analizador Sintáctico. En caso de palabras reservadas generará un token con aquella palabra reservada y en caso de IDs, meterá los IDs en la tabla correspondiente con zonaDec=true o devolverá error si ya ha sido declarado anteriormente. En caso de zonaDec = false, si no ha sido declarado anteriormente el ID lo insertará en la TS global como un entero devolviendo un token de ese ID recién creado (declaración implícita de variables), en cambio si ha sido declarado devolverá un token de ese ID.

El Analizador léxico acaba devolviendo un objeto 'TokenTablas', siendo este una dupla del token que se está generando y la pila de tablas sobre la cual estamos interactuando (esta estando constantemente pasándose entre el léxico, sintáctico y semántico), en la cual si se ha generado un id se habrá insertado en la tabla de símbolos correspondiente ese id.

2. Diseño del Analizador Sintáctico

2.1 Gramática LL(1)

NoTerminales = { D P B E O R W U Y V N L Q X S M T H A K F C Z }
Axioma = D
Terminales = { id *= = put get return if let for int boolean string ; function > void , && ( ) entero cadena + { } }
Producciones = {
D -> P
P -> B P
P -> F P
P -> lambda

E -> R O
O -> && R O
O -> lambda
R -> U W
W -> > U W
W -> lambda
U -> V Y
Y -> + V Y
Y -> lambda
V -> id N
N -> (L)
N -> lambda
V -> (E)
V -> entero
V -> cadena

S -> id M
M -> = E ;
M -> *= E ;
M -> (L)
S -> put E ;
S -> get id ;
S -> return X ;
L -> E Q
L -> lambda
Q -> , E Q
Q -> lambda
X -> E

B -> if (E) S
B -> let id T ;
B -> S
B -> for (id M ; E ; id M) Z

T -> int
T -> boolean
T -> string
F -> function id H (A) Z
Z -> { C }
H -> T
H -> void
A -> T id K
A -> void
K -> , T id K
K -> lambda
C -> B C
C -> lambda
}

2.2 Demostración de la Gramática LL(1)

Se ha realizado un análisis con la aplicación aportada SDGLL(1):

Iniciando análisis

Lectura del fichero realizada con éxito

Iniciando análisis LL1 de la gramática

Analizando símbolo A

Analizando producción A -> T id K

Analizando símbolo T

Analizando producción T -> int

FIRST de T -> int = { int }

Analizando producción T -> boolean

FIRST de T -> boolean = { boolean }

Analizando producción T -> string

FIRST de T -> string = { string }

FIRST de T = { boolean int string }

FIRST de A -> T id K = { boolean int string }

Analizando producción A -> void

FIRST de A -> void = { void }

FIRST de A = { boolean int string void }

Analizando símbolo B

Analizando producción B -> if (E) S

FIRST de B -> if (E) S = { if }

Analizando producción B -> let id T ;

FIRST de B -> let id T ; = { let }

Analizando producción B -> S

Analizando símbolo S

Analizando producción S -> id M

FIRST de S -> id M = { id }

Analizando producción S -> put E ;

FIRST de S -> put E ; = { put }

Analizando producción S -> get id ;

FIRST de S -> get id ; = { get }

Analizando producción S -> return X ;

FIRST de S -> return X ; = { return }

FIRST de S = { get id put return }

FIRST de B -> S = { get id put return }

Analizando producción B -> for (id M ; E ; id M) Z

FIRST de B -> for (id M ; E ; id M) Z = { for }

FIRST de B = { for get id if let put return }

Analizando símbolo C

Analizando producción C -> B C

FIRST de C -> B C = { for get id if let put return }

Analizando producción C -> lambda

FIRST de C -> lambda = { lambda }

FIRST de C = { for get id if let put return lambda }

Calculando FOLLOW de C

FOLLOW de C = { } }

Analizando símbolo D

Analizando producción D -> P

Analizando símbolo P

Analizando producción P -> B P

FIRST de P -> B P = { for get id if let put return }

Analizando producción P -> F P

Analizando símbolo F

Analizando producción F -> function id H (A) Z

FIRST de F -> function id H (A) Z = { function }

FIRST de F = { function }

FIRST de P -> F P = { function }

Analizando producción P -> lambda

FIRST de P -> lambda = { lambda }

FIRST de P = { for function get id if let put return lambda }

Calculando FOLLOW de P

Calculando FOLLOW de D

FOLLOW de D = { $ (final de cadena) }

FOLLOW de P = { $ (final de cadena) }

FIRST de D -> P = { for function get id if let put return lambda }

FIRST de D = { for function get id if let put return lambda }

Analizando símbolo E

Analizando producción E -> R O

Analizando símbolo R

Analizando producción R -> U W

Analizando símbolo U

Analizando producción U -> V Y

Analizando símbolo V

Analizando producción V -> id N

FIRST de V -> id N = { id }

Analizando producción V -> (E)

FIRST de V -> (E) = { (}

Analizando producción V -> entero

FIRST de V -> entero = { entero }

Analizando producción V -> cadena

FIRST de V -> cadena = { cadena }

FIRST de V = { (cadena entero id }

FIRST de U -> V Y = { (cadena entero id }

FIRST de U = { (cadena entero id }

FIRST de R -> U W = { (cadena entero id }

FIRST de R = { (cadena entero id }

FIRST de E -> R O = { (cadena entero id }

FIRST de E = { (cadena entero id }

Analizando símbolo H

Analizando producción H -> T

FIRST de H -> T = { boolean int string }

Analizando producción H -> void

FIRST de H -> void = { void }

FIRST de H = { boolean int string void }

Analizando símbolo K

Analizando producción K -> , T id K

FIRST de K -> , T id K = { , }

Analizando producción 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 símbolo L

Analizando producción L -> E Q

FIRST de L -> E Q = { (cadena entero id }

Analizando producción L -> lambda

FIRST de L -> lambda = { lambda }

FIRST de L = { (cadena entero id lambda }

Calculando FOLLOW de L

FOLLOW de L = {) }

Analizando símbolo M

Analizando producción M -> = E ;

FIRST de M -> = E ; = { = }

Analizando producción M -> *= E ;

FIRST de M -> *= E ; = { *= }

Analizando producción M -> (L)

FIRST de M -> (L) = { (}

FIRST de M = { (*= = }

Analizando símbolo N

Analizando producción N -> (L)

FIRST de N -> (L) = { (}

Analizando producción N -> lambda

FIRST de N -> lambda = { lambda }

FIRST de N = { (lambda }

Calculando FOLLOW de N

Calculando FOLLOW de V

Analizando símbolo Y

Analizando producción Y -> + V Y

FIRST de Y -> + V Y = { + }

Analizando producción Y -> lambda

FIRST de Y -> lambda = { lambda }

FIRST de Y = { + lambda }

Calculando FOLLOW de Y

Calculando FOLLOW de U

Analizando símbolo W

Analizando producción W -> > U W

FIRST de W -> > U W = { > }

Analizando producción W -> lambda

FIRST de W -> lambda = { lambda }

FIRST de W = { > lambda }

Calculando FOLLOW de W

Calculando FOLLOW de R

Analizando símbolo O

Analizando producción O -> && R O

FIRST de O -> && R O = { && }

Analizando producción O -> lambda

FIRST de O -> lambda = { lambda }

FIRST de O = { && lambda }

Calculando FOLLOW de O

Calculando FOLLOW de E

Analizando símbolo Q

Analizando producción Q -> , E Q

FIRST de Q -> , E Q = { , }

Analizando producción Q -> lambda

FIRST de Q -> lambda = { lambda }

FIRST de Q = { , lambda }

Calculando FOLLOW de Q

FOLLOW de Q = {) }

Calculando FOLLOW de X

FOLLOW de X = { ; }

FOLLOW de E = {) , ; }

FOLLOW de O = {) , ; }

FOLLOW de R = { &&) , ; }

FOLLOW de W = { &&) , ; }

FOLLOW de U = { &&) , ; > }

FOLLOW de Y = { &&) , ; > }

FOLLOW de V = { &&) + , ; > }

FOLLOW de N = { &&) + , ; > }

Analizando símbolo X

Analizando producción X -> E

FIRST de X -> E = { (cadena entero id }

FIRST de X = { (cadena entero id }

Analizando símbolo Z

Analizando producción Z -> { C }

FIRST de Z -> { C } = { { }

FIRST de Z = { { }

Análisis concluido satisfactoriamente

2.3 Tabla Sintáctica

Formada con la aplicación aportada SDGLL(1):

	&&	(	)	*=	+	,	;	=	>	boolean	cadena	entero	for	function	get	id	if	int	let	put	return	string	void	{	}	$ (final de cadena)
A	--	--	--	--	--	--	--	--	--	A → T id K	--	--	--	--	--	--	--	A → T id K	--	--	--	A → T id K	A → void	--	--	--
B	--	--	--	--	--	--	--	--	--	--	--	--	B → for (id M ; E ; id M) Z	--	B → S	B → S	B → if (E) S	--	B → let id T ;	B → S	B → S	--	--	--	--	--
C	--	--	--	--	--	--	--	--	--	--	--	--	C → B C	--	C → B C	C → B C	C → B C	--	C → B C	C → B C	C → B C	--	--	--	C → lambda	--
D	--	--	--	--	--	--	--	--	--	--	--	--	D → P	D → P	D → P	D → P	D → P	--	D → P	D → P	D → P	--	--	--	--	D → P
E	--	E → R O	--	--	--	--	--	--	--	--	E → R O	E → R O	--	--	--	E → R O	--	--	--	--	--	--	--	--	--	--
F	--	--	--	--	--	--	--	--	--	--	--	--	--	F → function id H (A) Z	--	--	--	--	--	--	--	--	--	--	--	--
H	--	--	--	--	--	--	--	--	--	H → T	--	--	--	--	--	--	--	H → T	--	--	--	H → T	H → void	--	--	--
K	--	--	K → lambda	--	--	K → , T id K	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
L	--	L → E Q	L → lambda	--	--	--	--	--	--	--	L → E Q	L → E Q	--	--	--	L → E Q	--	--	--	--	--	--	--	--	--	--
M	--	M → (L)	--	M → *= E ;	--	--	--	M → = E ;	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
N	N → lambda	N → (L)	N → lambda	--	N → lambda	N → lambda	N → lambda	--	N → lambda	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
O	O → && R O	--	O → lambda	--	--	O → lambda	O → lambda	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
P	--	--	--	--	--	--	--	--	--	--	--	--	P → B P	P → F P	P → B P	P → B P	P → B P	--	P → B P	P → B P	P → B P	--	--	--	--	P → lambda
Q	--	--	Q → lambda	--	--	Q → , E Q	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
R	--	R → U W	--	--	--	--	--	--	--	--	R → U W	R → U W	--	--	--	R → U W	--	--	--	--	--	--	--	--	--	--
S	--	--	--	--	--	--	--	--	--	--	--	--	--	--	S → get id ;	S → id M	--	--	--	S → put E ;	S → return X ;	--	--	--	--	--
T	--	--	--	--	--	--	--	--	--	T → boolean	--	--	--	--	--	--	--	T → int	--	--	--	T → string	--	--	--	--
U	--	U → V Y	--	--	--	--	--	--	--	--	U → V Y	U → V Y	--	--	--	U → V Y	--	--	--	--	--	--	--	--	--	--
V	--	V → (E)	--	--	--	--	--	--	--	--	V → cadena	V → entero	--	--	--	V → id N	--	--	--	--	--	--	--	--	--	--
W	W → lambda	--	W → lambda	--	--	W → lambda	W → lambda	--	W → > U W	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
X	--	X → E	--	--	--	--	--	--	--	--	X → E	X → E	--	--	--	X → E	--	--	--	--	--	--	--	--	--	--
Y	Y → lambda	--	Y → lambda	--	Y → + V Y	Y → lambda	Y → lambda	--	Y → lambda	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
Z	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	Z → { C }	--	--

2.4 Explicación del funcionamiento

El Analizador Sintáctico es el núcleo del programa. Desde él se llama tanto al Analizador Léxico como al Analizador Semántico si les toca actuar pasándoles las tablas de símbolos para que interactúen con ellas si es necesario. Principalmente actúa sobre tres métodos:

-algoritmo(): Método correspondiente a la realización del algoritmo descendente por tablas, saltando error si ocurre algún error sintáctico y llamando a los métodos M() y equip() cuando es su turno, no acabando hasta que salta error alguno de los analizadores (devolviendo el error correspondiente y su línea) o hasta que termine el algoritmo sin errores, devolviendo el parser final. Cuando realiza equip() llama al analizador léxico para recibir el siguiente token del programa, y en cada iteración llama al analizador semántico para preguntarle si se tiene que realizar una acción semántica.

-M(parser,pila,token,c): Método encargado de meter en la pila la expresión siguiente al no terminal que encabeza la pila 'c' (eliminando 'c' de la pila), con siguiente token 'token' y añade al parser la regla correspondiente a esa transición (metiendo en una pila auxiliar el no terminal 'c' para el uso del Analizador Semántico).

-equip(pila,token): Método encargado de al encontrase en la cima de la pila un 'token' / símbolo terminal, eliminarlo de la pila (añadiéndolo a la pila auxiliar para el uso del Analizador Semántico).

El Analizador Sintáctico mientras se vaya encontrando en la cima de la pila símbolos no terminales, llamará al método M() para seguir con las reglas hasta dar con el terminal que está buscando si hay buena sintaxis interactuando con una matriz de transiciones 'tabla_sintactico' que incluye qué reglas aplicar dependiendo del token que se busque. Al dar con él, llamará al Analizador Léxico para pedir el siguiente token (y añadiendo este a una lista de tokens para su posterior lectura) requerido para seguir el proceso hasta que la pila se vacíe. Mientras tanto, Cuando lo que haya en la cima de la pila sea una acción semántica, llamará al Analizador Semántico para realizar dicha acción.

3. Diseño del Analizador Semántico

Traducción dirigida por la sintaxis con las acciones semánticas:

```
1.  D ->  { TSGlobal = Crear_TS(); DespG = 0; } P { destruirTS(TSGlobal); }
2.  P ->  B P1              {}
3.  P ->  F P1              {}
4.  P -> lambda             {}
5.  E -> R O                {       if(O.tipo == tipo_ok ) then E.tipo = R.tipo;
                                    else E.tipo = logico;
                                    if(O.tipo != tipo_ok && R.tipo !=entero) then ERROR("EL OPERADOR
NO ES VALIDO");
                                    }
6.  O -> && R O1  {         if(O1.tipo == tipo_ok) then O.tipo = R.tipo;
                                    else if (R.tipo != boolean || O1.tipo != boolean) then ERROR("LOS
TIPOS DEBEN SER BOOLEANOS")
                                    }
7.  O -> lambda             {       O.tipo = vacio;
                                    }
8.  R -> U W                {       if(U.tipo=="entero" && W.tipo=="logico") then R.tipo=W.tipo;
                                    else if(W.tipo == tipo_ok) then R.tipo = U.tipo;
                                    else if (W.tipo != entero || U.tipo != entero) then ERROR("LOS TIPOS
DEBEN SER ENTEROS");
                                    }
9.  W -> > U W1             {       if(U.tipo=="entero" && W.tipo=="logico") then R.tipo=W.tipo;
                                    else if(W.tipo == tipo_ok) then R.tipo = U.tipo;
                                    else if (W.tipo != entero || U.tipo != entero) then ERROR("LOS TIPOS
DEBEN SER ENTEROS");
                                    }
10.  W -> lambda            {       W.tipo = vacio;
                                    }
11. U -> V Y                {       if(Y.tipo == tipo_ok) then U.tipo := V.tipo;
                                    else if (Y.tipo != entero || V.tipo != entero) then ERROR("LOS TIPOS
DEBEN SER ENTEROS");
                                    }
12. Y -> + V Y1             {       if (Y1.tipo != entero && V.tipo != entero) then error ("TIENE   QUE
HABER UN TIPO ENTERO");
                                    Y.tipo = V.tipo;
                                    }

13. Y -> lambda             {       Y.tipo = vacio;

                                    }

14. V -> id N               {       V.tipo = buscaTipoTS(id.pos);

                                    }



15. N -> ( L )              {       N.tipo = L.tipo;

                                    }
```

```
16. N -> lambda          {}      N.tipo = vacio

17. V -> ( E )           {       V.tipo = E.tipo;
                                 }
18. V -> entero          {       V.tipo = entero;
                                 V.tamaño = 1;
                                 }
19. V -> cadena          {       V.tipo = cadena;
                                 V.tamaño = 64;
                                 }
20. S -> id M ;          {       (id.tipo = buscaTipoTS(id.pos);
                                  if (id.tipo == funct) then id.tipoParam = M.tipoParam;
                                 else if(M.tipo==funct)
                                     then if (M.tipoRet==id.tipo)then S.tipo = M.tipo;
                                 else if (id.tipo == null) then ERROR("TIPO DEL ID NO ENCONTRADO");
                                 else if (id.tipo == M.tipo) then S.tipo = M.tipo;
                                 else S.tipo = ERROR;
                         }
21. M -> = E ;           {       M.tipo = E.tipo;
                         }
22. M -> *= E ;          {       M.tipo = E.tipo;
                         }

23. M -> ( L )           {       M.tipo = vacio;
                                 M.tipoParam = L.tipoParam;
                         }
24. S -> put E ;    {    S.tipo = E.tipo;
                                 }
25. S -> get id ;   {    id.tipo = buscaTipoTS(id.pos);
                                 if(id.tipo == null) then ERROR("TIPO DEL ID NO ENCONTRADO");
                                 else S.tipo = id.tipo;
                         }
26. S -> return X ;{     S.tipoRet = X.tipo;
                         }
27. L -> E Q             {       L.tipo = E.tipo * Q.tipo ;
                         }
28. L -> lambda          {       L.tipo = vacio;
                         }
29. Q -> , E Q1     {    if(Q1.tipo != vacio) then Q.tipo = E.tipo * Q1.tipo;
                                 else Q.tipo = E.tipo;
                         }
30. Q -> lambda          {Q.tipo = vacio;}
31. X -> E               {       X.tipo = E.tipo;
                         }
32. B -> if ( E ) S   {  if(E.tipo==logico) then B.tipo=S.tipo ;
                                 else ERROR("LA CONDICION DEL IF DEBE SER BOOLEANA");
                                 }
33. B ->{zonaDec=true;  } let id T ;        {       insertTipoTS(id.pos, T.tipo);
                                                    insertDespTS(id.pos, desp);
                                                    desp = desp + T.tamaño);
```

```
                                                zonaDec = false;
                                                }

34. B -> S                  {       B.tipo = S.tipo;
                                    }
35. B -> for ( id M ; E ; id1 M1 ) Z    {       if(id.tipo == entero && E.tipo == boolean && id1.tipo ==
                            entero && M.tipo==entero && M1.tipo==entero) then B.tipo = Z.tipo;
                                                else ERROR("LA SENTENCIA TIENE QUE SER UN ENTERO Y
                                                LAS EXPRESIONES BOOLEANAS");
                                                }
36. T -> int                {       T.tipo := entero;
                                    T.ancho := 1;
                            }
37. T -> boolean  {         T.tipo := logico; //
                                    T.ancho := 1;
                            }
38. T -> string             {       T.tipo = cadena;
                                    T.ancho = 64;
                            }
39. F -> function{zonaDec = true;} id { añadirEtiquetaTS(id.pos, genEtiq());
                                                añadirTipoTS(id.pos, function);
                                                zonaDec = false;
                                                TSActual = crearTS();Desp = 0;      }
                                                zonaDec = true;
                                    }
                        H ( A )     {       zonaDecActual = false;
                                                setNumParamTSGlobal(id.pos,A.nparam);
                                                setTipoParamTSGlobal(id.pos,A.tipoParam);
                                                setTipoRetTSGlobal(H.tipoRet);
                                    }
                          Z         {       if(Z.tipoRet != H.tipo) then ERROR("INCORRECTO RETORNO");
                                    destruirTS(TSActual);
                                    TSActual := TSGlobal;
                                    }
40. Z -> { C }              {       Z.tipoRet = C.tipo;
                                    }
41. H -> T                  {       H.tipo = T.tipo;
                                    }
42. H -> void               {       H.tipo = void;
                                    }
43. A ->{zonaDec = true   } T  id K
                                    InsertTipoTS(id.pos, T.tipo);
                                    InsertDespTS(id.pos, desp);
                                    desp = desp + T.tamaño;
                                    A.nParam++;
                                    A.tipoParam=A.tipoParam * id.tipo;
                                    zonaDec = false;
                                    }
44. A -> void               {       A.tipo = void;
                                    }
```

```
45. K -> {          zonaDec = true}, T id K
                                    InsertTipoTS(id.pos, T.tipo);
                                    InsertDespTS(id.pos, desp);
                                    desp = desp + T.tamaño;
                                    A.nParam++;
                                    A.tipoParam=A.tipoParam * id.tipo;
                                    zonaDec = false;
                                    }
46. K -> lambda         {}
47. C -> B C1           {           if(B.tipo == tipo_ok && C1.tipo == tipo_ok) then C.tipo = B.tipo ;
                                    else ERROR
                                    }
48. C -> lambda         {           C.tipo = vacio;
                                    }
```

3.1 Explicación del funcionamiento

El analizador semántico es llamado por el analizador sintáctico para procesar las acciones semánticas requeridas por el programa en caso de que se deba procesar alguna. En caso de que haya algún error semántico le devuelve un código de error para llamar al gestor de errores con el sitio específico del error. Principalmente funciona con estos métodos:

genEtiq(): Método utilizado para generar etiquetas aleatorias para las funciones. Estas etiquetas se guardan para que no se vuelvan a repetir en un futuro.

esAccionSemántica(c): Método usado para que el analizador sintáctico pueda detectar si es el momento de realizar alguna acción semántica (mirando si 'c' lo es).

ejecutarAcción(acción,,aux,pts): Método usado por el analizador sintáctico en caso de que se haya detectado una acción semántica. Recibe la 'acción' semántica y mediante un 'switch' en el que están metidas todas las acciones semánticas posibles, realiza el código relacionado con esa acción semántica modificando la pila auxiliar 'aux' pasada por el analizador sintáctico y realiza las modificaciones en la pila de tablas 'pts' pasada por el analizador sintáctico. Al terminar el código relacionado con la acción semántica correspondiente, el método devuelve una estructura de datos 'PilaAux_TS', cuyo contenido es la 'aux' modificada y la pila tablas 'pts' modificada.

4. Diseño de la Tabla de Símbolos

La tabla de símbolos es una estructura de datos donde almacenamos la información necesaria sobre los identificadores que se declaran y el desplazamiento de la Tabla de Símbolos. Cualquier parte del procesador tiene acceso a ella estando compuesta por una entrada por identificador y sus atributos, pudiendo obtenerlos y añadirlos cuando lo requiramos.

Principalmente está formada por una 'LinkedList<ID> tabla' siendo ID una estructura de datos creada con los atributos de los ID, haciendo que su posición en la tabla de símbolos sea su posición en la tabla.

Tiene una variable 'zonaDec' para saber si estamos en zona declarativa, una variable 'Global' para saber si es la tabla global y una variable 'actual' para saber si la TS es la TS actual sobre la que se trabaja.

Las tablas de símbolos serán guardadas en una PILA_TABLAS() donde el primer elemento de esa pila será la TSGlobal.

5. Diseño del Gestor de Errores

El gestor de errores consiste en una clase concreta para el gestor, y dependiendo de si se origina un error en el Analizador Léxico, Sintáctico o Semántico se llamará a un método distinto de la propia clase gestor_error. Los distintos errores tienen un código concreto y se le envía al gestor tanto el código del error, como datos necesarios para facilitar la comprensión del error (como el lexema mal escrito, lo que esperaba el lenguaje encontrar) incluyendo la línea en la que ocurre dicho error.

6. Casos de prueba

6.2 Casos de prueba sin errores

Hay adjuntada una carpeta llamada ‘casos de prueba’ con todos los casos con sus tokens, parse, tablas de símbolos, errores y el árbol sintáctico de cada caso de acierto para mejor visualización.

********** caso 1 **********

```
let a int;
a = 5;
let b int;
b = 6;
let c int;
c = 7;

function sumar int (int x, int y){
                                        return x+y ;}
let aa int ;
aa = sumar(a,b);

function sumar2 int(int x, int y) {    let z int ;
                                                let c string;
                                                c = 'molesto';
                                                z = x+y;
                                                return z ;
        }
let b_b int;
b_b = sumar2(aa,a);
```

TOKENS:

```
<let,>
<ID,0>
<int,>
<puntoycoma,>

<ID,0>
<igual,>
<constEnt,5>
<puntoycoma,>

<let,>
```

<ID,1>
<int,>
<puntoycoma,>

<ID,1>
<igual,>
<constEnt,6>
<puntoycoma,>

<let,>
<ID,2>
<int,>
<puntoycoma,>

<ID,2>
<igual,>
<constEnt,7>
<puntoycoma,>

<function,>
<ID,3>
<int,>
<abrirParentesis,>
<int,>
<ID,0>
<coma,>
<int,>
<ID,1>
<cerrarParentesis,>
<abrirLlave,>

<return,>
<ID,0>
<mas,>
<ID,1>
<puntoycoma,>
<cerrarLlave,>

<let,>
<ID,4>
<int,>
<puntoycoma,>

<ID,4>
<igual,>
<ID,3>
<abrirParentesis,>
<ID,0>
<coma,>
<ID,1>
<cerrarParentesis,>
<puntoycoma,>

<function,>
<ID,5>
<int,>
<abrirParentesis,>
<int,>
<ID,0>
<coma,>
<int,>
<ID,1>
<cerrarParentesis,>
<abrirLlave,>
<let,>
<ID,2>
<int,>
<puntoycoma,>

<let,>
<ID,3>
<string,>
<puntoycoma,>

<ID,3>

<igual,>

<cadena,'molesto'>

<puntoycoma,>

<ID,2>

<igual,>

<ID,0>

<mas,>

<ID,1>

<puntoycoma,>

<return,>

<ID,2>

<puntoycoma,>

<cerrarLlave,>

<let,>

<ID,6>

<int,>

<puntoycoma,>

<ID,6>

<igual,>

<ID,5>

<abrirParentesis,>

<ID,4>

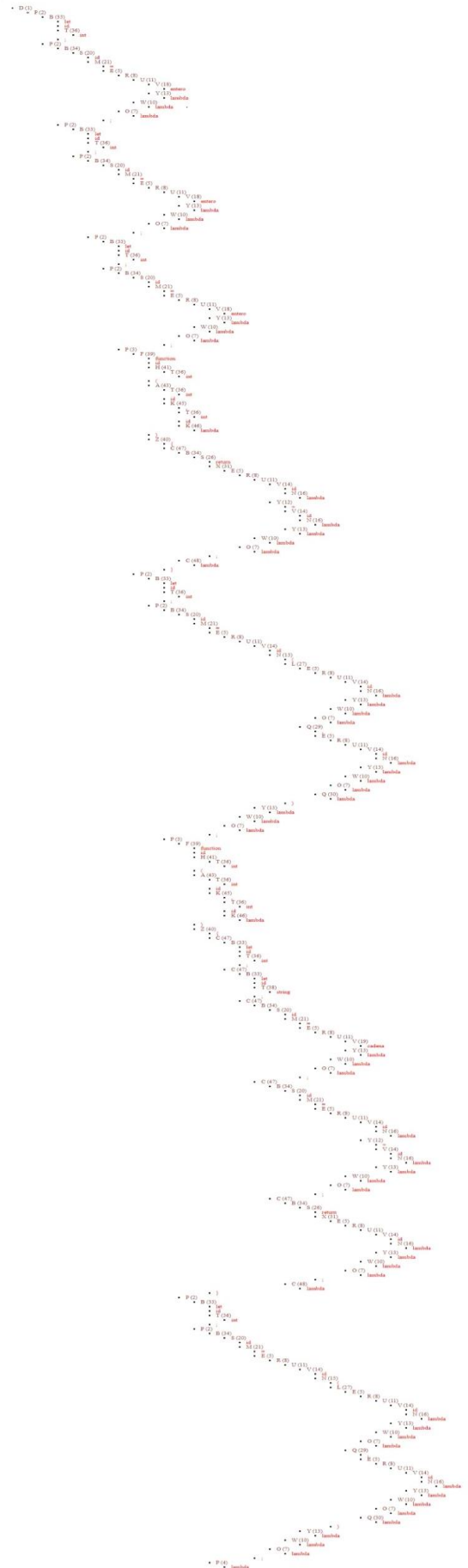
<coma,>

<ID,0>

<cerrarParentesis,>

<puntoycoma,>

ARBOL SINTACTICO:



TABLAS:

CONTENIDO TS *GLOBAL # 1

CONTENIDO TS GLOBAL # 1:

* LEXEMA: 'a'

Atributos:

+ Tipo: 'entero'

+ Despl: 0

--------- ----------

* LEXEMA: 'b'

Atributos:

+ Tipo: 'entero'

+ Despl: 1

--------- ----------

* LEXEMA: 'c'

Atributos:

+ Tipo: 'entero'

+ Despl: 2

--------- ----------

* LEXEMA: 'sumar'

Atributos:

+ Tipo: 'funct'

+ NumParam: 2

+ TipoParam01: 'entero'

+ TipoParam02: 'entero'

+ TipoRetorno: 'entero'

+ EtiqFuncion: 'Etiqueta_3'

--------- ----------

* LEXEMA: 'aa'

Atributos:

+ Tipo: 'entero'

+ Despl: 3

--------- ----------

* LEXEMA: 'sumar2'

Atributos:

+ Tipo: 'funct'

+ NumParam: 2

+ TipoParam01: 'entero'

+ TipoParam02: 'entero'

+ TipoRetorno: 'entero'

+ EtiqFuncion: 'Etiqueta_1'

--------- ----------

* LEXEMA: 'b_b'

Atributos:

+ Tipo: 'entero'

+ Despl: 4

CONTENIDO TS sumar # 2:

* LEXEMA: 'x'

Atributos:

+ Tipo: 'entero'

+ Despl: 0

--------- ----------

* LEXEMA: 'y'

Atributos:

+ Tipo: 'entero'

+ Despl: 1

CONTENIDO TS sumar2 # 3:

* LEXEMA: 'x'

Atributos:

+ Tipo: 'entero'

+ Despl: 0

--------- ----------

* LEXEMA: 'y'

Atributos:

+ Tipo: 'entero'

+ Despl: 1

--------- ----------

* LEXEMA: 'z'

Atributos:

+ Tipo: 'entero'

+ Despl: 2

--------- ----------

* LEXEMA: 'c'

Atributos:

+ Tipo: 'cadena'

+ Despl: 3

********** caso 2 **********

```
let t string;
        t = 'texto';

        function print string ( string var) {
                                                let c string; c = var;          return var;}
let a int ; a = 4;
let b int; b = 3;
let c int; c = 2;
        if(b>c && (a>b)) return a;


        function mult int (int x, int y) {      let z int;
                                                z *= x;
                                                return z;}
```

TOKENS:

<let,>
<ID,0>
<string,>
<puntoycoma,>

<ID,0>
<igual,>
<cadena,'texto'>
<puntoycoma,>

<function,>
<ID,1>
<string,>
<abrirParentesis,>
<string,>
<ID,0>
<cerrarParentesis,>
<abrirLlave,>

<let,>

<ID,1>

<string,>

<puntoycoma,>

<ID,1>

<igual,>

<ID,0>

<puntoycoma,>

<return,>

<ID,0>

<puntoycoma,>

<cerrarLlave,>

<let,>

<ID,2>

<int,>

<puntoycoma,>

<ID,2>

<igual,>

<constEnt,4>

<puntoycoma,>

<let,>

<ID,3>

<int,>

<puntoycoma,>

<ID,3>

<igual,>

<constEnt,3>

<puntoycoma,>

<let,>

<ID,4>

<int,>

<puntoycoma,>

<ID,4>

<igual,>

<constEnt,2>

<puntoycoma,>

<if,>
<abrirParentesis,>
<ID,3>
<mayorQ,>
<ID,4>
<and,>
<abrirParentesis,>
<ID,2>
<mayorQ,>
<ID,3>
<cerrarParentesis,>
<cerrarParentesis,>
<return,>
<ID,2>
<puntoycoma,>

<function,>
<ID,5>
<int,>
<abrirParentesis,>
<int,>
<ID,0>
<coma,>
<int,>
<ID,1>
<cerrarParentesis,>
<abrirLlave,>
<let,>
<ID,2>
<int,>
<puntoycoma,>

<ID,2>
<opMult,>
<ID,0>

<puntoycoma,>

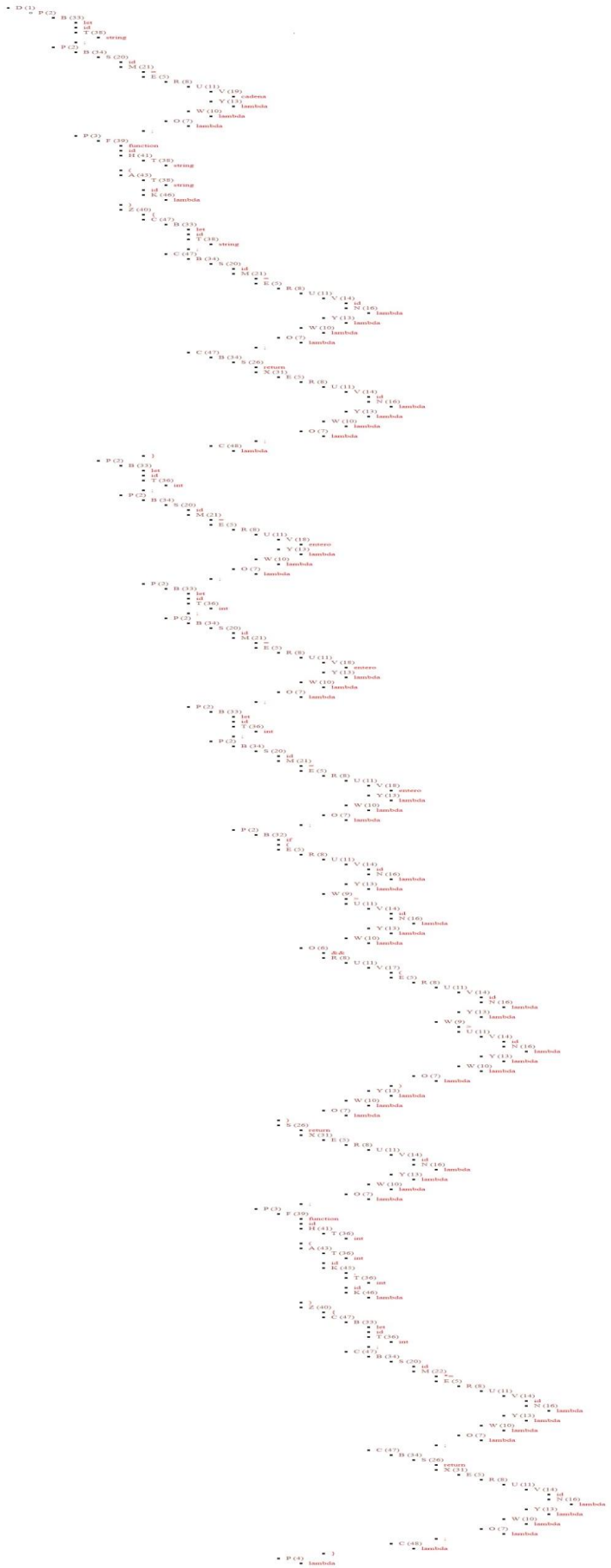
<return,>

<ID,2>

<puntoycoma,>

<cerrarLlave,>

ARBOL SINTACTICO:



TABLAS:

CONTENIDO TS GLOBAL # 1:

* LEXEMA: 't'

Atributos:

+ Tipo: 'cadena'

+ Despl: 0

--------- ----------

* LEXEMA: 'print'

Atributos:

+ Tipo: 'funct'

+ NumParam: 1

+ TipoParam01: 'cadena'

+ TipoRetorno: 'cadena'

+ EtiqFuncion: 'Etiqueta_1'

--------- ----------

* LEXEMA: 'a'

Atributos:

+ Tipo: 'entero'

+ Despl: 64

--------- ----------

* LEXEMA: 'b'

Atributos:

+ Tipo: 'entero'

+ Despl: 65

--------- ----------

* LEXEMA: 'c'

Atributos:

+ Tipo: 'entero'

+ Despl: 66

--------- ----------

* LEXEMA: 'mult'

Atributos:

+ Tipo: 'funct'

+ NumParam: 2

+ TipoParam01: 'entero'

+ TipoParam02: 'entero'

+ TipoRetorno: 'entero'

+ EtiqFuncion: 'Etiqueta_3'

CONTENIDO TS print # 2:

* LEXEMA: 'var'

Atributos:

+ Tipo: 'cadena'

+ Despl: 0

--------- ----------

* LEXEMA: 'c'

Atributos:

+ Tipo: 'cadena'

+ Despl: 64

CONTENIDO TS mult # 3:

* LEXEMA: 'x'

Atributos:

+ Tipo: 'entero'

+ Despl: 0

--------- ----------

* LEXEMA: 'y'

Atributos:

+ Tipo: 'entero'

+ Despl: 1

--------- ----------

* LEXEMA: 'z'

Atributos:

+ Tipo: 'entero'

+ Despl: 2

********** caso 3 **********

```
let bol boolean;
/*comentar
        io*/
let v int;
        v = 0;
let w int;
        w = 1;
let i int;
for(i = 0; 5>i;i=i+1) {
                v = v +2;
        if(bol && v>6) put v; }

let contador int;
function potencia_3 int (int x) {
                                        if( 3 > contador )
                                        contador = contador+1;
                x *= potencia_3(x);
                                        return x;
                                        }
```

TOKENS:

<let,>
<ID,0>
<boolean,>
<puntoycoma,>

<let,>
<ID,1>
<int,>
<puntoycoma,>

<ID,1>
<igual,>
<constEnt,0>
<puntoycoma,>

<let,>

<ID,2>

<int,>

<puntoycoma,>

<ID,2>

<igual,>

<constEnt,1>

<puntoycoma,>

<let,>

<ID,3>

<int,>

<puntoycoma,>

<for,>

<abrirParentesis,>

<ID,3>

<igual,>

<constEnt,0>

<puntoycoma,>

<constEnt,5>

<mayorQ,>

<ID,3>

<puntoycoma,>

<ID,3>

<igual,>

<ID,3>

<mas,>

<constEnt,1>

<cerrarParentesis,>

<abrirLlave,>

<ID,1>

<igual,>

<ID,1>

<mas,>
<constEnt,2>
<puntoycoma,>

<if,>
<abrirParentesis,>
<ID,0>
<and,>
<ID,1>
<mayorQ,>
<constEnt,6>
<cerrarParentesis,>
<put,>
<ID,1>
<puntoycoma,>
<cerrarLlave,>

<let,>
<ID,4>
<int,>
<puntoycoma,>

<function,>
<ID,5>
<int,>
<abrirParentesis,>
<int,>
<ID,0>
<cerrarParentesis,>
<abrirLlave,>

<if,>
<abrirParentesis,>
<constEnt,3>
<mayorQ,>
<ID,4>
<cerrarParentesis,>

<ID,4>

<igual,>

<ID,4>

<mas,>

<constEnt,1>

<puntoycoma,>

<ID,0>

<opMult,>

<ID,5>

<abrirParentesis,>

<ID,0>

<cerrarParentesis,>

<puntoycoma,>

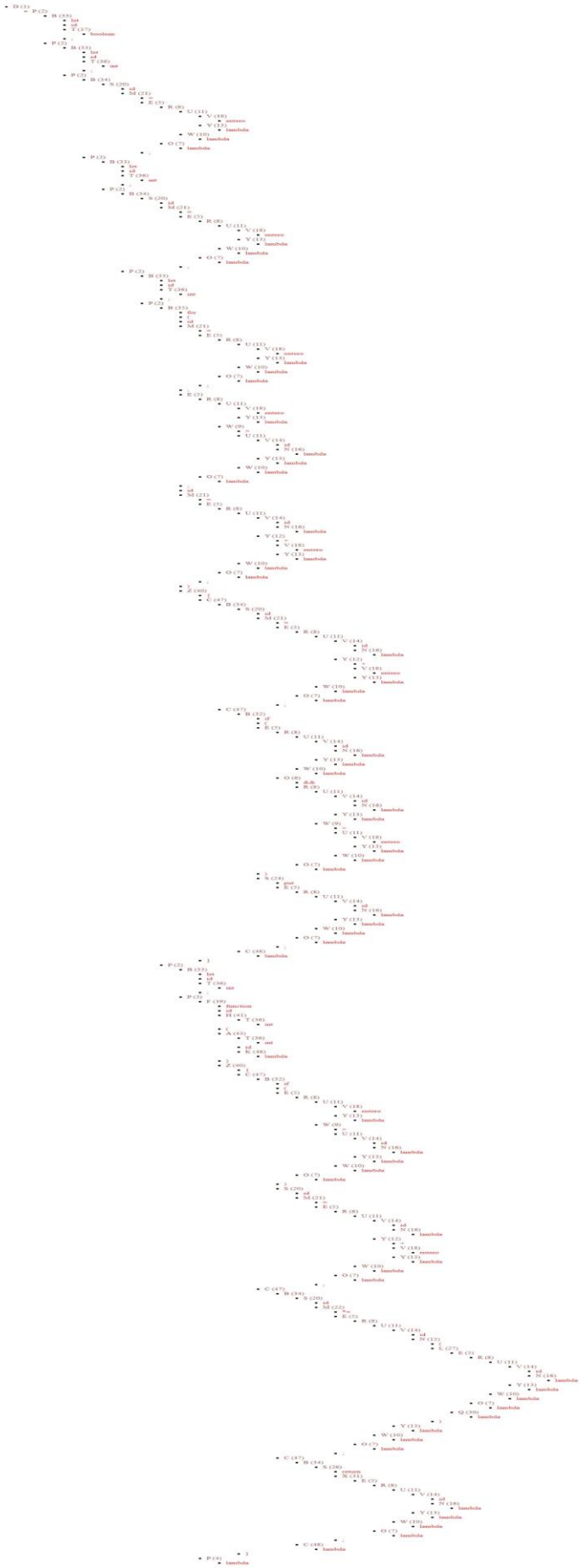
<return,>

<ID,0>

<puntoycoma,>

<cerrarLlave,>

ARBOL SINTACTICO:



TABLAS:

CONTENIDO TS GLOBAL # 1:

* LEXEMA: 'bol'

Atributos:

+ Tipo: 'logico'

+ Despl: 0

--------- ----------

* LEXEMA: 'v'

Atributos:

+ Tipo: 'entero'

+ Despl: 1

--------- ----------

* LEXEMA: 'w'

Atributos:

+ Tipo: 'entero'

+ Despl: 2

--------- ----------

* LEXEMA: 'i'

Atributos:

+ Tipo: 'entero'

+ Despl: 3

--------- ----------

* LEXEMA: 'contador'

Atributos:

+ Tipo: 'entero'

+ Despl: 4

--------- ----------

* LEXEMA: 'potencia_3'

Atributos:

+ Tipo: 'funct'

+ NumParam: 1

+ TipoParam01: 'entero'

+ TipoRetorno: 'entero'

+ EtiqFuncion: 'Etiqueta_9'

CONTENIDO TS potencia_3 # 2:

* LEXEMA: 'x'

Atributos:

+ Tipo: 'entero'

+ Despl: 0

********** caso 4 **********

```
let c string ;
let i int;

        for(i = 0; 15 >i; i=i+1) {

                                c = 'hola';         }

let var1 boolean;
let var2 boolean;
        if(var1 && var2) c = 'adios';

function NadaDeNada void (void) { let c string;
                                    c = 'esta funcion no sirve de nada' ;
                                   }
NadaDeNada();

function HaceAlgo void (void) { c = 'hace algo'; }
function HaceAlgo2 void (string var) { var = 'esta funcion tambien';}
```

TOKENS:

```
<let,>
<ID,0>
<string,>
<puntoycoma,>

<let,>
<ID,1>
<int,>
<puntoycoma,>

<for,>
<abrirParentesis,>
<ID,1>
<igual,>
<constEnt,0>
```

<puntoycoma,>

<constEnt,15>

<mayorQ,>

<ID,1>

<puntoycoma,>

<ID,1>

<igual,>

<ID,1>

<mas,>

<constEnt,1>

<cerrarParentesis,>

<abrirLlave,>

<ID,0>

<igual,>

<cadena,'hola'>

<puntoycoma,>

<cerrarLlave,>

<let,>

<ID,2>

<boolean,>

<puntoycoma,>

<let,>

<ID,3>

<boolean,>

<puntoycoma,>

<if,>

<abrirParentesis,>

<ID,2>

<and,>

<ID,3>

<cerrarParentesis,>

<ID,0>

<igual,>

<cadena,'adios'>
<puntoycoma,>

<function,>
<ID,4>
<void,>
<abrirParentesis,>
<void,>
<cerrarParentesis,>
<abrirLlave,>
<let,>
<ID,0>
<string,>
<puntoycoma,>

<ID,0>
<igual,>
<cadena,'esta funcion no sirve de nada'>
<puntoycoma,>

<cerrarLlave,>

<ID,4>
<abrirParentesis,>
<cerrarParentesis,>
<puntoycoma,>

<function,>
<ID,5>
<void,>
<abrirParentesis,>
<void,>
<cerrarParentesis,>
<abrirLlave,>
<ID,0>
<igual,>
<cadena,'hace algo'>

<puntoycoma,>

<cerrarLlave,>

<function,>

<ID,6>

<void,>

<abrirParentesis,>

<string,>

<ID,0>

<cerrarParentesis,>

<abrirLlave,>

<ID,0>

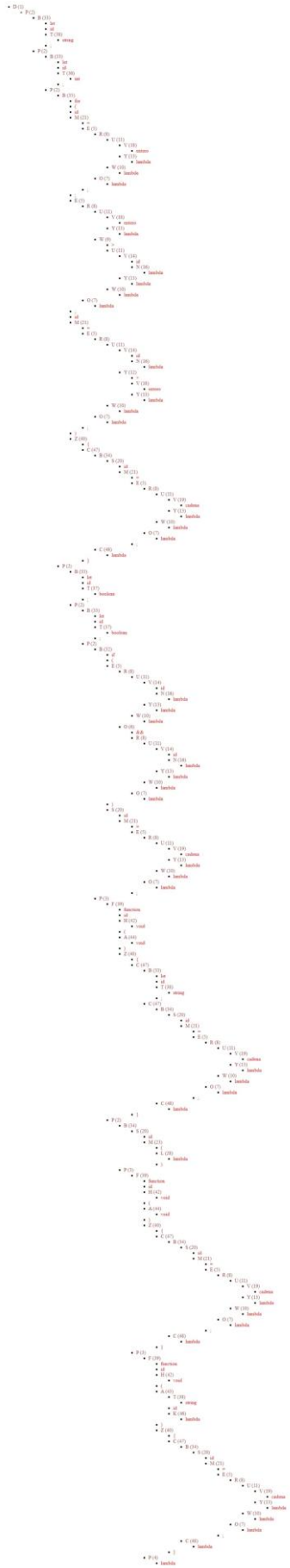
<igual,>

<cadena,'esta funcion tambien'>

<puntoycoma,>

<cerrarLlave,>

ARBOL SINTACTICO:



TABLAS:

CONTENIDO TS GLOBAL # 1:

* LEXEMA: 'c'

Atributos:

+ Tipo: 'cadena'

+ Despl: 0

--------- ----------

* LEXEMA: 'i'

Atributos:

+ Tipo: 'entero'

+ Despl: 64

--------- ----------

* LEXEMA: 'var1'

Atributos:

+ Tipo: 'logico'

+ Despl: 65

--------- ----------

* LEXEMA: 'var2'

Atributos:

+ Tipo: 'logico'

+ Despl: 66

--------- ----------

* LEXEMA: 'NadaDeNada'

Atributos:

+ Tipo: 'funct'

+ NumParam: 0

+ TipoRetorno: 'vacio'

+ EtiqFuncion: 'Etiqueta_8'

--------- ----------

* LEXEMA: 'HaceAlgo'

Atributos:

+ Tipo: 'funct'

+ NumParam: 0

+ TipoRetorno: 'vacio'

+ EtiqFuncion: 'Etiqueta_1'

--------- ----------

* LEXEMA: 'HaceAlgo2'

Atributos:

+ Tipo: 'funct'

+ NumParam: 1

+ TipoParam01: 'cadena'

+ TipoRetorno: 'vacio'

+ EtiqFuncion: 'Etiqueta_7'

CONTENIDO TS NadaDeNada # 2:

* LEXEMA: 'c'

Atributos:

+ Tipo: 'cadena'

+ Despl: 0

CONTENIDO TS HaceAlgo # 3:

CONTENIDO TS HaceAlgo2 # 4:

* LEXEMA: 'var'

Atributos:

+ Tipo: 'cadena'

+ Despl: 0

********** caso 5 **********

```
let a boolean;

function esTrue boolean(boolean c){ return c;}

function sumar int (int x , int y){
                                        return x+y;}
let b int;

function sumar2 int (int x, int y){ let z int;
                                    let i int;
                                    for(i=0;5>i;i=i+1){

                                        z = sumar(x,y);
                }
                        return sumar(x,y);
                                        }
```

TOKENS:

<let,>

<ID,0>

<boolean,>

<puntoycoma,>

<function,>

<ID,1>

<boolean,>

<abrirParentesis,>

<boolean,>

<ID,0>

<cerrarParentesis,>

<abrirLlave,>

<return,>

<ID,0>

<puntoycoma,>

<cerrarLlave,>

<function,>
<ID,2>
<int,>
<abrirParentesis,>
<int,>
<ID,0>
<coma,>
<int,>
<ID,1>
<cerrarParentesis,>
<abrirLlave,>

<return,>
<ID,0>
<mas,>
<ID,1>
<puntoycoma,>
<cerrarLlave,>

<let,>
<ID,3>
<int,>
<puntoycoma,>

<function,>
<ID,4>
<int,>
<abrirParentesis,>
<int,>
<ID,0>
<coma,>
<int,>
<ID,1>
<cerrarParentesis,>
<abrirLlave,>
<let,>
<ID,2>

<int,>
<puntoycoma,>

<let,>
<ID,3>
<int,>
<puntoycoma,>

<for,>
<abrirParentesis,>
<ID,3>
<igual,>
<constEnt,0>
<puntoycoma,>
<constEnt,5>
<mayorQ,>
<ID,3>
<puntoycoma,>
<ID,3>
<igual,>
<ID,3>
<mas,>
<constEnt,1>
<cerrarParentesis,>
<abrirLlave,>

<ID,2>
<igual,>
<ID,2>
<abrirParentesis,>
<ID,0>
<coma,>
<ID,1>
<cerrarParentesis,>
<puntoycoma,>

<cerrarLlave,>

<return,>

<ID,2>

<abrirParentesis,>

<ID,0>

<coma,>

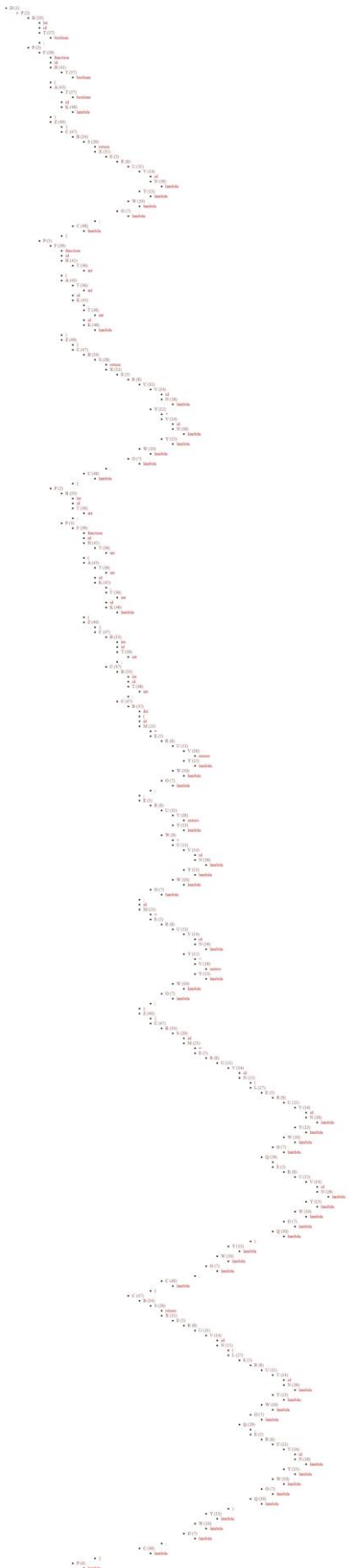
<ID,2>

<cerrarParentesis,>

<puntoycoma,>

<cerrarLlave,>

ARBOL SINTACTICO:



TABLAS:

CONTENIDO TS GLOBAL # 1:

* LEXEMA: 'a'

Atributos:

+ Tipo: 'logico'

+ Despl: 0

--------- ----------

* LEXEMA: 'esTrue'

Atributos:

+ Tipo: 'funct'

+ NumParam: 1

+ TipoParam01: 'logico'

+ TipoRetorno: 'logico'

+ EtiqFuncion: 'Etiqueta_6'

--------- ----------

* LEXEMA: 'sumar'

Atributos:

+ Tipo: 'funct'

+ NumParam: 2

+ TipoParam01: 'entero'

+ TipoParam02: 'entero'

+ TipoRetorno: 'entero'

+ EtiqFuncion: 'Etiqueta_1'

--------- ----------

* LEXEMA: 'b'

Atributos:

+ Tipo: 'entero'

+ Despl: 1

--------- ----------

* LEXEMA: 'sumar2'

Atributos:

+ Tipo: 'funct'

+ NumParam: 2

+ TipoParam01: 'entero'

+ TipoParam02: 'entero'

+ TipoRetorno: 'entero'

+ EtiqFuncion: 'Etiqueta_3'

CONTENIDO TS esTrue # 2:

* LEXEMA: 'c'

Atributos:

+ Tipo: 'logico'

+ Despl: 0

CONTENIDO TS sumar # 3:

* LEXEMA: 'x'

Atributos:

+ Tipo: 'entero'

+ Despl: 0

--------- ----------

* LEXEMA: 'y'

Atributos:

+ Tipo: 'entero'

+ Despl: 1

CONTENIDO TS sumar2 # 4:

* LEXEMA: 'x'

Atributos:

+ Tipo: 'entero'

+ Despl: 0

--------- ----------

* LEXEMA: 'y'

Atributos:

+ Tipo: 'entero'

+ Despl: 1

--------- ----------

* LEXEMA: 'z'

Atributos:

+ Tipo: 'entero'

+ Despl: 2

--------- ----------

* LEXEMA: 'i'

Atributos:

+ Tipo: 'entero'

+ Despl: 3

6.2 Casos de prueba con errores

********** caso 1 **********

```
let a int;
a = 5;
let b int;
b = 6;
let 2c int; /*error declarando variable con numero delante*/
c = 7;

function sumar int (int x, int y){
                                        return x+y ;}
let aa int ;
aa = sumar(a,b);

function sumar2 int(int x, int y) {    let z int ;
                                                let c string;
                                                c = 'molesto';
                                                z = x+y;
                                                return z ;
        }
let bb int;
bb = sumar2(aa,a);
```

ERROR LINEA 5, NO ESTA PERMITIDA LA SINTAXIS DEL LEXEMA EN ESTE LENGUAJE, LAS VARIABLES NO SE PUEDEN DECLARAR CON UN NUMERO AL PRINCIPO

********** caso 2 **********

```
let t string;
        t = 'texto';
        /*mala sintaxis declarando funcion*/
        function string print ( string var) {
                                                let c string; c = var;          return var;}
let a int ; a = 4;
let b int; b = 3;
let c int; c = 2;
        if(b>c && a > c) return a;


        function mult int (int x, int y) {      let z int;
                                                z *= x;
                                                return z;
```

ERROR LINEA 3, ERROR SINTACTICO , ESPERADO EL TERMINAL 'ID' EN VEZ DEL ENCONTRADO 'string'

********** caso 3 **********

```
let bol boolean;
/*comentar
        io*/
let v int;
        v = 0;
let w int;
        w = 1;
let i int;
for(i = 0; 5>i;i=i+1) {
                v = v +2;
 /* esperado && */
        if(bol &* v>6) put v;
         }
```

ERROR LINEA 12, NO ESTA PERMITIDA LA SINTAXIS DEL LEXEMA &* EN ESTE LENGUAJE, ESPERADO COMO LEXEMA &&

********** caso 4 **********

```
let i int;
let c int;
        for(i = 0; 15>i; i=i+1) {
                /*c tipo distinto */
                        c = 'hola';         }

let var1 boolean;
let var2 boolean;
        if(var1 && var2) c = 'adios';
```

ERROR SEMANTICO LINEA 5, NO COINCIDE EL TIPO INTENTADO ASIGNAR A c CON SU TIPO

********** caso 5 **********

```
let a boolean;
let b int;
let c int;
function sumar int (int x , int y){
                                        return x+y;}
let z int;
        /*parametro de funcion erroneo, a es logico, necesita un entero*/
        z = sumar(b,a);
```

ERROR SEMANTICO LINEA 8,LOS PARAMETROS DE LA FUNCION sumar NO COINCIDEN CON LOS DECLARADOS