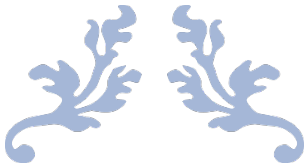


MEMORIA DE PRÁCTICA: CREACIÓN DE UN PROCESADOR

Procesadores de lenguajes

GRUPO 76
GARCÍA TEJADA, LORETO
GUTIÉRREZ MARTÍN, MARÍA

Índice

Analizador léxico

Introducción

Para la realización del Procesador, tenemos las siguientes características:

- Sentencias: Sentencia repetitiva (while)
- Operadores especiales: Asignación con y lógico (&=)
- Técnicas de Análisis Sintáctico: Ascendente LR
- Comentarios: Comentario de bloque (/* */)
- Cadenas: Con comillas dobles (" ")

Definición de tokens

Elemento	Código	Atributo
Boolean	BOOLEAN	
(	parentA	
)	parentC	
{	corcheteA	
}	corcheteC	
;	puntcoma	
,	coma	
=	igual	
&=	asig	
Identificador	id	número
Cadena	cadena	cadena(“c”)
Constante entera	entero	número
function	FUNCTION	
if	IF	
input	INPUT	
int	INT	
print	PRINT	
return	RETURN	
string	STRING	
var	VAR	
while	WHILE	
‘+’	suma	
||	or	
<	menor	

Gramática regular

S->delS | lA| dB | = | < | |C | + | &D | ce | “E

A -> lA | dA | _A | λ

B -> dB | λ

C -> |

D -> =

E -> cE | “

l= ({a_z},{A_Z})

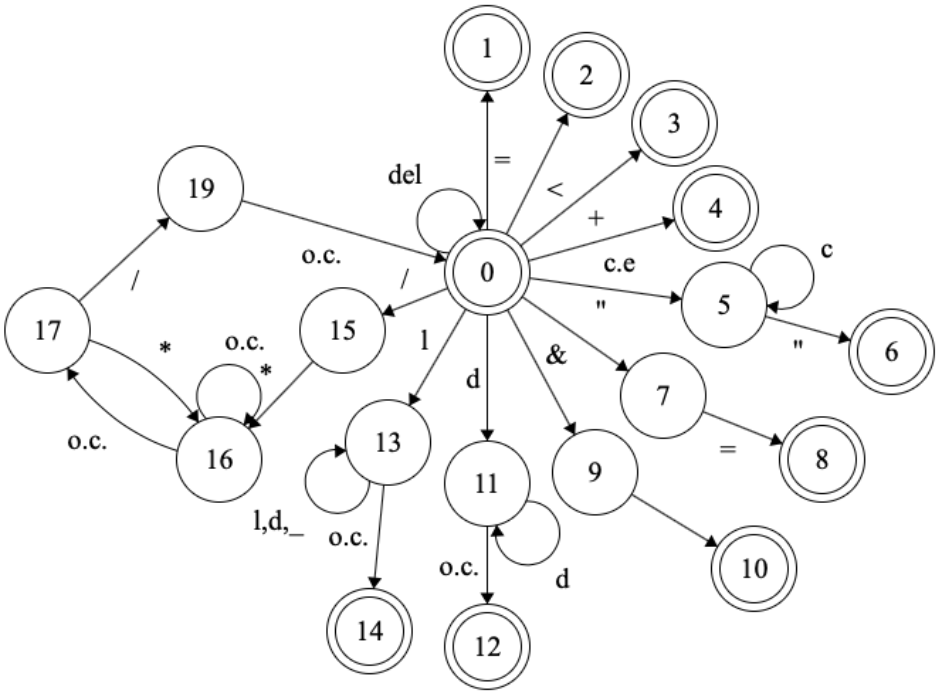
d = {0_9}

c = cualquier carácter excepto “

ce = { ( ) { } , ; }

del = { b, \n, \t, <eof> }

Autómata finito determinista



Nota: Cualquier transición que no esté contemplada en el autómata, generará un error.
o.c. = otro carácter

Acciones semánticas y errores

Letra	Transición de estado	Acción semántica
A	0:0	Leer
B	0:1	Leer; GenToken (igual,_)
C	0:2	Leer; GenToken (menor,_)
D	0:3	Leer; GenToken (suma,_)
E	0:4	Leer; GenToken (c.e.,_)
F	0:5	lexema = “; cont = 0; Leer
G	5:5	concat (lexema, c); cont++; Leer
H	5:6	concat (lexema, “); if cont < 64 then GenToken (cadena, lexema) else error (“Cadena de longitud incorrecta”); Leer
I	0:7	Leer
J	7:8	Leer; GenToken (asig,_)
K	0:9	Leer
L	9:10	Leer; GenToken (or,_)
M	0:11	valor = valorASCII(d); Leer
N	11:11	valor = valor *10 + valorASCII(d); Leer
O	11:12	Leer; If (valor<32767) then GenToken (entero, valor) else error (“Entero mayor de 32767”)
P	0:13	lexema = l; Leer
Q	13:13	concat (lexema, l/d/_); Leer
R	13:14	if lexema ∋ tabla_palabrasClave then GenToken (lexema,_) else p:= buscarTS (lexema){ if p == null then p = insertarTS (lexema)} GenToken (id,p)
T	0:15	Leer
U	15:16	Leer
V	16:16	Leer
W	16:17	Leer
X	17:16	Leer
Y	17:18	Leer
Z	18:0	Leer

Matriz de transición

	Del		L		D		=		<		|		+		&		Ce		“		C		o.c		*		/	
0	0	A	14	Q	12	N	1	B	2	C	10	L	3	D	8	J	4	E	6	G							16	T
1																												
2																												
3																												
4																												
5																												
6																			7	I	6	H						
7																												
8							9	K																				
9																												
10											11	M																
11																												
12					12	O																	13	P				
13																												
14			14(_)	R	14	R																	15	S				
15																												
16																									17	U		
17																							17	V	18	W		
18																							17	X			19	Y
19																							0	Z				

Diseño de tabla de símbolos

El diseño de nuestra tabla de símbolos se corresponde con el **diseño completo** que se requiere en la documentación (exceptuando el modo de los parámetros, si son por referencia o valor).

Por ejemplo:

TABLA PRINCIPAL #1:
* LEXEMA: 'a'
    +tipo:'entero'
    +despl: 0
* LEXEMA: 'b'
    +tipo:'entero'
    +despl: 2
* LEXEMA: 'suma'
    +tipo: 'funcion'
    + numParam: 2
    + TipoParam1: 'entero'
    + TipoParam2: 'entero'
    + tiporetorno: 'entero'

TABLA de la FUNCION suma #2:
* LEXEMA: 'num1'
    +tipo: 'entero'
    +despl: 0
* LEXEMA: 'num2'
    +tipo: 'entero'
    +despl: 2

Analizador sintáctico

Gramática del sintáctico

Terminales = { var int string boolean entero function if while return { } ( ) id + || < cadena print input lambda = &= ; , }

NoTerminales = { P D T F T1 A A1 C S L L1 X SC E U V G }

Axioma = P

Producciones = {

P -> D P
P -> F P
P -> SC P
P -> lambda
D -> var T id ;
T -> int
T -> string
T -> boolean
F -> function T1 id (A) { C }
T1 -> T
T1 -> lambda
A -> lambda
A -> T id A1
A1 -> lambda
A1 -> , T id A1
C -> D C
C -> SC C
C -> lambda
S -> id = E ;
S -> id &= E ;
S -> print (E) ;
V -> (E)
V -> id (L)
S -> input (id) ;
S -> id (L) ;
S -> return X ;
L -> lambda
L -> E L1
L1 -> lambda
L1 -> , E L1
X -> E
X -> lambda
SC -> while (E) { C }
SC -> if (E) S
SC -> S
E -> E || G
E -> G
G -> G < U
G -> U
U -> U + V
U -> V
V -> id
V -> entero
V -> cadena

}

Autómata reconocedor de prefijos variables

state 0
(0) S' -> . P
(1) P -> . D P
(2) P -> . F P
(3) P -> . SC P
(4) P -> . empty
(5) D -> . VAR T id puntcoma
(9) F -> . FUNCTION T1 id parentA A parentC corcheteA C corcheteC
(31) SC -> . WHILE parentA E parentC corcheteA C corcheteC
(32) SC -> . IF parentA E parentC S
(33) SC -> . S

(45) empty -> .
(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma
(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 1
(31) SC -> WHILE . parentA E parentC corcheteA C corcheteC

state 2
(21) S -> PRINT . parentA E parentC puntcoma

state 3
(24) S -> RETURN . X puntcoma
(29) X -> . E
(30) X -> . empty
(34) E -> . E or G
(35) E -> . G
(45) empty -> .
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 4
(19) S -> id . igual E puntcoma
(20) S -> id . asig E puntcoma
(23) S -> id . parentA L parentC puntcoma

state 5
(4) P -> empty .

state 6
(9) F -> FUNCTION . T1 id parentA A parentC corcheteA C corcheteC
(10) T1 -> . T
(11) T1 -> . empty
(6) T -> . INT
(7) T -> . STRING
(8) T -> . BOOLEAN
(45) empty -> .

state 7
(1) P -> D . P
(1) P -> . D P
(2) P -> . F P
(3) P -> . SC P
(4) P -> . empty
(5) D -> . VAR T id puntcoma
(9) F -> . FUNCTION T1 id parentA A parentC corcheteA C corcheteC
(31) SC -> . WHILE parentA E parentC corcheteA C corcheteC
(32) SC -> . IF parentA E parentC S
(33) SC -> . S
(45) empty -> .
(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma
(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 8
(2) P -> F . P
(1) P -> . D P
(2) P -> . F P
(3) P -> . SC P
(4) P -> . empty
(5) D -> . VAR T id puntcoma
(9) F -> . FUNCTION T1 id parentA A parentC corcheteA C corcheteC
(31) SC -> . WHILE parentA E parentC corcheteA C corcheteC
(32) SC -> . IF parentA E parentC S
(33) SC -> . S
(45) empty -> .

(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma
(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 9
(0) S' -> P .

state 10
(33) SC -> S .

state 11
(5) D -> VAR . T id puntcoma
(6) T -> . INT
(7) T -> . STRING
(8) T -> . BOOLEAN

state 12
(22) S -> INPUT . parentA E parentC puntcoma

state 13
(32) SC -> IF . parentA E parentC S

state 14
(3) P -> SC . P
(1) P -> . D P
(2) P -> . F P
(3) P -> . SC P
(4) P -> . empty
(5) D -> . VAR T id puntcoma
(9) F -> . FUNCTION T1 id parentA A parentC corcheteA C corcheteC
(31) SC -> . WHILE parentA E parentC corcheteA C corcheteC
(32) SC -> . IF parentA E parentC S
(33) SC -> . S
(45) empty -> .
(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma
(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 15
(31) SC -> WHILE parentA . E parentC corcheteA C corcheteC
(34) E -> . E or G
(35) E -> . G
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 16
(21) S -> PRINT parentA . E parentC puntcoma
(34) E -> . E or G
(35) E -> . G
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 17
(43) V -> parentA . E parentC
(34) E -> . E or G
(35) E -> . G
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena

(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 18
(35) E -> G .
(36) G -> G . menor U

state 19
(40) V -> id .
(44) V -> id . parentA L parentC

state 20
(42) V -> cadena .

state 21
(37) G -> U .
(38) U -> U . suma V

state 22
(39) U -> V .

state 23
(24) S -> RETURN X . puntcoma

state 24
(29) X -> E .
(34) E -> E . or G

state 25
(30) X -> empty .

state 26
(41) V -> entero .

state 27
(23) S -> id parentA . L parentC puntcoma
(25) L -> . empty
(26) L -> . id L1
(45) empty -> .

state 28
(20) S -> id asig . E puntcoma
(34) E -> . E or G
(35) E -> . G
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 29
(19) S -> id igual . E puntcoma
(34) E -> . E or G
(35) E -> . G
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 30
(7) T -> STRING .

state 31
(6) T -> INT .

state 32
(9) F -> FUNCTION T1 . id parentA A parentC corcheteA C corcheteC

state 33
(8) T -> BOOLEAN .

state 34
(10) T1 -> T .

state 35
(11) T1 -> empty .

state 36
(1) P -> D P .

state 37
(2) P -> F P .

state 38
(5) D -> VAR T . id puntcoma

state 39
(22) S -> INPUT parentA . E parentC puntcoma
(34) E -> . E or G
(35) E -> . G
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 40
(32) SC -> IF parentA . E parentC S
(34) E -> . E or G
(35) E -> . G
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 41
(3) P -> SC P .

state 42
(31) SC -> WHILE parentA E . parentC corcheteA C corcheteC
(34) E -> E . or G

state 43
(21) S -> PRINT parentA E . parentC puntcoma
(34) E -> E . or G

state 44
(43) V -> parentA E . parentC
(34) E -> E . or G

state 45
(36) G -> G menor . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 46
(44) V -> id parentA . L parentC
(25) L -> . empty
(26) L -> . id L1
(45) empty -> .

state 47
(38) U -> U suma . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 48
(24) S -> RETURN X puntcoma .

state 49
(34) E -> E or . G
(36) G -> . G menor U
(37) G -> . U
(38) U -> . U suma V
(39) U -> . V
(40) V -> . id
(41) V -> . entero
(42) V -> . cadena
(43) V -> . parentA E parentC
(44) V -> . id parentA L parentC

state 50
(23) S -> id parentA L . parentC puntcoma

state 51
(26) L -> id . L1

(27) L1 -> . empty
(28) L1 -> . coma id L1
(45) empty -> .

state 52
(25) L -> empty .

state 53
(20) S -> id asig E . puntcoma
(34) E -> E . or G

state 54
(19) S -> id igual E . puntcoma
(34) E -> E . or G

state 55
(9) F -> FUNCTION T1 id . parentA A parentC corcheteA C corcheteC

state 56
(5) D -> VAR T id . puntcoma

state 57
(22) S -> INPUT parentA E . parentC puntcoma
(34) E -> E . or G

state 58
(32) SC -> IF parentA E . parentC S
(34) E -> E . or G

state 59
(31) SC -> WHILE parentA E parentC . corcheteA C corcheteC

state 60
(21) S -> PRINT parentA E parentC . puntcoma

state 61
(43) V -> parentA E parentC .

state 62
(36) G -> G menor U .
(38) U -> U . suma V

state 63
(44) V -> id parentA L . parentC

state 64
(38) U -> U suma V .

state 65
(34) E -> E or G .
(36) G -> G . menor U

state 66
(23) S -> id parentA L parentC . puntcoma

state 67
(28) L1 -> coma . id L1

state 68
(27) L1 -> empty .

state 69
(26) L -> id L1 .

state 70
(20) S -> id asig E puntcoma .

state 71
(19) S -> id igual E puntcoma .

state 72
(9) F -> FUNCTION T1 id parentA . A parentC corcheteA C corcheteC
(12) A -> . empty
(13) A -> . T id A1
(45) empty -> .
(6) T -> . INT
(7) T -> . STRING
(8) T -> . BOOLEAN

state 73
(5) D -> VAR T id puntcoma .

state 74
(22) S -> INPUT parentA E parentC . puntcoma

state 75

(32) SC -> IF parentA E parentC . S
(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma
(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 76
(31) SC -> WHILE parentA E parentC corcheteA . C corcheteC
(16) C -> . D C
(17) C -> . SC C
(18) C -> . empty
(5) D -> . VAR T id puntcoma
(31) SC -> . WHILE parentA E parentC corcheteA C corcheteC
(32) SC -> . IF parentA E parentC S
(33) SC -> . S
(45) empty -> .
(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma
(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 77
(21) S -> PRINT parentA E parentC puntcoma .

state 78
(44) V -> id parentA L parentC .

state 79
(23) S -> id parentA L parentC puntcoma .

state 80
(28) L1 -> coma id . L1
(27) L1 -> . empty
(28) L1 -> . coma id L1
(45) empty -> .

state 81
(9) F -> FUNCTION T1 id parentA A . parentC corcheteA C corcheteC

state 82
(13) A -> T . id A1

state 83
(12) A -> empty .

state 84
(22) S -> INPUT parentA E parentC puntcoma .

state 85
(32) SC -> IF parentA E parentC S .

state 86
(18) C -> empty .

state 87
(31) SC -> WHILE parentA E parentC corcheteA C . corcheteC

state 88
(16) C -> D . C
(16) C -> . D C
(17) C -> . SC C
(18) C -> . empty
(5) D -> . VAR T id puntcoma
(31) SC -> . WHILE parentA E parentC corcheteA C corcheteC
(32) SC -> . IF parentA E parentC S
(33) SC -> . S
(45) empty -> .
(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma

(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 89
(17) C -> SC . C
(16) C -> . D C
(17) C -> . SC C
(18) C -> . empty
(5) D -> . VAR T id puntcoma
(31) SC -> . WHILE parentA E parentC corcheteA C corcheteC
(32) SC -> . IF parentA E parentC S
(33) SC -> . S
(45) empty -> .
(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma
(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 90
(28) L1 -> coma id L1 .

state 91
(9) F -> FUNCTION T1 id parentA A parentC . corcheteA C corcheteC

state 92
(13) A -> T id . A1
(14) A1 -> . empty
(15) A1 -> . coma T id A1
(45) empty -> .

state 93
state 99
(14) A1 -> empty .

state 100
(9) F -> FUNCTION T1 id parentA A parentC corcheteA C . corcheteC
(31) SC -> WHILE parentA E parentC corcheteA C corcheteC .

state 94
(16) C -> D C .

state 95
(17) C -> SC C .

state 96
(9) F -> FUNCTION T1 id parentA A parentC corcheteA . C corcheteC
(16) C -> . D C
(17) C -> . SC C
(18) C -> . empty
(5) D -> . VAR T id puntcoma
(31) SC -> . WHILE parentA E parentC corcheteA C corcheteC
(32) SC -> . IF parentA E parentC S
(33) SC -> . S
(45) empty -> .
(19) S -> . id igual E puntcoma
(20) S -> . id asig E puntcoma
(21) S -> . PRINT parentA E parentC puntcoma
(22) S -> . INPUT parentA E parentC puntcoma
(23) S -> . id parentA L parentC puntcoma
(24) S -> . RETURN X puntcoma

state 97
(13) A -> T id A1 .

state 98
(15) A1 -> coma . T id A1
(6) T -> . INT
(7) T -> . STRING
(8) T -> . BOOLEAN

state 101
(15) A1 -> coma T . id A1

state 102

(9) F -> FUNCTION T1 id parentA A parentC corcheteA C corcheteC .

(45) empty ->

state 103

(15) A1 -> coma T id . A1
(14) A1 -> . empty
(15) A1 -> . coma T id A1

state 104

(15) A1 -> coma T id A1 .

Demostración de que la gramática es válida

No hay conflictos ni reducción/reducción ni reducción/desplazamiento, por lo tanto, la gramática es válida.

Analizador semántico

Traducción dirigida por la sintaxis

Reglas:

1. P -> D P
2. P -> F P
3. P -> SC P
4. P -> lambda
5. D -> var T id ;
   if buscarTSActual(id) == true then tipo_error
   else insertarTSActual(id)
6. T -> int
   T.tipo := int
7. T -> string
   T.tipo := string
8. T -> boolean
   T.tipo := boolean
9. F -> function T1 id (A) { C }
   AñadirTS(id.pos), AñadirArgumTS(A.tipo), AñadirTipoReto(T1.tipo)
   If T1.tipo != None then F.tiporet:= T1.tipo
   Crear TSLocal, AñadirTSLocal(A.tipo)
10. T1 -> T
    T1.tipo := T.tipo
11. T1 -> lambda
12. A -> lambda
13. A -> T id A1
    AñadirArgumTS(funcion,id.pos, T.tipo)
    numParam++
    A.tipo:= T.tipo
14. A1 -> lambda
15. A1 -> , T id A1

AñadirArgumTS(funcion,id.pos, T.tipo)
numParam++
A1.tipo:= T.tipo

16. C -> D C
    C.tipo := D.tipo
17. C -> SC C
    C.tipo := SC.tipo
18. C -> lambda
19. S -> id = E ;
    if buscarTipoTS(id.pos) = E.tipo then S.tipo:= E.tipo
    else tipo_error("tipos distintos")
20. S -> id &= E ;
    if buscarTipoTS(id.pos) = E.tipo then S.tipo:= E.tipo
    else tipo_error("tipos distintos")
21. S -> print (E) ;
    If E.tipo==boolean then tipo_error(“No se pueden imprimir booleanos”)
    S.tipo != tipo_error then S.tipo := ok
    else S.tipo:=tipo_error

22. S -> input (id) ;
    If buscarTipoTS(id.pos) ==Boolean then tipo_error(“No se pueden almacenar boolanos”)
    else S.tipo:= ok

23. S -> id (L) ;
    if buscarTipoTS(id.pos)==function and L.tipo==buscarTiposArgTS(function, id.pos) then
    V.tipo:=buscarTipoRetorno(id.pos)
    else tipo_error (“La llamada a la función es errónea”)

24. S -> return X ;
    S.tipo != tipo_error then S.tipo := ok
    else S.tipo:=tipo_error
    S.tiporet := X.tipo
25. L -> lambda
26. L -> E L1
    L1.tipo := E.tipo
27. L1 -> lambda
28. L1 -> , E L1
    L1.tipo := E.tipo
29. X -> E
    X.tipo := E.tipo
30. X -> lambda
31. SC -> while (E) { C }
    if E.tipo== bolean then SC.tipo = ok
    else tipo_error (no es una condicion)
32. SC -> if (E) S

if E.tipo== boolean then SC.tipo = ok
else tipo_error (no es una condicion)

33. SC -> S
    SC.tipo := S.tipo
34. E -> E || G
    if E.tipo=G.tipo=boolean then E.tipo:=boolean
    else tipo_error("Solo se pueden comparar tipos booleanos")
35. E -> G
    E.tipo := G.tipo
36. G -> G < U
    if G.tipo=U.tipo=entero then G.tipo:=entero
    else tipo_error("Solo se pueden comparaciones de enteros")
37. G -> U
    G.tipo := U.tipo
38. U -> U + V
    if U.tipo=V.tipo=entero then U.tipo:=entero
    else tipo_error("Solo se pueden sumar enteros")
39. U -> V
    U.tipo := V.tipo
40. V -> id
    AñadirTipoTS(id.pos, V.tipo)
41. V -> entero
    V.tipo := entero
42. V -> cadena
    V.tipo := cadena
43. V -> (E)
    V.tipo := E.tipo
44. V -> id (L)
    if buscarTipoTS(id.pos)==function and
    L.tipo==buscarTiposArgTS(function, id.pos) then
    V.tipo:=buscarTipoRetorno(id.pos)
    else tipo_error (“La llamada a la función es errónea”)

Anexo

Pruebas correctas:

Para los casos de prueba correctos, detallaremos los ficheros resultantes y el árbol VAST de la primera prueba. El resto de ficheros sobre el resto de pruebas se encuentran en el disco.

Prueba 1

```
var boolean a;
var string c;
var int b;
cadena = c;
booleano &=a;

function boolean existe(boolean e, string texto, int b){
   if (e||booleano) return a;
   print("Existo");
   input(cadena);
   existe(a,cadena,b);
}
booleano2 = existe(booleano,c,b);
```

Parse: A 8 5 7 5 6 5 40 39 37 35 19 33 40 39 37 35 20 33 8 10 8 7 6 14 15 15 13 40 39 37 35 40 39 37 34 40 39 37 35 29 24 32 42 39 37 35 21 33 22 33 40 39 37 35 40 39 37 35 40 39 37 35 27 28 28 26 23 33 18 17 17 17 17 9 40 39 37 35 40 39 37 35 40 39 37 35 27 28 28 26 44 39 37 35 19 33 4 3 2 3 3 1 1 1

Tokens:

<VAR, >
<BOOLEAN, >
<id,1>
<puntcoma, >
<VAR, >
<STRING, >
<id,2>
<puntcoma, >
<VAR, >
<INT, >
<id,4>
<puntcoma, >
<id,8>
<igual, >
<id,2>
<puntcoma, >
<id,16>
<asig, >
<id,1>
<puntcoma, >
<FUNCTION, >
<BOOLEAN, >
<id,32>
<parentA, >
<BOOLEAN, >
<id,64>
<coma, >
<STRING, >
<id,128>
<coma, >
<INT, >
<id,4>
<parentC, >
<corcheteA, >
<IF, >
<parentA, >
<id,64>
<or, >

<id,16>
<parentC, >
<RETURN, >
<id,1>
<puntcoma, >
<PRINT, >
<parentA, >
<cadena,"Existo">
<parentC, >
<puntcoma, >
<INPUT, >
<parentA, >
<id,8>
<parentC, >
<puntcoma, >
<id,32>
<parentA, >
<id,1>
<coma, >
<id,8>
<coma, >
<id,4>
<parentC, >
<puntcoma, >
<corcheteC, >
<id,256>
<igual, >
<id,32>
<parentA, >
<id,16>
<coma, >
<id,2>
<coma, >
<id,4>
<parentC, >
<puntcoma, >

Tabla de símbolos:

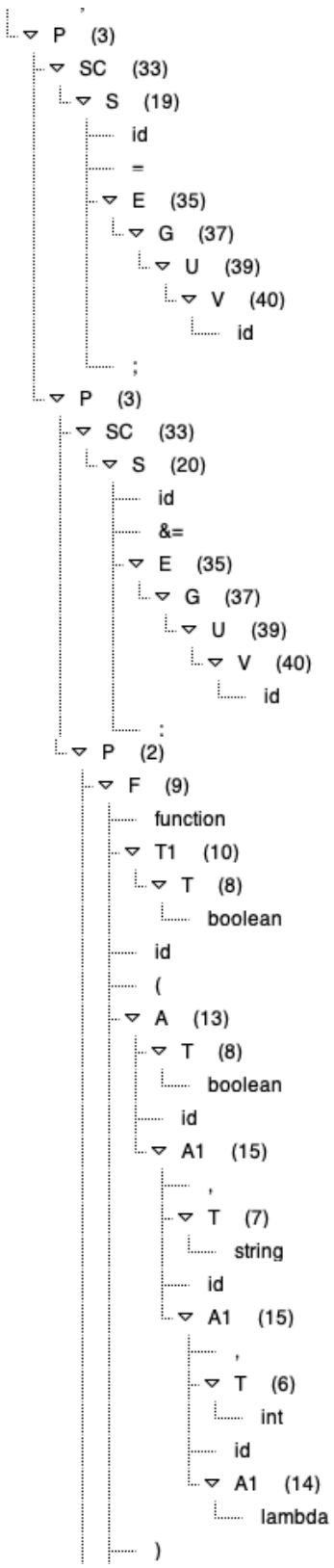
TABLA PRINCIPAL #1:
* LEXEMA: 'a'
    + Tipo: 'BOOLEAN'
    + Despl: 0
* LEXEMA: 'c'
    + Tipo: 'STRING'
    + Despl: 1
* LEXEMA: 'b'
    + Tipo: 'INT'
    + Despl: 65
* LEXEMA: 'cadena'
    + Tipo: 'STRING'
    + Despl: 66
* LEXEMA: 'booleano'
    + Tipo: 'BOOLEAN'
    + Despl: 130
* LEXEMA : 'existe'
    + Tipo: 'function'
    + NumParam: 3
    + TipoParam1: 'BOOLEAN'
    + TipoParam2: 'STRING'
    + TipoParam3: 'INT'
    + TipoRetorno: 'BOOLEAN'
    + EtiqFuncion: 'ETexiste'
* LEXEMA: 'booleano2'

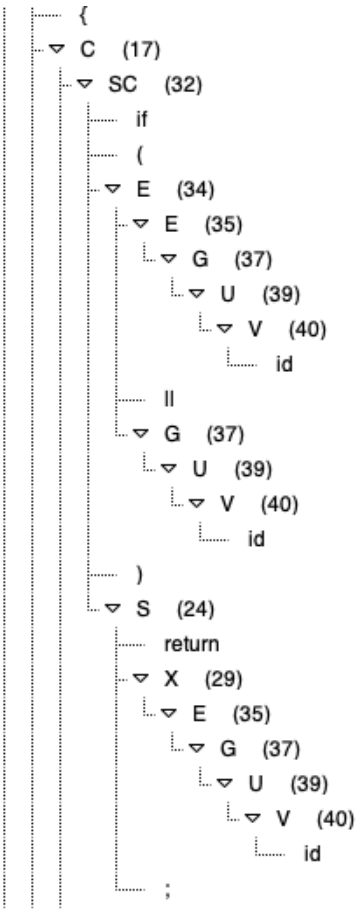
+ Tipo: 'BOOLEAN'
+ Despl: 131

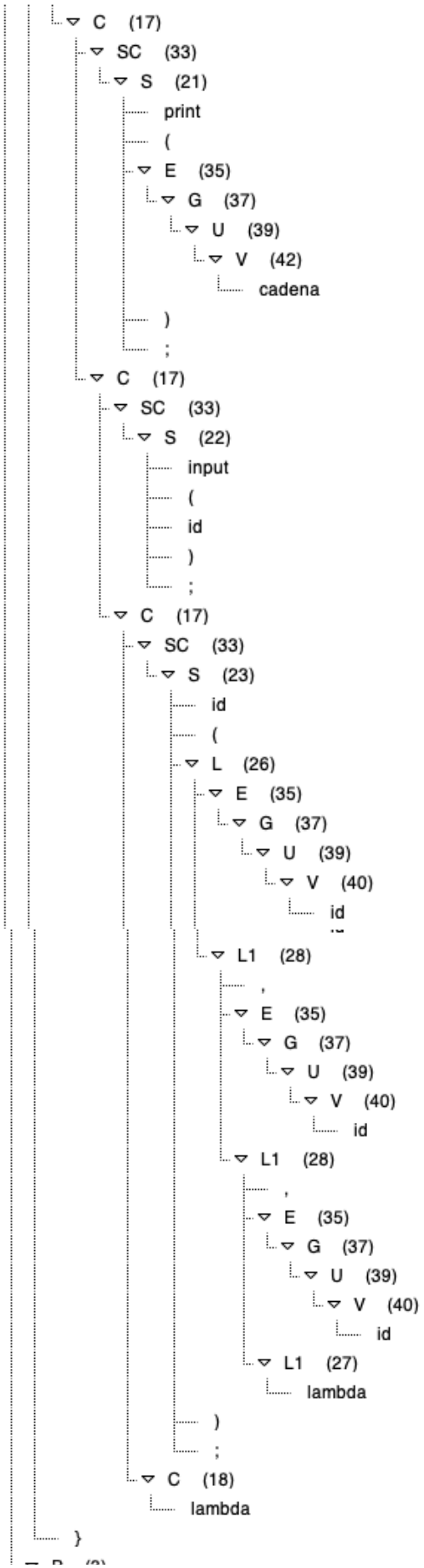
TABLA de la FUNCION existe #2:

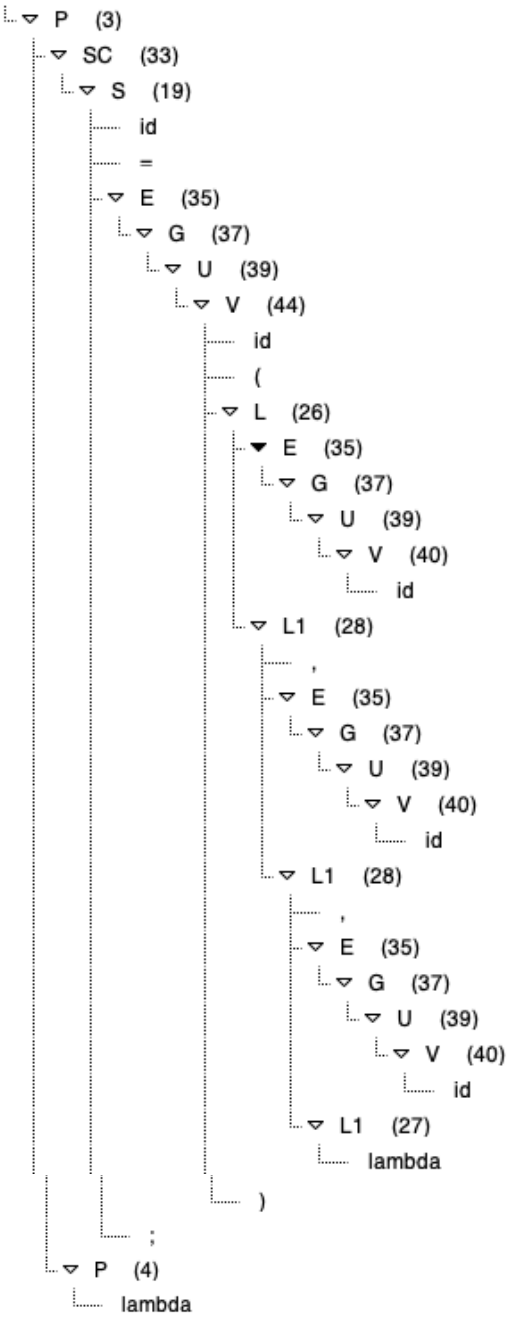
* LEXEMA: 'e'
  + Tipo: 'BOOLEAN'
  + Despl: 0
* LEXEMA: 'texto'
  + Tipo: 'STRING'
  + Despl: 1
* LEXEMA: 'b'
  + Tipo: 'INT'
  + Despl: 65

Árbol VAST:


P (3)
SC (33)
S (19)
id
=
E (35)
G (37)
U (39)
V (40)
id
;
P (3)
SC (33)
S (20)
id
&=
E (35)
G (37)
U (39)
V (40)
id
;
P (2)
F (9)
function
T1 (10)
T (8)
boolean
id
(
A (13)
T (8)
boolean
id
A1 (15)
,
T (7)
string
id
A1 (15)
,
T (6)
int
id
A1 (14)
lambda
)


{
C (17)
SC (32)
if
(
E (34)
E (35)
G (37)
U (39)
V (40)
id
||
G (37)
U (39)
V (40)
id
)
S (24)
return
X (29)
E (35)
G (37)
U (39)
V (40)
id
;


C (17)
SC (33)
S (21)
print
(
E (35)
G (37)
U (39)
V (42)
cadena
)
;
C (17)
SC (33)
S (22)
input
(
id
)
;
C (17)
SC (33)
S (23)
id
(
L (26)
E (35)
G (37)
U (39)
V (40)
id
L1 (28)
,
E (35)
G (37)
U (39)
V (40)
id
L1 (28)
,
E (35)
G (37)
U (39)
V (40)
id
L1 (27)
lambda
)
;
C (18)
lambda
}



Prueba 2

```
var int a;
var int b;
var int c;
print ("Introduce el primer operando");
input (a);
function int suma (int num1, int num2)
{
	while (a<0) {
		var int res;
		res = num1+num2;
		return res;
```

```
                suma(a,b);
        }
}
c = suma (a, b);
print (c);
```

Prueba 3

```
var int a;
var int b;
a = 3;
b = a;
if (a < b) b = 1;
if (b < a) b = 8;
a = a + b;
print (a);
print (b);
```

Prueba 4

```
var string texto;
function pideTexto ()
{
        print ("Introduce un texto");
        input (texto);
}
function imprime (string msg)
{
        print (msg);
}
pideTexto();
var string textoAux;
textoAux = texto;
imprime (textoAux);
```

Prueba 5

```
var int a;
var int b;
var int c;
print ("Introduce el primer operando");
input (a);
print ("Introduce el segundo operando");
input (b);
function int suma (int num1, int num2)
{
        var int res;
        res = num1+num2;
        return res;
```

```
}
c = suma (a, b);
print (c);
```

Pruebas incorrectas:

Prueba 1

```
var boolean a;
var string c;
var int b;
cadena &= c;
booleano =a;

function boolean existe(boolean e, string texto, int b){
   if (e > booleano) return a;
   print("Existo");
   input(cadena);
   existe(a,cadena,b);
}
booleano2 = existe(booleano,c,b);
```

Errores:
Error lexico en la linea: 8. Token ilegal: >
Error semantico en la linea: 4. No se pueden hacer asignaciones logicas con variables que no sean booleanas
Error sintactico en la linea: 7. Se ha leído el token id cuando no era el token esperado

Prueba 2

```
var int a;
var int b;
var int c;
print ("Introduce el primer operando");
input (a);
function int suma (int num1, int num2, int num3)
{
        while (a<0) {
                var int res;
                res = num1 || num2;
                return res;
        }
}
c = suma (a, b);
print (c);
```

Errores:
Error semantico en la linea: 9. Solo se admiten comparaciones de disyuncion de variables de tipo "boolean"
Error semantico en la linea: 10. No se puede asignar a una variable de tipo "INT" una expresion de tipo "None"

Error semantico en la linea: 13. El numero de argumentos pasados a la funcion "suma" es invalido, deberia de ser 3
Error semantico en la linea: 13. El numero de argumentos pasados a la funcion "suma" es invalido, deberia de ser 3

Prueba 3

```
var int a;
var int b;
a = 3;
b = a
if (a > b) b = 1;
if (b < a) b = 8;
a = a + b;
print ();
print (b);
```

Errores:
Error lexico en la linea: 5. Token ilegal: >
Error sintactico en la linea: 4. Se ha leído el token IF cuando no era el token esperado

Prueba 4

```
var string texto;
function imprime (string msg){
	print (msg);
}
function pideTexto () {
	var int res;
	res = num1+num2;
	res = num1 || num2;
}
pideTexto(texto);
var string textoAux;
textoAux = texto;
imprime (i);
```

Errores:
Error semantico en la linea: 7. Solo se admiten comparaciones de disyuncion de variables de tipo "boolean"
Error semantico en la linea: 8. No se puede asignar a una variable de tipo "INT" una expresion de tipo "None"
Error semantico en la linea: 10. La funcion "pideTexto" ha recibido argumentos cuando ha sido declarada sin ellos
Error semantico en la linea: 10. El numero de argumentos pasados a la funcion "pideTexto" es invalido, deberia de ser 0
Error semantico en la linea: 13. El argumento esperado de la llamada funcion "imprime" deberia de ser del tipo "STRING" ,pero ha sido del tipo "INT"

Prueba 5

```
var boolean a;
var int c;
print ("Introduce el primer operando");
input (a);
print ("Introduce el segundo operando");
input (b);
function suma (int num1, int num2)
{
        var int res;
        res = num1+num2;
        return res;
}
c = suma (a, b);
print (a);
```

Errores:
Error semantico en la linea: 4. Solo se pueden almacenar expresiones de tipo cadena o entero escritas previamente por teclado
Error semantico en la linea: 11. El tipo de retorno de la funcion el cual es "None" es distinto del tipo de la variable devuleta en el "return", que es "INT"
Error semantico en la linea: 12. El argumento esperado de la llamada funcion "suma" deberia de ser del tipo "INT" ,pero ha sido del tipo "BOOLEAN"
Error semantico en la linea: 13. No se puede asignar a una variable de tipo "INT" una expresion de tipo "None"
Error semantico en la linea: 14. Solo se pueden imprimir por pantalla expresiones de tipo cadena o entero