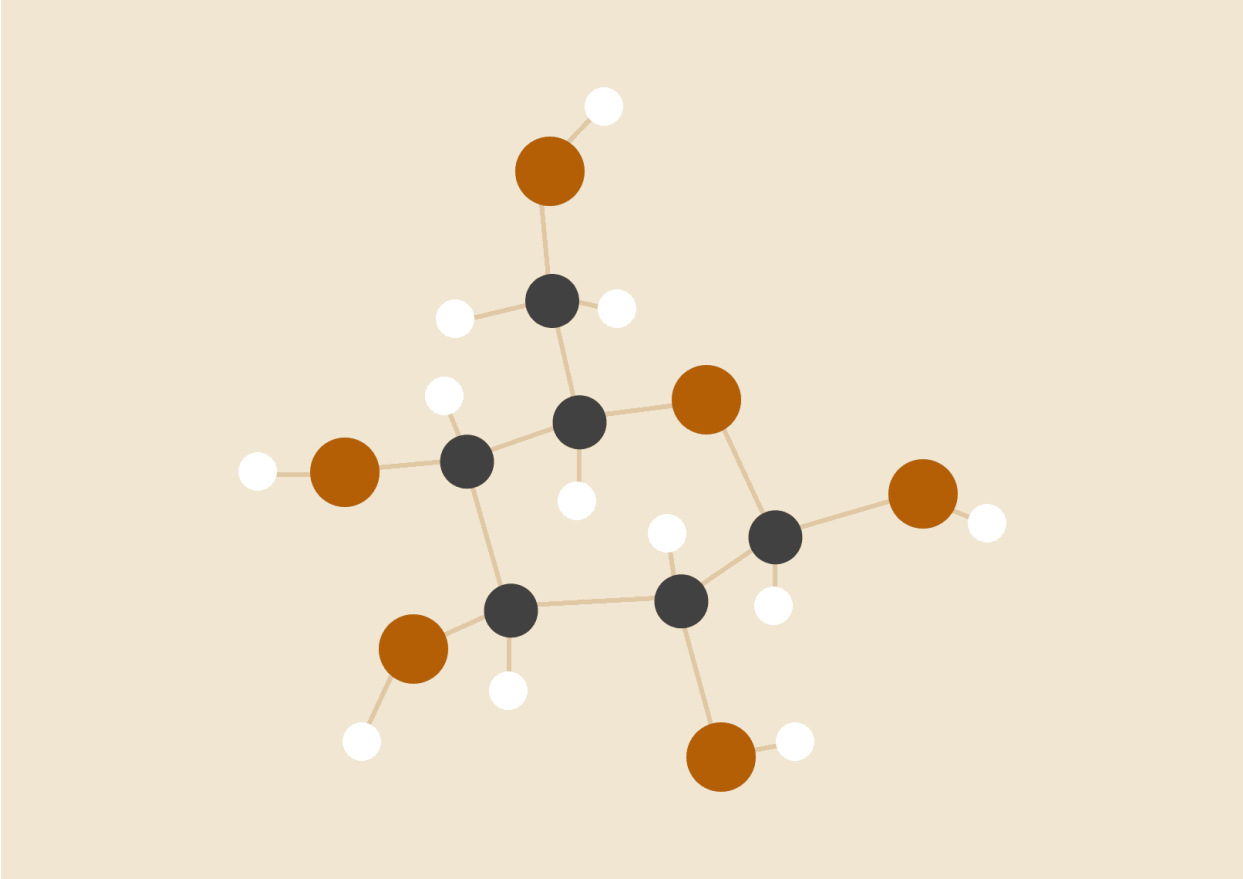
PRÁCTICA PROCESADORES DE LENGUAJES

Grupo 6



Ana García López de Asiaín (210360)
Alvaro García-Caro Bartolomé (210363)
Alfonso Marín (210253)

13/01/2025

Grupo 5S1M

ÍNDICE

1. DISEÑO DEL ANALIZADOR LÉXICO

Definición de Tokens

1. <ENTERO, valor>
2. <CADENA, lexema>
3. <FUNCTION, ->
4. <BOOLEAN, ->
5. <PARENTABRE, ->
6. <PARENTCIERRA, ->
7. <SUMA, ->
8. <IGUIGU, ->
9. <AND, ->
10. <IGUAL, ->
11. <ASIGMULT, ->
12. <ID, posid>
13. <VAR, ->
14. <INT, ->
15. <STRING, ->
16. <OUTPUT, ->
17. <INPUT, ->
18. <PUNTYCOM, ->
19. <COMA, ->
20. <RETURN, ->

21. <IF, ->

22. <SWITCH, ->

23. <LLAVEABRE, ->

24. <LLAVECIERRA, ->

25. <EOF, ->

26. <BREAK,->

27. <CASE,->

28. <VOID,->

29. <DOSPUNTOS,->

Definición de la Gramática

S -> del S | d A | l B | / C | * E | ‘ F | = G | &H| + | ; | : | , | { | } | (|) | eof

A -> d A | λ

B -> d B | l B | _ B| λ

C -> *C’

C’ -> c’ C’ | *C”

C” -> c”C’ | *C” | / S

E -> =

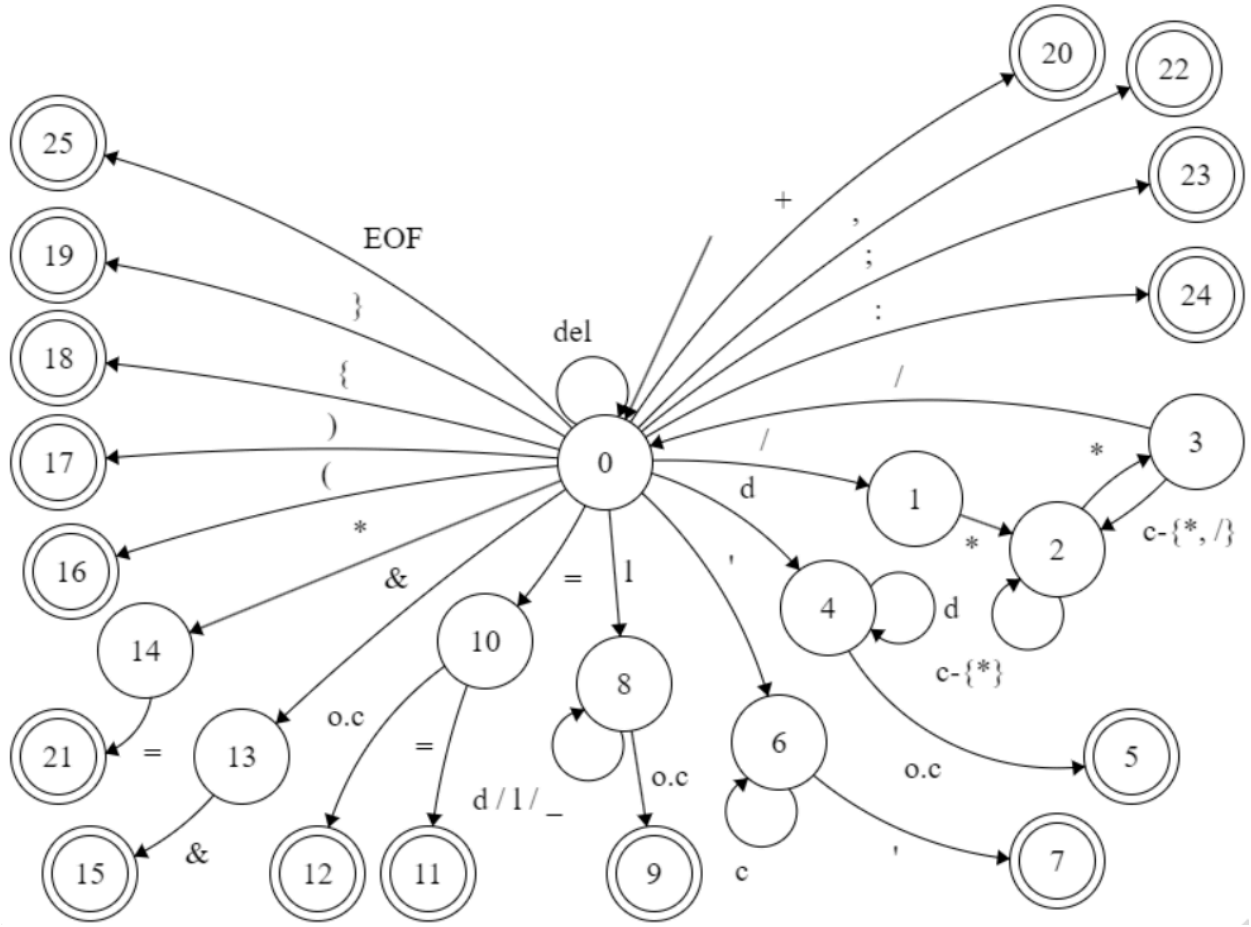
F -> c F | ‘

G -> = | λ

H -> &

donde del : {ƀ, TAB, eof}, l : caracteres que pertenecen al conjunto de las letras mayúsculas y minúsculas, es decir, {a-z, A-Z}, c : carácter, c’: cualquier carácter menos * c”: cualquier carácter menos * y /

Definición del Autómata Finito Determinista



Definición de las Acciones Semánticas

0:0 -> Leer;

0:1 -> Leer; valor = valor(d);

1:1 -> valor = valor * 10 + d; Leer

1:11 -> if (valor>32767) then GestorErrores(“Entero no puede ser superior a 32767”)
else GenToken(ENTERO, valor);

0:8 -> lexema = l; Leer;

8:8 -> lexema = lexema + l/d/_ ; Leer;

8:9 -> if(esPalabraReservada(lexema))
GenToken(lexema.toUpperCase(), -)

```
        }else{
                GenToken(ID, posid)

        } Leer;
```

0:3 -> Leer;

3:4 -> Leer;

4:4 -> Leer;

4:5 -> Leer;

5:4 -> Leer;

5:0 -> Leer;

0:14 -> Leer;

14:21 -> GenToken(ASIGMULT, -);

0:6 -> Leer; lex = “” cont = 0

6:6 -> lex = lex + c; Leer; cont++;

6:7 -> if(cont > 64) then GestorErrores(“Cadena no puede ser superior a 64”)
else GenToken(CADENA, lex);

0:10 -> Leer;

10:11 -> Leer; GenToken(IGUIGU, -)

10:12 -> GenToken(IGUAL, -);

0:13 -> Leer;

13:15 -> Leer; GenToken(AND, -)

0:20 -> Leer; GenToken(SUMA, -)

0:23 -> Leer; GenToken(PUNTYCOM, -)

0:22 -> Leer; GenToken(COMA, -)

0:18 -> Leer; GenToken(CORCHABRE, -)

0:19 -> Leer; GenToken(CORCHCIERRA, -)

0:16 -> Leer; GenToken(PARENTABRE, -)

0:17 -> Leer; GenToken(PARENTCIERRA, -)

0:25 -> Leer; GenToken(DOSPUNTOS, -)

0:25 -> GenToken(EOF, -);

2. DISEÑO DEL ANALIZADOR SINTÁCTICO

Definición de la Gramática

Terminales = { if var id input output eof (;) , { : } entero int boolean function

string void return cadena switch case break = *= == + && }

NoTerminales = { E 1 R 2 U 3 V O S D L Q X B Y M N T F H A K C P }

Axioma = P

Producciones = {

E -> R 1

1 -> && R 1

1 -> lambda

R -> U 2

2 -> == U 2

2 -> lambda

U -> V 3

3 -> + V 3

3 -> lambda

V -> id O

V -> (E)

V -> entero

V -> cadena

O -> lambda

O -> (L)

S -> id D ;

S -> output E ;

S -> input id ;

S -> return X ;

D -> = E

D -> *= E

D -> (L)

L -> E Q

L -> lambda

Q -> , E Q

Q -> lambda

X -> E

X -> lambda

B -> if (E) S

B -> var T id ;

B -> S

B -> switch (E) { Y }

Y -> case entero : C M

M -> break ; N

M -> N

N -> Y

N -> lambda

T -> int

T -> boolean

T -> string

F -> function H id (A) { C }

H -> T

H -> void

A -> T id K

A -> void

K -> , T id K

K -> lambda

C -> B C

C -> lambda

P -> B P

P -> F P

P -> eof

P -> lambda

}

Comprobación de que es LL1

Analizando símbolo 1

Analizando producción 1 -> && R 1

FIRST de 1 -> && R 1 = { && }

Analizando producción 1 -> lambda

FIRST de 1 -> lambda = { lambda }

FIRST de 1 = { && lambda }

Calculando FOLLOW de 1

Calculando FOLLOW de E

Calculando FOLLOW de D

FOLLOW de D = { ; }

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

Calculando FOLLOW de L

FOLLOW de L = {) }

FOLLOW de Q = {) }

Calculando FOLLOW de X

FOLLOW de X = { ; }

FOLLOW de E = {) , ; }

FOLLOW de 1 = {) , ; }

Analizando símbolo 2

Analizando producción 2 -> == U 2

FIRST de 2 -> == U 2 = { == }

Analizando producción 2 -> lambda

FIRST de 2 -> lambda = { lambda }

FIRST de 2 = { == lambda }

Calculando FOLLOW de 2

Calculando FOLLOW de R

FOLLOW de R = { &&) , ; }

FOLLOW de 2 = { &&) , ; }

Analizando símbolo 3

Analizando producción 3 -> + V 3

FIRST de 3 -> + V 3 = { + }

Analizando producción 3 -> lambda

FIRST de 3 -> lambda = { lambda }

FIRST de 3 = { + lambda }

Calculando FOLLOW de 3

Calculando FOLLOW de U

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

FOLLOW de 3 = { &&) , ; == }

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 -> var T id ;

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

Analizando producción B -> S

Analizando símbolo S

Analizando producción S -> id D ;

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

Analizando producción S -> output E ;

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

Analizando producción S -> input id ;

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

Analizando producción S -> return X ;

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

FIRST de S = { id input output return }

FIRST de B -> S = { id input output return }

Analizando producción B -> switch (E) { Y }

FIRST de B -> switch (E) { Y } = { switch }

FIRST de B = { id if input output return switch var }

Analizando símbolo C

Analizando producción C -> B C

FIRST de C -> B C = { id if input output return switch var }

Analizando producción C -> lambda

FIRST de C -> lambda = { lambda }

FIRST de C = { id if input output return switch var lambda }

Calculando FOLLOW de C

Analizando símbolo M

Analizando producción M -> break ; N

FIRST de M -> break ; N = { break }

Analizando producción M -> N

Analizando símbolo N

Analizando producción N -> Y

Analizando símbolo Y

Analizando producción Y -> case entero : C M

FIRST de Y -> case entero : C M = { case }

FIRST de Y = { case }

FIRST de N -> Y = { case }

Analizando producción N -> lambda

FIRST de N -> lambda = { lambda }

FIRST de N = { case lambda }

Calculando FOLLOW de N

Calculando FOLLOW de M

Calculando FOLLOW de Y

FOLLOW de Y = { } }

FOLLOW de M = { } }

FOLLOW de N = { } }

FIRST de M -> N = { case lambda }

FIRST de M = { break case lambda }

FOLLOW de C = { break case } }

Analizando símbolo D

Analizando producción D -> = E

FIRST de D -> = E = { = }

Analizando producción D -> *= E

FIRST de D -> *= E = { *= }

Analizando producción D -> (L)

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

FIRST de D = { (*= = }

Analizando símbolo E

Analizando producción E -> R 1

Analizando símbolo R

Analizando producción R -> U 2

Analizando símbolo U

Analizando producción U -> V 3

Analizando símbolo V

Analizando producción V -> id O

FIRST de V -> id O = { 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 3 = { (cadena entero id }

FIRST de U = { (cadena entero id }

FIRST de R -> U 2 = { (cadena entero id }

FIRST de R = { (cadena entero id }

FIRST de E -> R 1 = { (cadena entero id }

FIRST de E = { (cadena entero id }

Analizando símbolo F

Analizando producción F -> function H id (A) { C }

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

FIRST de F = { function }

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 }

Analizando símbolo O

Analizando producción O -> lambda

FIRST de O -> lambda = { lambda }

Analizando producción O -> (L)

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

FIRST de O = { (lambda }

Calculando FOLLOW de O

Calculando FOLLOW de V

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

FOLLOW de O = { &&) + , ; == }

Analizando símbolo P

Analizando producción P -> B P

FIRST de P -> B P = { id if input output return switch var }

Analizando producción P -> F P

FIRST de P -> F P = { function }

Analizando producción P -> eof

FIRST de P -> eof = { eof }

Analizando producción P -> lambda

FIRST de P -> lambda = { lambda }

FIRST de P = { eof function id if input output return switch var lambda }

Calculando FOLLOW de P

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

Analizando símbolo X

Analizando producción X -> E

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

Analizando producción X -> lambda

FIRST de X -> lambda = { lambda }

FIRST de X = { (cadena entero id lambda }

Análisis concluido satisfactoriamente

Utilizando la herramienta que hemos utilizado para comprobar si la gramática es LL1, obtenemos también la tabla sintáctica, que se muestra en la siguiente hoja en la figura 1.

Tabla sintáctica

	&&	(	)	*=	+	,	:	;	=	==	boolean	break	cadena	case	entero	eof	function	id	if	input	int	output	return	string	switch	var	void	{	}	$ (final de cadena)
1	1 → && R 1	--	1 → lambda	--	--	1 → lambda	--	1 → lambda	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
2	2 → lambda	--	2 → lambda	--	--	2 → lambda	--	2 → lambda	--	2 → == U 2	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
3	3 → lambda	--	3 → lambda	--	3 → + V 3	3 → lambda	--	3 → lambda	--	3 → lambda	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
A	--	--	--	--	--	--	--	--	--	--	A → T id K	--	--	--	--	--	--	--	--	--	A → T id K	--	--	A → T id K	--	--	A → void	--	--	--
B	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	B → S	B → if (E) S	B → S	--	B → S	B → S	--	B → switch (E) { Y }	B → var T id ;	--	--	--	--
C	--	--	--	--	--	--	--	--	--	--	--	C → lambda	--	C → lambda	--	--	--	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 → (L)	--	D → *= E	--	--	--	--	D → = E	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
E	--	E → R 1	--	--	--	--	--	--	--	--	--	--	E → R 1	--	E → R 1	--	--	E → R 1	--	--	--	--	--	--	--	--	--	--	--	--
F	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	F → function H id (A) { C }	--	--	--	--	--	--	--	--	--	--	--	--	--
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 → break ; N	--	M → N	--	--	--	--	--	--	--	--	--	--	--	--	--	--	M → N	--
N	--	--	--	--	--	--	--	--	--	--	--	--	--	N → Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	N → lambda	--
O	O → lambda	O → (L)	O → lambda	--	O → lambda	O → lambda	--	O → lambda	--	O → lambda	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
P	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	P → eof	P → F P	P → B 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 2	--	--	--	--	--	--	--	--	--	--	R → U 2	--	R → U 2	--	--	R → U 2	--	--	--	--	--	--	--	--	--	--	--	--
S	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	S → id D ;	--	S → input id ;	--	S → output E ;	S → return X ;	--	--	--	--	--	--	--
T	--	--	--	--	--	--	--	--	--	--	T → boolean	--	--	--	--	--	--	--	--	--	T → int	--	--	T → string	--	--	--	--	--	--
U	--	U → V 3	--	--	--	--	--	--	--	--	--	--	U → V 3	--	U → V 3	--	--	U → V 3	--	--	--	--	--	--	--	--	--	--	--	--
V	--	V → (E)	--	--	--	--	--	--	--	--	--	--	V → cadena	--	V → entero	--	--	V → id O	--	--	--	--	--	--	--	--	--	--	--	--
X	--	X → E	--	--	--	--	--	X → lambda	--	--	--	--	X → E	--	X → E	--	--	X → E	--	--	--	--	--	--	--	--	--	--	--	--
Y	--	--	--	--	--	--	--	--	--	--	--	--	--	Y → case entero : C M	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Figura 1. Tabla sintáctica de la gramática proporcionada.

3. DISEÑO DEL ANALIZADOR SEMÁNTICO

Esquema de traducción

Producciones = {

E -> R 1 **{1}**

1 -> && R 1 **{2}**

1 -> lambda **{3}**

R -> U 2 **{4}**

2 -> == U 2 **{5}**

2 -> lambda **{6}**

U -> V 3 **{7}**

3 -> + V 3 **{8}**

3 -> lambda **{9}**

V -> id O **{10}**

V -> (E) **{11}**

V -> entero **{12}**

V -> cadena **{13}**

O -> lambda **{14}**

O -> (L) **{15}**

S -> id D ; **{16}**

S -> output E ; **{17}**

S -> input id **{18.1}**; **{18.2}**

S -> return X ; **{19}**

D -> = E **{20}**

D -> *= E **{21}**

D -> (L) **{22}**

L -> E **{23}** Q **{23.1}**

L -> lambda **{24}**

Q -> , E **{25}** Q **{25.1}**

Q -> lambda **{26}**

X -> E **{27}**

X -> lambda **{28}**

B -> if (E) S **{29}**

B -> var T id ; **{30}**

B -> S **{31}**

B -> switch (E) **{32.1}** { Y } **{32.2}**

Y -> case entero : C M **{33}**

M -> break ; N **{34}**

M -> N **{35}**

N -> Y **{36}**

N -> lambda **{37}**

T -> int **{38}**

T -> boolean **{39}**

T -> string **{40}**

F -> function H id **{41.1}** (A) **{41.3}** { C } **{41.4}**

H -> T **{42}**

H -> void **{43}**

A -> T id **{44.1}** K **{44.2}**

A -> void

K -> , T id **{46.1}** K **{46.2}**

K -> lambda **{47}**

C -> B C **{48}**

C -> lambda **{49}**

P -> B P **{50}**

P -> F P **{51}**

P -> eof **{52}**

P -> lambda

}

Leyenda EdT

{1}
E.tipo := if (1.tipo = void || R.tipo = 1.tipo) then R.tipo else tipo_error

{2}
1.tipo := if (1.tipo = void || (1.tipo = boolean && R.tipo = boolean)) then R.tipo else tipo_error

{3}
1.tipo := void

{4}
R.tipo := if (2.tipo = void) then U.tipo else if (2.tipo = U.tipo) then boolean else tipo_error

{5}
2.tipo := if (2.tipo = void) then U.tipo else if (2.tipo = U.tipo) then boolean else tipo_error

{6}
2.tipo := void

{7}
U.tipo := if (3.tipo = void || 3.tipo = V.tipo) then V.tipo else tipo_error

{8}
3.tipo := if (3.tipo = void || (3.tipo = entero && V.tipo = entero)) then V.tipo else tipo_error

{9}
3.tipo := void

{10}
V.tipo := if (buscarTSLocal(idPos) || buscarTSPrincipal(idPos)) then id.Tipo else tipo_error

{11}
V.tipo := E.tipo

{12}
V.tipo := entero

{13}
V.tipo := string

{14}
O.tipo := void

{15}
O.tipo := L.tipo

{16}
S.tipo := if (id.tipo = D.tipo) then tipo_ok else tipo_error

{17}
S.tipo := if (E.tipo = boolean || E.tipo = entero) then boolean else tipo_error

{18.1}
id.tipo := buscarTSTipo(id.pos); if (id.tipo ∈ {cadena, entero}) then tipo_ok else tipo_error

{18.2}
{ }

{19}
tipoRetorno := buscarTSTipoRetorno(idFuncionActual); if (X.tipo = tipoRetorno) then tipo_ok else tipo_error

{20}
D.tipo := E.tipo

{21}
D.tipo := E.tipo (o bien nada -> {})

{22}
D.tipo := if (ArrayListN = buscarTSTipoParametros(idPos)) then “function” else tipo_error

{23}
ArrayListN.add(E.tipo)

{23.1}
zonaDeclaracion := false

{24}
ArrayListN.add(void)

{25}
ArrayListN.add(E.tipo)

{25.1}
zonaDeclaracion := false

{26}
Q.tipo := void

{27}
X.tipo := E.tipo; zonaDeclaracion := false

{28}
X.tipo := void

{29}
B.tipo := if (E.tipo = boolean) then E.tipo else tipo_error; zonaDeclaracion := false

{30}
id.tipo := T.tipo; insertarTSTipoYdesp(id.pos, id.tipo, desp); id.desp := desp + T.ancho

{31}
{ }

{32.1}
B.tipo := if (E.tipo = entero) then tipo_ok else tipo_error; zonaDeclaracion := false

{32.2}
{ }

{33-36}
{ }

{37}
N.tipo := void

{38}
T.tipo := int; T.ancho := 2

{39}
T.tipo := boolean; T.ancho := 1

{40}
T.tipo := string; T.ancho := 64

{41.1}
posIdFuncionActual := pila.peek(); insertarTipoPosIdFuncionActual(“function”);
insertarTipoRetorno(H.tipo); zonaDeclaracion := true

{41.3}
zonaDeclaracion := false

{41.4}
{ }

{42}

H.tipo := T.tipo

{43}
H.tipo := void

{44.1}
id.tipo := T.tipo; insertTSTipoYdesp(id.tipo, desp); desp := desp + T.ancho

{44.2}
(No pone nada)

{45}
A.tipo := void

{46.1}
id.tipo := T.tipo; ArrayListTS.add(id.tipo); insertarTSTipoNueva(id.pos, id.tipo, id.desp); id.desp := T.ancho + desp

{46.2}
{ }

{47}
insertNumParam(posIdFuncionActual, ArrayListN.size())

{48-53}

{ }

4. TABLA DE SÍMBOLOS

La tabla de símbolos es una estructura de datos donde se almacenará la información de los identificadores. Posee una entrada para cada uno de los identificadores con una serie de celdas para atributos, que son:

- Lexema corresponde al nombre de la entrada.
- Tipo corresponde al tipo de la entrada (entero, lógico, cadena, o función).
- Desplazamiento corresponde al desplazamiento de la entrada[1].
- NumParam corresponde al número de parámetros de la entrada (OJO, solo cuando tipo == 'funcion').
- TipoParam y ModoParam corresponde al tipo y modo de pasar el parámetro (OJO, solo cuando tipo == 'funcion').
- ModoParam en el lenguaje de la práctica siempre es por valor (1 para valor, 2 para referencia)[2].
- TipoDev corresponde al tipo que se devueve de la entrada (OJO, solo cuando tipo == 'funcion').
- Etiq corresponde a la etiqueta de la entrada (OJO, solo cuando tipo == 'funcion').

Lexema	Tipo	Desplazamiento	Num Param	Tipo Param	ModoParam	Tipo Dev	Etiq

Figura 2. Diseño de la Tabla de Símbolos.

[1] Los tipos enteros y lógicos tienen desplazamiento 1. Los tipos cadenas tienen desplazamiento 64. Los tipos vacíos y las funciones no tienen desplazamiento.
[2] En este lenguaje sólo se pasan por valor. En las TS se mostrará como 1 (1 para valor, 2 para referencia).

5. ANEXO (CASOS DE PRUEBA)

Por cada código se presenta el contenido de los tokens, tablas de símbolos generadas, el parse y su árbol. Si hubiera errores, se muestra el archivo de errores generado por el Gestor de Errores, y el contenido de los otros 4 archivos hasta donde se genera el error.

- **Código 1 (Sin errores)**

CÓDIGO:

```
/* Declaración de variables

 globales de los 3 tipos pos

 ibles */


var int x; x *= 6;

var string cadena;

cadena = 'Pruebaaaaaaaaaaaaaaaaa';


var boolean bol;


output 'Contenido de cadena: \n';

output cadena; output 'Asigna el valor a la variables x:';

input x;


/* Declaración de una

función para imprimir parámetros en función de un valor */
```

```
function void impriParam  (int valor, string cad1,
string cad2, string cad3){

    var int aux;
    aux = valor;
    var boolean z;

    switch (valor){
        case 1: output cad1;
        break;
        case 2: output cad2;
        case 3: output cad3;
        break;
    }

    var int aux2;
    aux2 *= aux;
}

/* Declaración de una función para comprobar si dos números son iguales */

/* function boolean check(int valor1, int valor2){
    return valor1 == valor 2;
} */
```

/* Fin de código */

PARSE:

Descendente 50 30 38 50 31 16 21 1 4 7 12 9 6 3 50 30 40 50 31 16 20 1 4 7 13 9 6 3 50 30 39 50 31 17 1 4 7 13 9 6 3 50 31 17 1 4 7 10 14 9 6 3 50 31 17 1 4 7 13 9 6 3 50 31 18 51 41 43 44 38 46 40 46 40 46 40 47 48 30 38 48 31 16 20 1 4 7 10 14 9 6 3 48 30 39 48 32 1 4 7 10 14 9 6 3 33 48 31 17 1 4 7 10 14 9 6 3 49 34 36 33 48 31 17 1 4 7 10 14 9 6 3 49 35 36 33 48 31 17 1 4 7 10 14 9 6 3 49 34 37 48 30 38 48 31 16 21 1 4 7 10 14 9 6 3 49 52

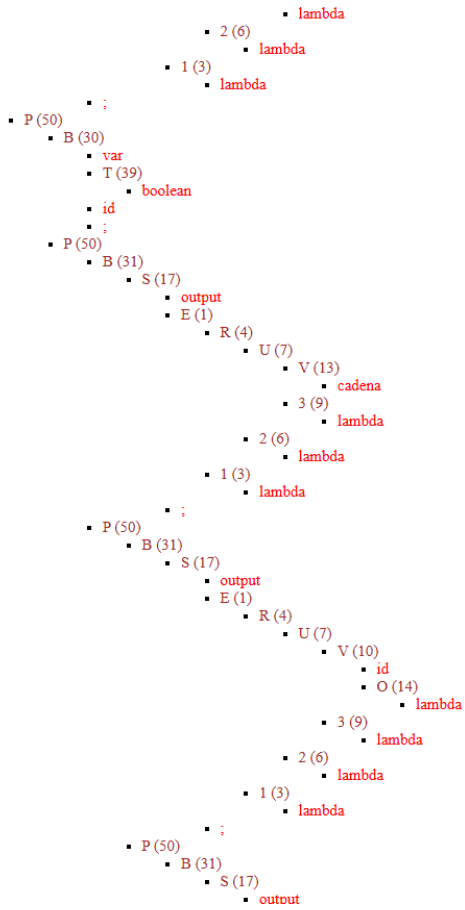
ÁRBOL:

Árbol resultado de:

Gramática: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Utilidades\Documentacion 24-25\Gramática 24-25.txt

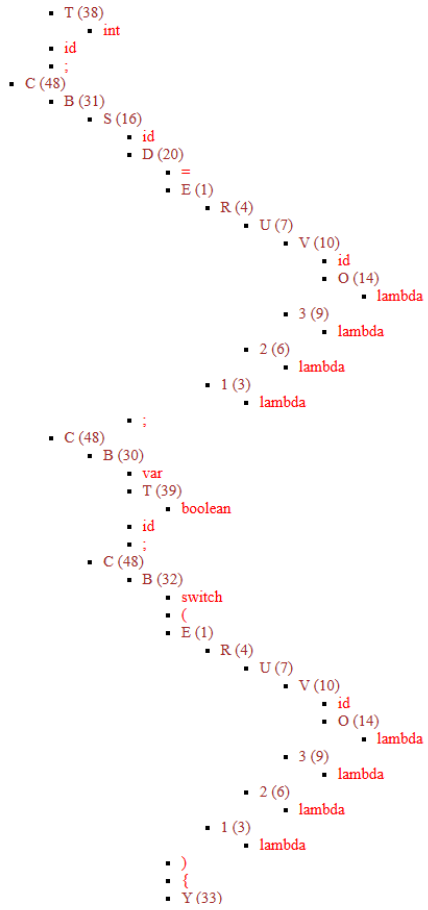
Parse: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Código\Práctica PDL\Códigos de prueba\C1\Parse.txt

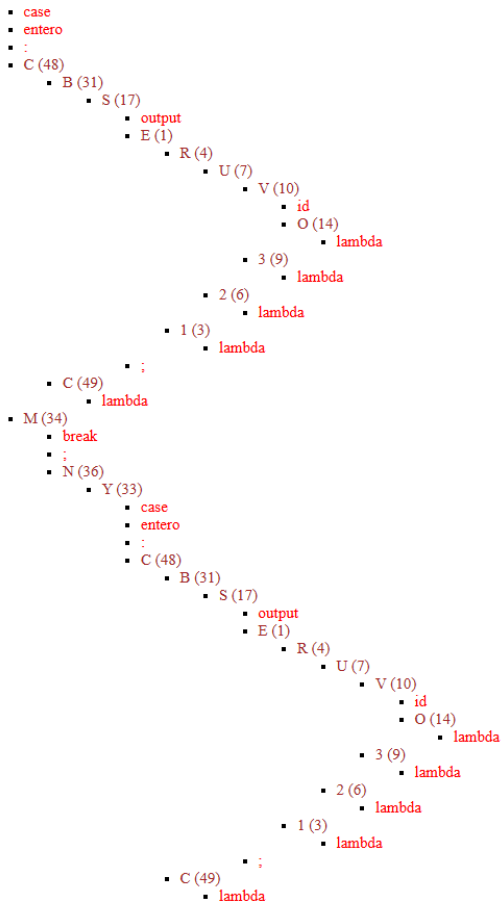
- P (50)
  - B (30)
    - var
    - T (38)
      - int
    - id
    - ;
  - P (50)
    - B (31)
      - S (16)
        - id
        - D (21)
          - *=
          - E (1)
            - R (4)
              - U (7)
                - V (12)
                  - entero
                - 3 (9)
                  - lambda
              - 2 (6)
                - lambda
            - 1 (3)
              - lambda
        - ;
    - P (50)
      - B (30)
        - var
        - T (40)
          - string
        - id
        - ;
      - P (50)
        - B (31)
          - S (16)
            - id
            - D (20)
              - =
              - E (1)
                - R (4)
                  - U (7)
                    - V (13)
                      - cadena
                    - 3 (9)


lambda
2 (6)
lambda
1 (3)
lambda
;
P (50)
B (30)
var
T (39)
boolean
id
;
P (50)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (50)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (50)
B (31)
S (17)
output

- E (1)
  - R (4)
    - U (7)
      - V (13)
        - cadena
      - 3 (9)
        - lambda
    - 2 (6)
      - lambda
  - 1 (3)
    - lambda
- ;

- P (50)
  - B (31)
    - S (18)
      - input
      - id
      - ;
    - P (51)
      - F (41)
        - function
        - H (43)
          - void
        - id
        - (
        - A (44)
          - T (38)
            - int
          - id
          - K (46)
            - ,
            - T (40)
              - string
            - id
            - K (46)
              - ,
              - T (40)
                - string
              - id
              - K (46)
                - ,
                - T (40)
                  - string
                - id
                - K (47)
                  - lambda
        -)
        - {
        - C (48)
          - B (30)
            - var


T (38)
int
id
;
C (48)
B (31)
S (16)
id
D (20)
=
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (30)
var
T (39)
boolean
id
;
C (48)
B (32)
switch
(
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
{
Y (33)


case
entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (36)
Y (33)
case
entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda

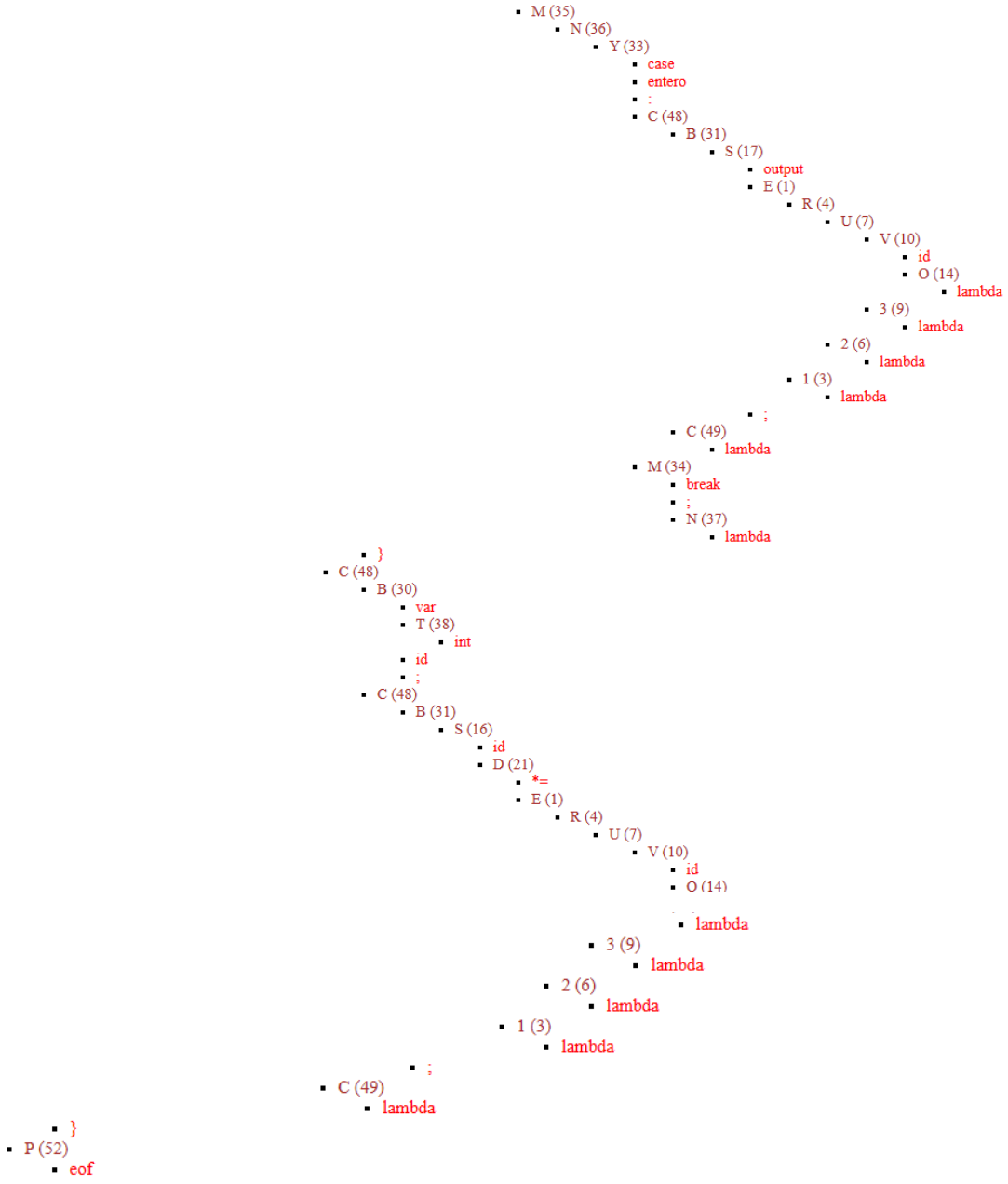

M (35)
N (36)
Y (33)
case
entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (37)
lambda
}
C (48)
B (30)
var
T (38)
int
id
;
C (48)
B (31)
S (16)
id
D (21)
*=
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
}
P (52)
eof

Tabla de Símbolos:

Contenido de la Tabla de Simbolos Global #0 :

---

* LEXEMA : 'x'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 0

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

* LEXEMA : 'cadena'

  ATRIBUTOS:

  + tipo : 'cadena'

  + despl : 1

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

* LEXEMA : 'bol'

  ATRIBUTOS:

  + tipo : 'logico'

  + despl : 65

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

* LEXEMA : 'impriParam'

ATRIBUTOS:

+ tipo : 'funcion'

+ numParam : 4

+ TipoParam1 : 'entero'

+ ModoParam1 : 1

+ TipoParam2 : 'cadena'

+ ModoParam2 : 1

+ TipoParam3 : 'cadena'

+ ModoParam3 : 1

+ TipoParam4 : 'cadena'

+ ModoParam4 : 1

+ TipoRetorno : 'vacio'

+ EtiqFuncion : 'EtimpriParam1'

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

Contenido de la Tabla de Simbolos de la funcion con numero de etiqueta #1 :

---

* LEXEMA : 'valor'

ATRIBUTOS:

+ tipo : 'entero'

+ despl : 0

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

```
* LEXEMA : 'cad1'
  ATRIBUTOS:
  + tipo : 'cadena'
  + despl : 1

--------- ----------
* LEXEMA : 'cad2'
  ATRIBUTOS:
  + tipo : 'cadena'
  + despl : 65

--------- ----------
* LEXEMA : 'cad3'
  ATRIBUTOS:
  + tipo : 'cadena'
  + despl : 129

--------- ----------
* LEXEMA : 'aux'
  ATRIBUTOS:
  + tipo : 'entero'
  + despl : 193

--------- ----------
```

* LEXEMA : 'z'

  ATRIBUTOS:

  + tipo : 'logico'

  + despl : 194

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

* LEXEMA : 'aux2'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 195

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

TOKENS:

<VAR, >

<INT, >

<ID, 0>

<PUNTYCOM, >

<ID, 0>

<ASIGMULT, >

<ENTERO, 6>

<PUNTYCOM, >

<VAR, >

<STRING, >

<ID, 1>

<PUNTYCOM, >

<ID, 1>

<IGUAL, >

<CADENA, "Pruebaaaaaaaaaaaaaaaaa">

<PUNTYCOM, >

<VAR, >

<BOOLEAN, >

<ID, 2>

<PUNTYCOM, >

<OUTPUT, >

<CADENA, "Contenido de cadena: \n">

<PUNTYCOM, >

<OUTPUT, >

<ID, 1>

<PUNTYCOM, >

<OUTPUT, >

<CADENA, "Asigna el valor a la variables x:">

<PUNTYCOM, >

<INPUT, >

<ID, 0>

<PUNTYCOM, >

<FUNCTION, >

<VOID, >

<ID, 3>

<PARENTABRE, >

<INT, >

<ID, 4>

<COMA, >

<STRING, >

<ID, 5>

<COMA, >

<STRING, >

<ID, 6>

<COMA, >

<STRING, >

<ID, 7>

<PARENTCIERRA, >

<LLAVEABRE, >

<VAR, >

<INT, >

<ID, 8>

<PUNTYCOM, >

<ID, 8>

<IGUAL, >

<ID, 4>

<PUNTYCOM, >

<VAR, >

<BOOLEAN, >

<ID, 9>

<PUNTYCOM, >

<SWITCH, >

<PARENTABRE, >

<ID, 4>

<PARENTCIERRA, >

<LLAVEABRE, >

<CASE, >

<ENTERO, 1>

<DOSPUNTOS, >

<OUTPUT, >

<ID, 5>

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 2>

<DOSPUNTOS, >

<OUTPUT, >

<ID, 6>

<PUNTYCOM, >

<CASE, >

<ENTERO, 3>

<DOSPUNTOS, >

<OUTPUT, >

<ID, 7>

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<LLAVECIERRA, >

<VAR, >

<INT, >

<ID, 10>

<PUNTYCOM, >

<ID, 10>

<ASIGMULT, >

<ID, 8>

<PUNTYCOM, >

<LLAVECIERRA, >

<EOF, >

- **Código 2 (sin errores)**

CÓDIGO:

var int num1;

num1 *= 47; /* Número 1 */

```
var int num2;
num2 = 56; /* Número 2 */

output 'Suma de los dos números num1 y num2: ';
output (num1 + num2);

function string cad(int num1, string r){
    if((num1 + num2) == 103)
        return r;

}

output 'Próxima función: hola';

function void hola(void){
    output 'Contenido de la función: ';
    output 'HOLAAAAAAAAAAA';
}

/* **Fin de códigoo
oooooooo*oooo
oo */
```

PARSE:

Descendente 50 30 38 50 31 16 21 1 4 7 12 9 6 3 50 30 38 50 31 16 20 1 4 7 12 9 6 3 50 31 17 1 4 7 13 9 6 3 50 31 17 1 4 7 11 1 4 7 10 14 8 10 14 9 6 3 9 6 3 51 41 42 40 44 38 46 40 47 48 29 1 4 7 11 1 4 7 10 14 8 10 14 9 6 3 9 5 7 12 9 6 3 19 27 1 4 7 10 14 9 6 3 49 50 31 17 1 4 7 13 9 6 3 51 41 43 45 48 31 17 1 4 7 13 9 6 3 48 31 17 1 4 7 13 9 6 3 49 52

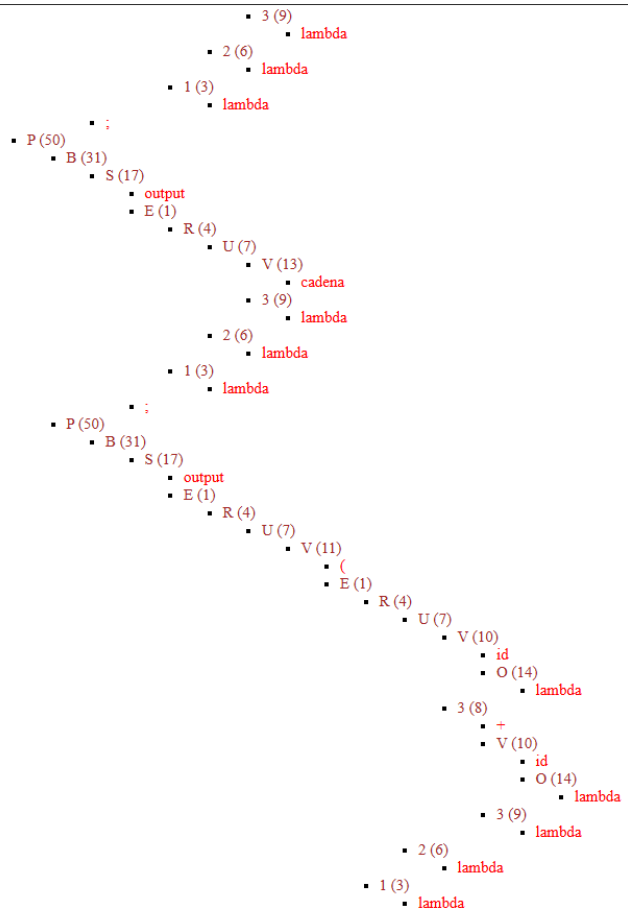
ÁRBOL:

Árbol resultado de:

Gramática: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Utilidades\Documentacion 24-25\Gramática 24-25.txt

Parse: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Código\Práctica PDL\Códigos de prueba\C2\Parse.txt

- P (50)
  - B (30)
    - var
    - T (38)
      - int
    - id
    - ;
  - P (50)
    - B (31)
      - S (16)
        - id
        - D (21)
          - *=
          - E (1)
            - R (4)
              - U (7)
                - V (12)
                  - entero
                - 3 (9)
                  - lambda
              - 2 (6)
                - lambda
            - 1 (3)
              - lambda
        - ;
    - P (50)
      - B (30)
        - var
        - T (38)
          - int
        - id
        - ;
      - P (50)
        - B (31)
          - S (16)
            - id
            - D (20)
              - =
              - E (1)
                - R (4)
                  - U (7)
                    - V (12)
                      - entero


3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (50)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (50)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (11)
(
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (8)
+
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda


)
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (51)
F (41)
function
H (42)
T (40)
string
id
(
A (44)
T (38)
int
id
K (46)
,
T (40)
string
id
K (47)
lambda
)
{
C (48)
B (29)
if
(
E (1)
R (4)
U (7)
V (11)
(
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (8)
+
V (10)
id
O (14)
lambda


O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
3 (9)
lambda
2 (5)
==
U (7)
V (12)
entero
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
S (19)
return
X (27)
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
}
P (50)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)

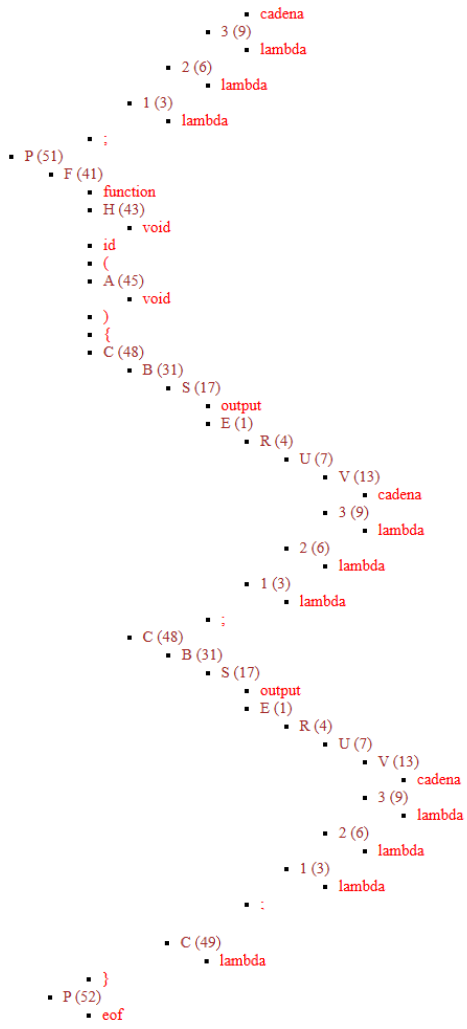


Tabla de Símbolos:

Contenido de la Tabla de Simbolos Global #0 :

---

* LEXEMA : 'num1'

  ATRIBUTOS:

+ tipo : 'entero'

+ despl : 0

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

* LEXEMA : 'num2'

ATRIBUTOS:

+ tipo : 'entero'

+ despl : 1

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

* LEXEMA : 'cad'

ATRIBUTOS:

+ tipo : 'funcion'

+ numParam : 2

+ TipoParam1 : 'entero'

+ ModoParam1 : 1

+ TipoParam2 : 'cadena'

+ ModoParam2 : 1

+ TipoRetorno : 'cadena'

+ EtiqFuncion : 'Etcad1'

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

* LEXEMA : 'hola'

ATRIBUTOS:

+ tipo : 'funcion'

+ numParam : 0

+ TipoRetorno : 'vacio'

+ EtiqFuncion : 'Ethola2'

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

Contenido de la Tabla de Simbolos de la funcion con numero de etiqueta #1 :

---

* LEXEMA : 'num1'

ATRIBUTOS:

+ tipo : 'entero'

+ despl : 0

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

* LEXEMA : 'r'

ATRIBUTOS:

+ tipo : 'cadena'

+ despl : 1

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

Contenido de la Tabla de Simbolos de la funcion con numero de etiqueta #2 :

TOKENS:

<VAR, >

<INT, >

<ID, 0>

<PUNTYCOM, >

<ID, 0>

<ASIGMULT, >

<ENTERO, 47>

<PUNTYCOM, >

<VAR, >

<INT, >

<ID, 1>

<PUNTYCOM, >

<ID, 1>

<IGUAL, >

<ENTERO, 56>

<PUNTYCOM, >

<OUTPUT, >

<CADENA, "Suma de los dos números num1 y num2: ">

<PUNTYCOM, >

<OUTPUT, >

<PARENTABRE, >

<ID, 0>

<SUMA, >

<ID, 1>

<PARENTCIERRA, >

<PUNTYCOM, >

<FUNCTION, >

<STRING, >

<ID, 2>

<PARENTABRE, >

<INT, >

<ID, 0>

<COMA, >

<STRING, >

<ID, 3>

<PARENTCIERRA, >

<LLAVEABRE, >

<IF, >

<PARENTABRE, >

<PARENTABRE, >

<ID, 0>

<SUMA, >

<ID, 1>

<PARENTCIERRA, >

<IGUIGU, >

<ENTERO, 103>

<PARENTCIERRA, >

<RETURN, >

<ID, 3>

<PUNTYCOM, >

<LLAVECIERRA, >

<OUTPUT, >

<CADENA, "Próxima función: hola">

<PUNTYCOM, >

<FUNCTION, >

<VOID, >

<ID, 4>

<PARENTABRE, >

<VOID, >

<PARENTCIERRA, >

<LLAVEABRE, >

<OUTPUT, >

<CADENA, "Contenido de la función: ">

<PUNTYCOM, >

<OUTPUT, >

<CADENA, "HOLAAAAAAAAAAAA">

<PUNTYCOM, >

<LLAVECIERRA, >

<EOF, >

- **Código 3 (sin errores)**

CÓDIGO:

```
/* Variables globales */
var int contadorGlobal1;

/* Función para clasificar números */
function void clasificarNumero (int numero) {
    switch (numero) {
        case 1:
            output 'Es uno';
            break;
        case 2:
            output 'Es dos';
            break;
        case 3:
            output 'Es tres';
    }
}

/* Función principal */
```

```
function void main(void) {
    /* Variables locales */
    var int numero;

    numero *= 2;
    var int resultado;

    /* Mostrar mensaje global */
    output mensajeGlobal;

    /* Clasificar un número */
    clasificarNumero(numero);

    /* Operador especial: Asignación con multiplicación */
    numero *= 3;
    output 'Resultado después de multiplicar: ';
    output numero;

    /* Actualización del contador global */
    contadorGlobal *= 2;
    output 'Valor del contador global: ';
    output contadorGlobal;
}
```

/* Iniciar el programa */

main();

PARSE:

Descendente 50 30 38 51 41 43 44 38 47 48 32 1 4 7 10 14 9 6 3 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 35 37 49 51 41 43 45 48 30 38 48 31 16 21 1 4 7 12 9 6 3 48 30 38 48 31 17 1 4 7 10 14 9 6 3 48 31 16 22 23 1 4 7 10 14 9 6 3 26 48 31 16 21 1 4 7 12 9 6 3 48 31 17 1 4 7 13 9 6 3 48 31 17 1 4 7 10 14 9 6 3 48 31 16 21 1 4 7 12 9 6 3 48 31 17 1 4 7 13 9 6 3 48 31 17 1 4 7 10 14 9 6 3 49 50 31 16 22 24 52

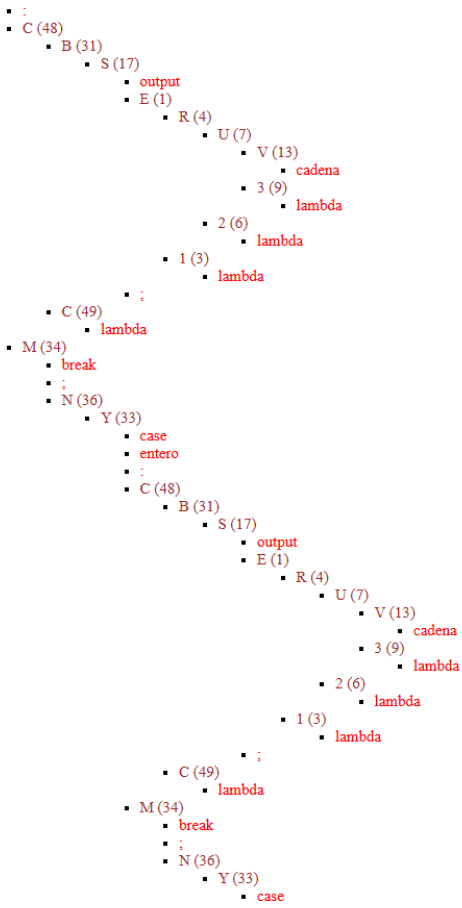
ÁRBOL:

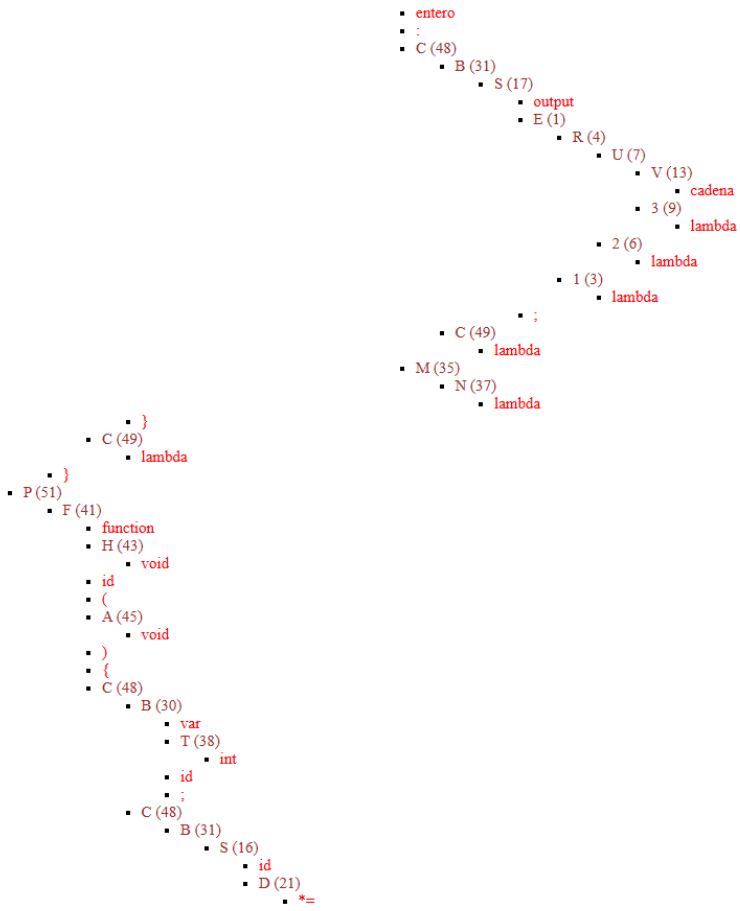
Árbol resultado de:

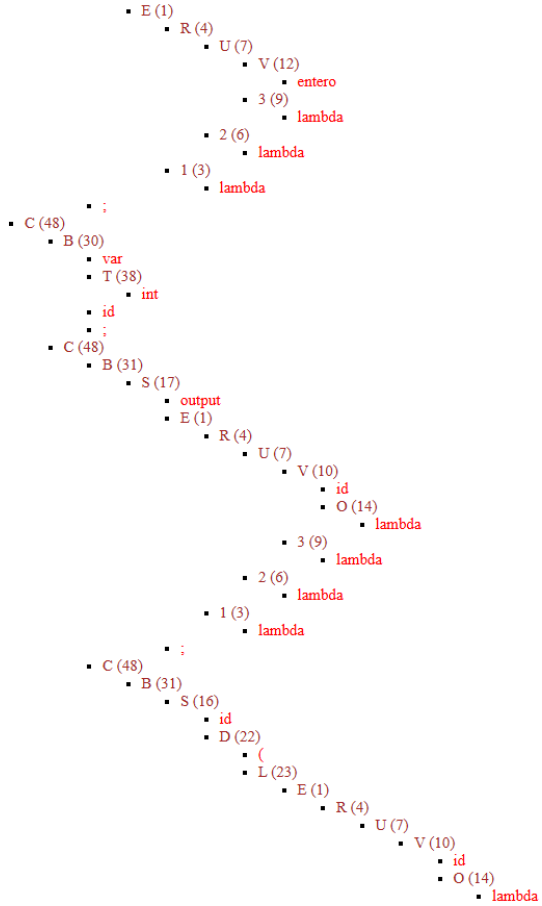
Gramática: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Utilidades\Documentacion 24-25\Gramática 24-25.txt

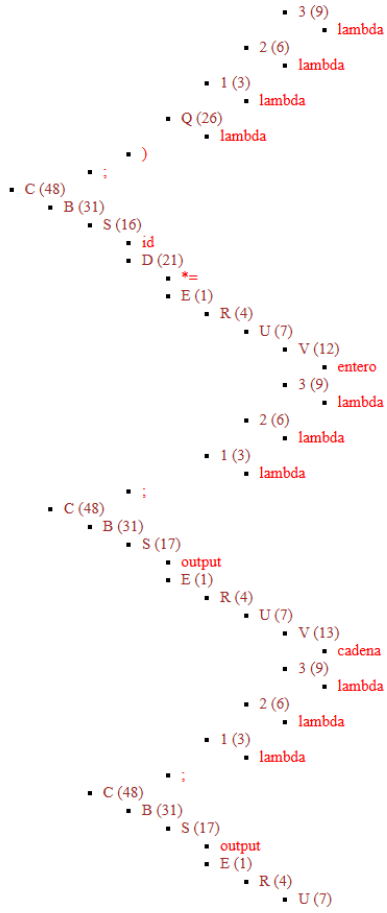
Parse: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Código\Práctica PDL\Códigos de prueba\C3\Parse.txt

- P (50)
  - B (30)
    - var
    - T (38)
      - int
    - id
    - ;
  - P (51)
    - F (41)
      - function
      - H (43)
        - void
      - id
      - (
      - A (44)
        - T (38)
          - int
        - id
        - K (47)
          - lambda
      -)
      - {
      - C (48)
        - B (32)
          - switch
          - (
          - E (1)
            - R (4)
              - U (7)
                - V (10)
                  - id
                  - O (14)
                    - lambda
                - 3 (9)
                  - lambda
              - 2 (6)
                - lambda
            - 1 (3)
              - lambda
          -)
          - {
          - Y (33)
            - case
            - entero


:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (36)
Y (33)
case
entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (36)
Y (33)
case


entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (35)
N (37)
lambda
}
C (49)
lambda
}
P (51)
F (41)
function
H (43)
void
id
(
A (45)
void
)
{
C (48)
B (30)
var
T (38)
int
id
;
C (48)
B (31)
S (16)
id
D (21)
*=


E (1)
R (4)
U (7)
V (12)
entero
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (30)
var
T (38)
int
id
;
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (31)
S (16)
id
D (22)
(
L (23)
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda


3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (26)
lambda
)
;
C (48)
B (31)
S (16)
id
D (21)
*=
E (1)
R (4)
U (7)
V (12)
entero
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)

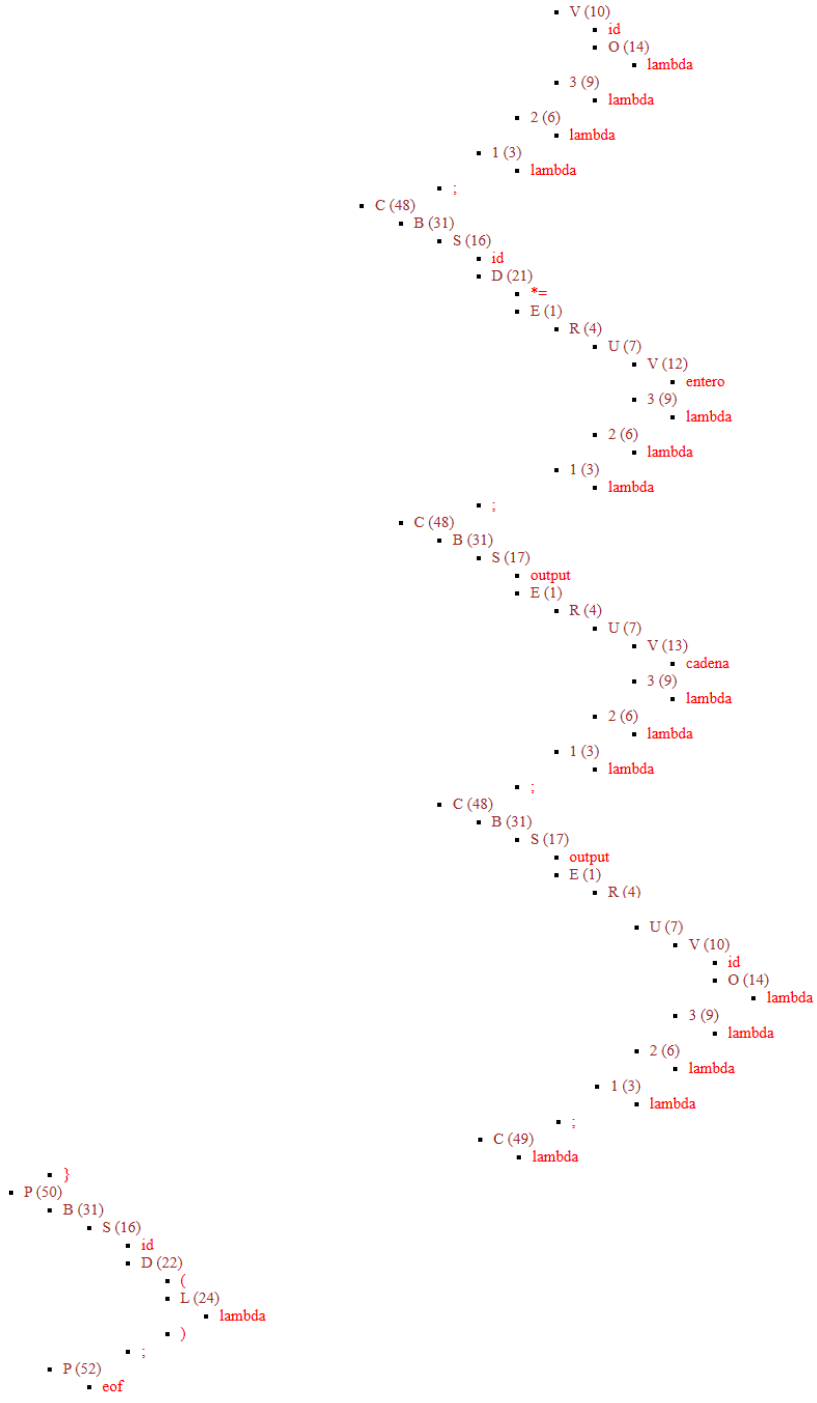

V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (31)
S (16)
id
D (21)
*=
E (1)
R (4)
U (7)
V (12)
entero
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
}
P (50)
B (31)
S (16)
id
D (22)
(
L (24)
lambda
)
;
P (52)
eof

TABLA DE SÍMBOLOS:

Contenido de la Tabla de Simbolos Global #0 :

---

* LEXEMA : 'contadorGlobal1'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 0

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

* LEXEMA : 'mensajeGlobal'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 1

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

* LEXEMA : 'contadorGlobal'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 2

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

* LEXEMA : 'clasificarNumero'

  ATRIBUTOS:

```
  + tipo : 'funcion'
  + numParam : 1
  + TipoParam1 : 'entero'
  + ModoParam1 : 1
  + TipoRetorno : 'vacio'
  + EtiqFuncion : 'EtclasificarNumero1'


--------- ----------
* LEXEMA : 'main'
  ATRIBUTOS:
  + tipo : 'funcion'
  + numParam : 0
  + TipoRetorno : 'vacio'
  + EtiqFuncion : 'Etmain2'


--------- ----------
------------------------------------------------
Contenido de la Tabla de Simbolos de la funcion con numero de etiqueta #1 :
------------------------------------------------
* LEXEMA : 'numero'
  ATRIBUTOS:
  + tipo : 'entero'
  + despl : 0
```

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

Contenido de la Tabla de Simbolos de la funcion con numero de etiqueta #2 :

----

* LEXEMA : 'numero'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 0

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

* LEXEMA : 'resultado'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 1

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

TOKENS:

<VAR, >

<INT, >

<ID, 0>

<PUNTYCOM, >

<FUNCTION, >

<VOID, >

<ID, 1>

<PARENTABRE, >

<INT, >

<ID, 2>

<PARENTCIERRA, >

<LLAVEABRE, >

<SWITCH, >

<PARENTABRE, >

<ID, 2>

<PARENTCIERRA, >

<LLAVEABRE, >

<CASE, >

<ENTERO, 1>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Es uno">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 2>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Es dos">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 3>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Es tres">

<PUNTYCOM, >

<LLAVECIERRA, >

<LLAVECIERRA, >

<FUNCTION, >

<VOID, >

<ID, 3>

<PARENTABRE, >

<VOID, >

<PARENTCIERRA, >

<LLAVEABRE, >

<VAR, >

<INT, >

<ID, 2>

<PUNTYCOM, >

<ID, 2>

<ASIGMULT, >

<ENTERO, 2>

<PUNTYCOM, >

<VAR, >

<INT, >

<ID, 4>

<PUNTYCOM, >

<OUTPUT, >

<ID, 5>

<PUNTYCOM, >

<ID, 1>

<PARENTABRE, >

<ID, 2>

<PARENTCIERRA, >

<PUNTYCOM, >

<ID, 2>

<ASIGMULT, >

<ENTERO, 3>

<PUNTYCOM, >

<OUTPUT, >

<CADENA, "Resultado después de multiplicar: ">

<PUNTYCOM, >

<OUTPUT, >

<ID, 2>

<PUNTYCOM, >

<ID, 6>

<ASIGMULT, >

<ENTERO, 2>

<PUNTYCOM, >

<OUTPUT, >

<CADENA, "Valor del contador global: ">

<PUNTYCOM, >

<OUTPUT, >

<ID, 6>

<PUNTYCOM, >

<LLAVECIERRA, >

<ID, 3>

<PARENTABRE, >

<PARENTCIERRA, >

<PUNTYCOM, >

<EOF, >

- **Código 4 (sin errores)**

CÓDIGO:

/* Código corto pero eficiente */

var boolean z;

```
var int num1;
num1 = 4456;

output '--PRINT DE NUM1 --';
output num1;



/* Función para determinar el día de la semana */
function void determinarDia(int dia) {
    switch (dia) {
        case 1:
            output 'Lunes';
            break;
        case 2:
            output 'Martes';
            break;
        case 3:
            output 'Miércoles';
            break;
        case 4:
            output 'Jueves';
            break;
        case 5:
```

```
            output 'Viernes';
            break;
        case 6:
            output 'Sábado';
            break;
        case 7:
            output 'Domingo';
            break;
    }
    var string afterSwitch;
    output
    afterSwitch;
}
```

PARSE:

Descendente 50 30 39 50 30 38 50 31 16 20 1 4 7 12 9 6 3 50 31 17 1 4 7 13 9 6 3 50 31 17 1 4 7 10 14 9 6 3 51 41 43 44 38 47 48 32 1 4 7 10 14 9 6 3 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 34 37 48 30 40 48 31 17 1 4 7 10 14 9 6 3 49 52

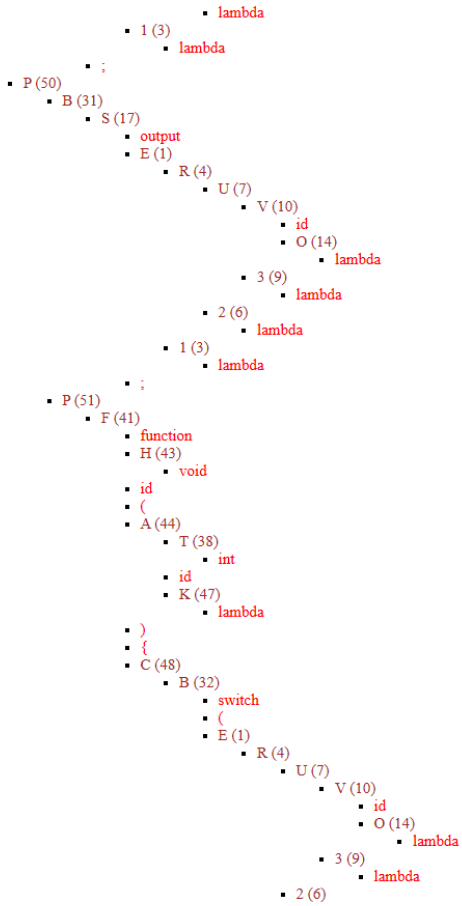
ÁRBOL:

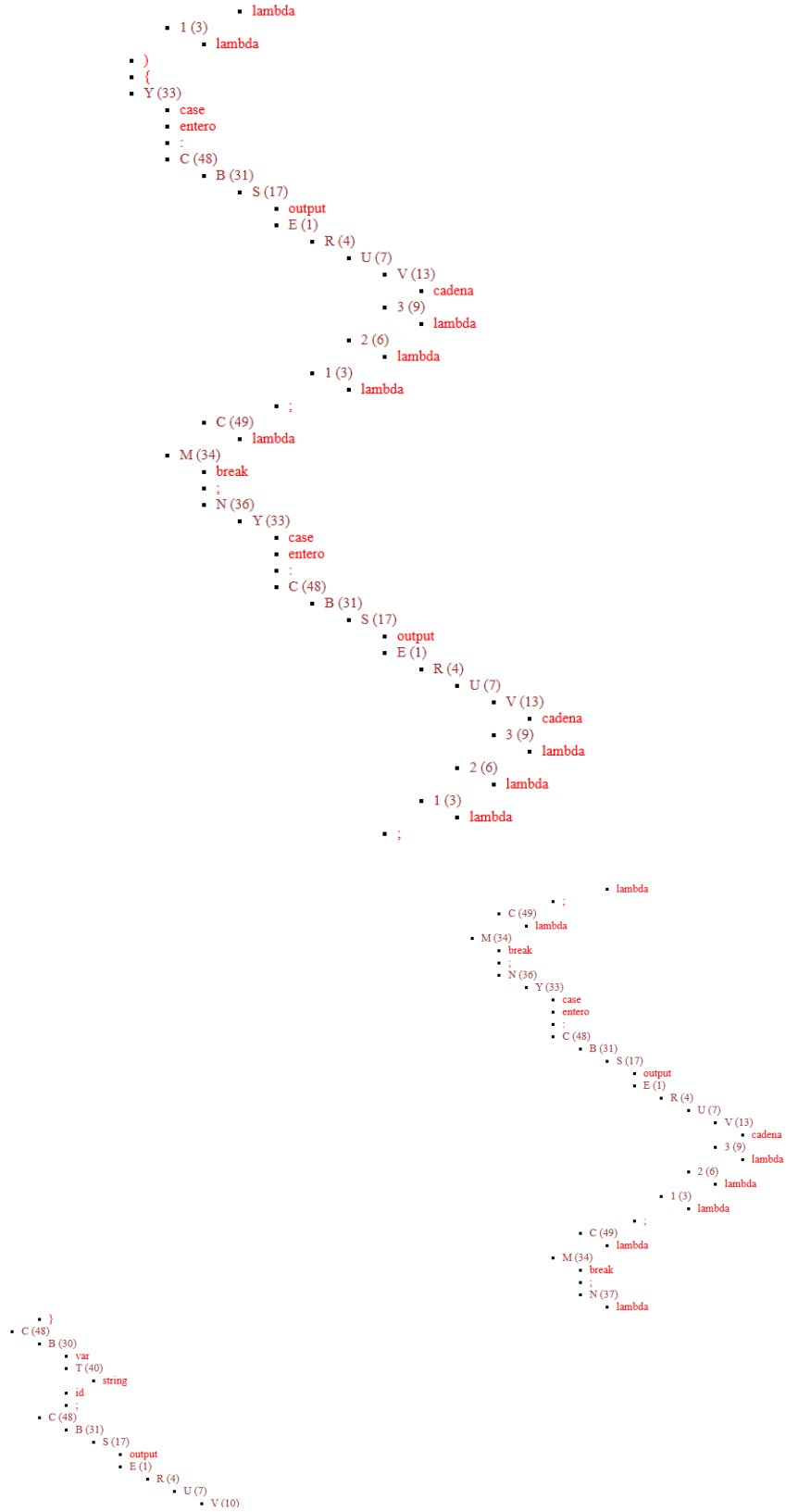
Árbol resultado de:

Gramática: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Utilidades\Documentacion 24-25\Gramática 24-25.txt

Parse: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Código\Práctica PDL\Códigos de prueba\C4\Parse.txt

- P (50)
  - B (30)
    - var
    - T (39)
      - boolean
    - id
    - ;
  - P (50)
    - B (30)
      - var
      - T (38)
        - int
      - id
      - ;
    - P (50)
      - B (31)
        - S (16)
          - id
          - D (20)
            - =
            - E (1)
              - R (4)
                - U (7)
                  - V (12)
                    - entero
                  - 3 (9)
                    - lambda
                - 2 (6)
                  - lambda
              - 1 (3)
                - lambda
          - ;
      - P (50)
        - B (31)
          - S (17)
            - output
            - E (1)
              - R (4)
                - U (7)
                  - V (13)
                    - cadena
                  - 3 (9)
                    - lambda
                - 2 (6)


lambda
1 (3)
lambda
;
P (50)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (51)
F (41)
function
H (43)
void
id
(
A (44)
T (38)
int
id
K (47)
lambda
)
{
C (48)
B (32)
switch
(
E (1)
R (4)
U (7)
V (10)
id
O (14)
lambda
3 (9)
lambda
2 (6)


lambda
1 (3)
lambda
)
{
Y (33)
case
entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (36)
Y (33)
case
entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
lambda
;
C (49)
lambda
M (34)
break
;
N (36)
Y (33)
case
entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (37)
lambda
}
C (48)
B (30)
var
T (40)
string
id
;
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (10)

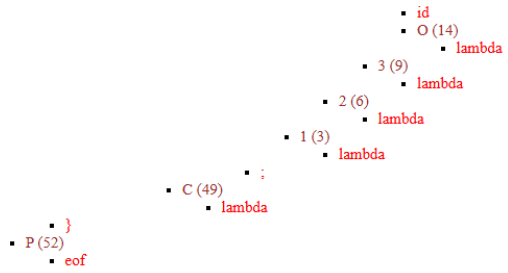


TABLA DE SÍMBOLOS:

Contenido de la Tabla de Simbolos Global #0 :

---

* LEXEMA : 'z'

  ATRIBUTOS:

  + tipo : 'logico'

  + despl : 0

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

* LEXEMA : 'num1'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 1

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

* LEXEMA : 'determinarDia'

ATRIBUTOS:

+ tipo : 'funcion'

+ numParam : 1

+ TipoParam1 : 'entero'

+ ModoParam1 : 1

+ TipoRetorno : 'vacio'

+ EtiqFuncion : 'EtdeterminarDia1'

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

Contenido de la Tabla de Simbolos de la funcion con numero de etiqueta #1 :

---

* LEXEMA : 'dia'

ATRIBUTOS:

+ tipo : 'entero'

+ despl : 0

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

* LEXEMA : 'afterSwitch'

ATRIBUTOS:

+ tipo : 'cadena'

+ despl : 1

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

TOKENS:

<VAR, >

<BOOLEAN, >

<ID, 0>

<PUNTYCOM, >

<VAR, >

<INT, >

<ID, 1>

<PUNTYCOM, >

<ID, 1>

<IGUAL, >

<ENTERO, 4456>

<PUNTYCOM, >

<OUTPUT, >

<CADENA, "--PRINT DE NUM1 --">

<PUNTYCOM, >

<OUTPUT, >

<ID, 1>

<PUNTYCOM, >

<FUNCTION, >

<VOID, >

<ID, 2>

<PARENTABRE, >

<INT, >

<ID, 3>

<PARENTCIERRA, >

<LLAVEABRE, >

<SWITCH, >

<PARENTABRE, >

<ID, 3>

<PARENTCIERRA, >

<LLAVEABRE, >

<CASE, >

<ENTERO, 1>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Lunes">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 2>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Martes">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 3>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Miércoles">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 4>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Jueves">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 5>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Viernes">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 6>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Sábado">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 7>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Domingo">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<LLAVECIERRA, >

<VAR, >

<STRING, >

<ID, 4>

<PUNTYCOM, >

<OUTPUT, >

<ID, 4>

<PUNTYCOM, >

<LLAVECIERRA, >

<EOF, >

- **Código 5 (sin errores)**

CÓDIGO:

```
/* Función para evaluar un número */
function void evaluarNumero(int numero) {
    switch (numero) {
        case 10:
            output 'Es un diez';
            break;
        case 20:
            output 'Es un veinte';
            break;
        case 30:
            output 'Es un treinta';
            break;
    }
}
```

```
/* Función principal */
function void main(void) {
    var int valor;

    /* Evaluar el número */
    evaluarNumero(valor);

    /* Actualizar el valor */
    valor *= 2;
    output 'El valor actualizado es: ';
    output valor;
}

/* Iniciar el programa */
main();
```

PARSE:

Descendente 51 41 43 44 38 47 48 32 1 4 7 10 14 9 6 3 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 34 36 33 48 31 17 1 4 7 13 9 6 3 49 34 37 49 51 41 43 45 48 30 38 48 31 16 22 23 1 4 7 10 14 9 6 3 26 48 31 16 21 1 4 7 12 9 6 3 48 31 17 1 4 7 13 9 6 3 48 31 17 1 4 7 10 14 9 6 3 49 50 31 16 22 24 52

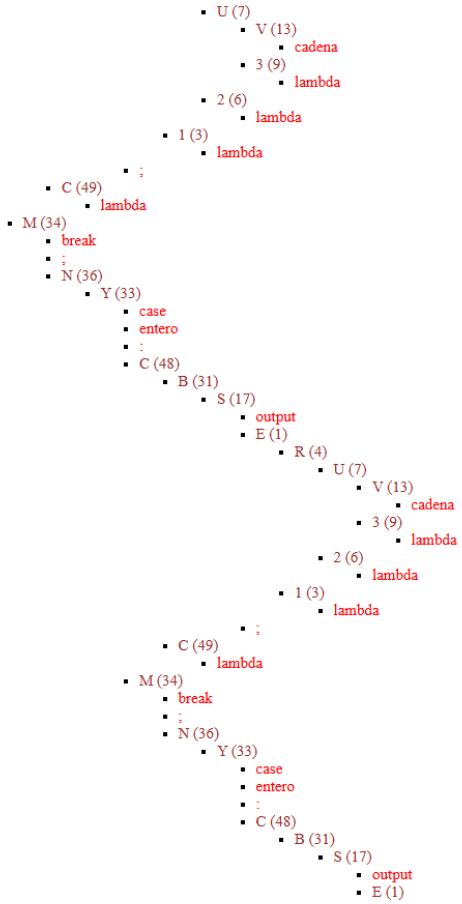
ÁRBOL:

Árbol resultado de:

Gramática: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Utilidades\Documentacion 24-25\Gramática 24-25.txt

Parse: C:\Users\Alvar\OneDrive\Escritorio\Uni\Asignaturas\3º\1er cuatri\Procesadores de Lenguajes\Práctica\PDL-24-25\Código\Práctica PDL\Códigos de prueba\C5\Parse.txt

- P (51)
  - F (41)
    - function
    - H (43)
      - void
    - id
    - (
    - A (44)
      - T (38)
        - int
      - id
      - K (47)
        - lambda
    -)
    - {
    - C (48)
      - B (32)
        - switch
        - (
        - E (1)
          - R (4)
            - U (7)
              - V (10)
                - id
                - O (14)
                  - lambda
              - 3 (9)
                - lambda
            - 2 (6)
              - lambda
          - 1 (3)
            - lambda
        -)
        - {
        - Y (33)
          - case
          - entero
          - :
          - C (48)
            - B (31)
              - S (17)
                - output
                - E (1)
                  - R (4)


U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (36)
Y (33)
case
entero
:
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (36)
Y (33)
case
entero
:
C (48)
B (31)
S (17)
output
E (1)


R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (49)
lambda
M (34)
break
;
N (37)
lambda
}
C (49)
lambda
}
P (51)
F (41)
function
H (43)
void
id
(
A (45)
void
)
{
C (48)
B (30)
var
T (38)
int
id
;
C (48)
B (31)
S (16)
id
D (22)
(
L (23)
E (1)
R (4)
U (7)
V (10)

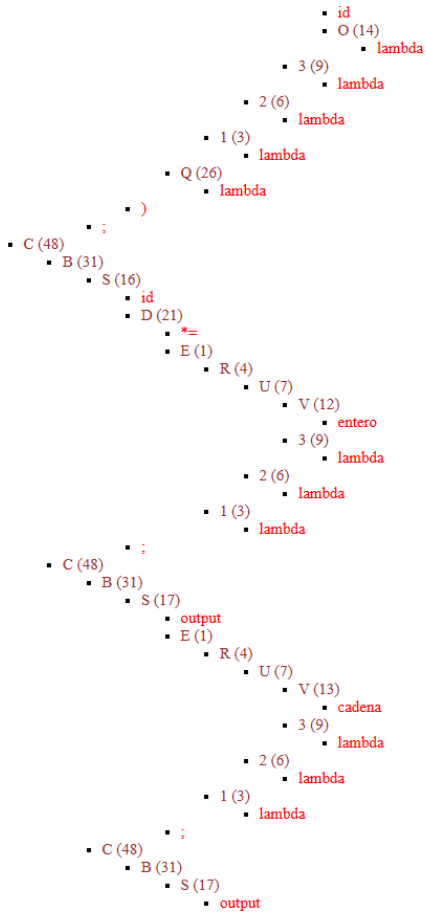

id
O (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (26)
lambda
)
;
C (48)
B (31)
S (16)
id
D (21)
*=
E (1)
R (4)
U (7)
V (12)
entero
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (31)
S (17)
output
E (1)
R (4)
U (7)
V (13)
cadena
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (48)
B (31)
S (17)
output

TABLA DE SÍMBOLOS:

Contenido de la Tabla de Simbolos Global #0 :

---

* LEXEMA : 'evaluarNumero'

  ATRIBUTOS:

  + tipo : 'funcion'

  + numParam : 1

  + TipoParam1 : 'entero'

+ ModoParam1 : 1

+ TipoRetorno : 'vacio'

+ EtiqFuncion : 'EtevaluarNumero1'

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

* LEXEMA : 'main'

ATRIBUTOS:

+ tipo : 'funcion'

+ numParam : 0

+ TipoRetorno : 'vacio'

+ EtiqFuncion : 'Etmain2'

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

Contenido de la Tabla de Simbolos de la funcion con numero de etiqueta #1 :

---

* LEXEMA : 'numero'

ATRIBUTOS:

+ tipo : 'entero'

+ despl : 0

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

Contenido de la Tabla de Simbolos de la funcion con numero de etiqueta #2 :

---

* LEXEMA : 'valor'

  ATRIBUTOS:

  + tipo : 'entero'

  + despl : 0

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

TOKENS:

<FUNCTION, >

<VOID, >

<ID, 0>

<PARENTABRE, >

<INT, >

<ID, 1>

<PARENTCIERRA, >

<LLAVEABRE, >

<SWITCH, >

<PARENTABRE, >

<ID, 1>

<PARENTCIERRA, >

<LLAVEABRE, >

<CASE, >

<ENTERO, 10>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Es un diez">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 20>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Es un veinte">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<CASE, >

<ENTERO, 30>

<DOSPUNTOS, >

<OUTPUT, >

<CADENA, "Es un treinta">

<PUNTYCOM, >

<BREAK, >

<PUNTYCOM, >

<LLAVECIERRA, >

<LLAVECIERRA, >

<FUNCTION, >

<VOID, >

<ID, 2>

<PARENTABRE, >

<VOID, >

<PARENTCIERRA, >

<LLAVEABRE, >

<VAR, >

<INT, >

<ID, 3>

<PUNTYCOM, >

<ID, 0>

<PARENTABRE, >

<ID, 3>

<PARENTCIERRA, >

<PUNTYCOM, >

<ID, 3>

<ASIGMULT, >

<ENTERO, 2>

<PUNTYCOM, >

<OUTPUT, >

<CADENA, "El valor actualizado es: ">

<PUNTYCOM, >

<OUTPUT, >

<ID, 3>

<PUNTYCOM, >

<LLAVECIERRA, >

<ID, 2>

<PARENTABRE, >

<PARENTCIERRA, >

<PUNTYCOM, >

<EOF, >

- **Código 6 (con errores)**

CÓDIGO:

```
/* Código con error léxico*/

var int valorGlobal;
valorGlobal = 10;
var string mensaje;
mensaje  =
'Error léxico';

/* Función principal */
function void main(void) {
    var int numero;
```

```
    /* Uso de un símbolo no válido */
    numero #= 2; /* El Gestor de Errores detectará el error correctamente. */

    /* Mostrar resultado */
    output 'Valor actualizado: ';
    output numero;
}

/* Iniciar el programa */
main();
```

ERROR:

Error Léxico en línea 14: Carácter '#' no soportado o desconocido por el lenguaje.

- **Código 7 (Con errores)**

CÓDIGO:

```
/* Variables globales */
var int numero5;
var string mensaje;
mensaje = 'Este es un mensaje';
```

```
/* Función principal */
function void main(void) {
    /* Intento de asignar una cadena a una variable entera */
    numero = 'Error semántico'; /* Error: Asignación de un string a una variable int */

    /* Mostrar resultados */
    output 'El valor de la variable es: ';
    output numero;
}
```

ERROR:

Error Semántico en línea 9: Tipo incompatible para la asignación de la variable 'numero'.

- **Código 8 (Con error)**

CÓDIGO:

```
output 'Se presenta un error semántico:';
output 'El gestor \n de errores lo tratará debidamente';

/* Comentario inutil con variable inutil */
var boolean boola;

/* Función que devuelve la suma de 3 variables no declaradas previamente, y
por tanto, se convierten en globales y enteras */
```

```
function int noDeclarada(void){
    return num1 + num2 + num3;
}

/* Función para clasificar un número */
function void clasificar(int numero) {
    var string cad;
    cad = 'cadena';
    switch (cad) { /* Error: Se usa un string en lugar de un entero */
        case 1:
            output 'Es uno';
            break;
        case 2:
            output 'Es dos';
            break;
        case 3:
            output 'Es tres';
            break;
    }
    var int nunca;
    var boolean sellega;
}

/* Nunca se llega aquí...*/
```

ERROR:
Error Semántico en línea 17: La entrada con lexema 'cad' no es de tipo entero y debe serlo, ya que se está evaluando en un switch.

- **Código 9 (Con errores)**

CÓDIGO:

```
/* Código con error semántico en la llamada a una función */

/* Función que suma dos números enteros */
function int suma(int a, int b) {
    return a + b;
}

/* Función principal */
function void main(void) {
    var string mensaje;

    /* Llamada con un parámetro de tipo incorrecto */
    output suma(5, mensaje); /* Error: Se pasa un string donde se espera un int */

    output 'El resultado es: ';
```

```
    output resultado;
}
```

ERROR:

Error Semántico en línea 13: Se está llamando a la función 'suma' con un parámetro de tipo incorrecto.

- **Código 10 (Con errores)**

Código:

```
/* Función principal sin tipo de retorno --> Error sintáctico */
function main(void) {
    numero = 10;

    /* Mostrar el valor de la variable */
    output 'El valor de la variable es: ';
    output numero;
}

/* Iniciar el programa */
main();
```

Errores:
Error Sintáctico en línea 2: La instrucción 'function' debe especificar un tipo de retorno válido (int, boolean, string o void).