MEMORIA FINAL PROCESADORES DE LENGUAJES

ENTREGA FINAL - PDL

CONTENIDO

- **Diseño del Analizador Léxico** actualizado: tokens, gramática, autómata, acciones semánticas y errores.
- **Diseño del Analizador Sintáctico** actualizado: gramática, demostración de que la gramática es adecuada para el método de Análisis Sintáctico asignado, y las tablas, autómata o procedimientos de dicho Analizador.
- **Diseño del Analizador Semántico**: Traducción Dirigida por la Sintaxis con las acciones semánticas.
- **Diseño de la Tabla de Símbolos** completa: descripción de su estructura final y organización.

DAVID DE FRUTOS ZAFRA
PABLO GARCÍA GARCÍA
GRUPO 119

ANALIZADOR LÉXICO

DISEÑO DE TOKENS

Constante Entera: <NumEnt, valor>

Identificador: <Id, posTS>

String: <String, ->

Int: <Int, ->

Boolean: <Boolean, ->

If: <If, ->

Else: <Else, ->

Let:<Let, ->

Print: <Print, ->

Input: <Input, ->

Return: <Return, ->

Function: <Function, ->

Cadena: <Cadena, Lexema>

Operador suma: <Suma, ->

Operador resta: <Resta, ->

Operador de asignación: <Asig, ->

Operador de asignación con resta: <AsigResta, ->

Operador Lógico And: <And, ->

Operador Menor: <Menor, ->

Operador Mayor: <Mayor, ->

Paréntesis abierto: <PAbierto, ->

Paréntesis cerrado: <PCerrado ->

Llave Abierta: <KAbierta, ->

Llave Cerrada: <KCerrada, ->

Coma: <Coma, ->

Punto y Coma: <PuntComa, ->

DISEÑO DE GRAMÁTICA

S → LA | lA | dB| -C | /D | = | < | >| ‘E | &G | + | (|) | { | } | , | ; | del S

A → LA | lA | dA | _A | λ

B → dB | λ

C → = | λ

D → /F

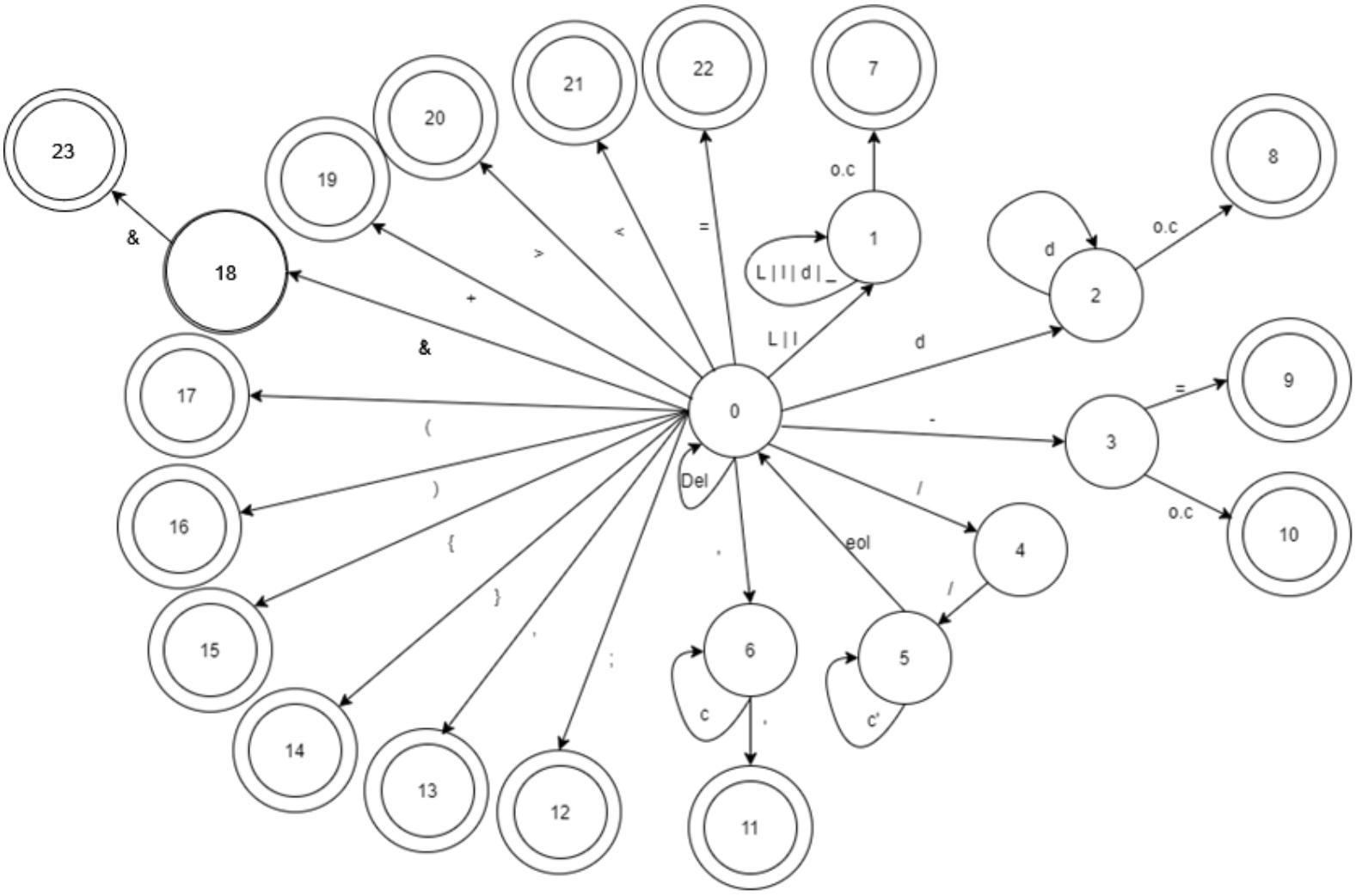
E → cE | ’

F → c’F | eolS

G → &

L : letra mayúscula	l: letra minúscula
d: digíto (0…9)	
c: cualquier tipo de carácter o símbolo excepto: ’	
c’: cualquier tipo de carácter que no sea eol	

DISEÑO DE AUTÓMATA



ACCIONES SEMÁNTICAS

Para comprobar las acciones semánticas que se generan se verán paso a paso cada una de las transiciones del autómata previamente diseñado.

0-0: Leer

0-1: lexema = l | L; Leer

1-1: lexema += (l | L | d | _); Leer

1-7: pal = buscarPalReservada(lexema)

If pal != null -> Gen_Token(pal,-);Leer

else buscar lexema en TS

if no esta -> añadir a TS: Gen_Token(Id,posTS); Leer

0-2: valor = valorA(d); Leer

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

2-8: if valor > 32767 then Error("El máximo entero valido será el 32767")

else Gen_Token(NumEnt,valor); Leer

0-3: Leer

3-9: Gen_Token(AsigResta,-); Leer

3-10: Gen_Token(Resta,-); Leer

0-4: Leer

4-5: Leer

5-5: Leer

5-0: Leer

0-6: cont = 0; Leer

6-6: lexema += c; cont += 1; Leer

6-11: if cont > 64 then Error("Una cadena no puede contener más de 64 caracteres")

else Gen_Token(Cadena, lexema); Leer

0-12: Gen_Token(PuntComa,-); Leer

0-13: Gen_Token(Coma,-); Leer

0-14: Gen_Token(KCerrada,-); Leer

0-15: Gen_Token(KAbierta,-); Leer

0-16: Gen_Token(PCerrado,-); Leer

0-17: Gen_Token(PAbierto,-); Leer

0-18: Leer

18-23: Gen_Token(And,-); Leer

0-19: Gen_Token(Suma,-); Leer

0-20: Gen_Token(Mayor,-); Leer

0-21: Gen_Token(Menor,-); Leer

0-22: Gen_Token(Asig,-); Leer

ERRORES

Los ya descritos en las transiciones 2-8 y 6-11 en referencia a las limitaciones del lenguaje en la representación de enteros y de cadenas.

Si llega un carácter que no pertenece al alfabeto de entrada.

Cualquier transición no descrita en el autómata generará un caso de error.

ANALIZADOR SINTÁCTICO

DISEÑO DE GRAMÁTICA

P → BP | FP | λ

B → let T id; | if (E) G| S

T → int | string | boolean

G → S | { C } O

C → BC | λ

O → else { C } | λ

S → id W | print (E); | input (id);| return X;

W → -=E; | =E; | (L);

X → E | λ

L → EQ | λ

Q → , EQ | λ

F → function id H (A) { C }

H → T | λ

A → T id K | λ

K → , T id K | λ

E → RE'

E' → &&RE' | λ

R → UR'

R' → <UR' | >UR' | λ

U → VU'

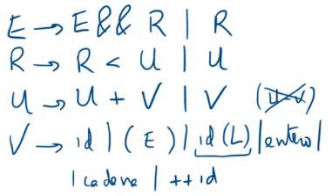
U' → +VU' | -VU' | λ

V → id D | (E) | entero | cadena

D → (L) | λ

Hemos comprobado que nuestra gramática cumple las propiedades de la gramática LL:

- Hemos eliminado la recursividad por la izquierda transformando las reglas desde la gramática sugerida que suponían un problema. Como se puede ver en la imagen



- No es ambigua para ninguna regla
- Esta factorizada dado que ningún consecuente de dos o más reglas de un No terminal comienza igual.

DISEÑO TABLA LL

	let	id	;	if	int	string	boolean	(	)	{	}	else	print	input	return	-	=	-=	,	function	&&	<	>	+	entero	cadena	$
P	P → BP	P → BP		P → BP									P → BP	P → BP	P → BP					P → FP							P → λ
B	B → let T id;	B → S		B → if G									B → S	B → S	B → S												
T					T → int	T → string	T → boolean																				
G		G → S								G → { C } O			G → S	G → S	G → S												
C	C → BC	C → BC		C → BC							C → λ		C → BC	C → BC	C → BC												
O	O → λ	O → λ		O → λ								O → else { C }	O → λ	O → λ	O → λ					O → λ							
S		S → id W											S → print (E);	S → input (id)	S → return X												
W								W → (L);									W → =E;	W → -=E;									
X		X → E	X → λ					X → E																		X → E	X → E	
L		L → EQ						L → EQ	L → λ																	L → EQ	L → EQ	
Q									Q → λ										Q → , EQ								
F																				F → function id H (A) { C }							
H					H → T	H → T	H → T	H → λ																			
A					A → T id K	A → T id K	A → T id K		A → λ																		
K									K → λ										K → , T id K								
E		E → RE'						E → RE'																	E → RE'	E → RE'	
E'			E' → λ						E' → λ										E' → λ		E' → &&RE'						
R		R → UR'						R → UR'																	R → UR'	R → UR'	
R'			R' → λ						R' → λ										R' → λ		R' → λ	R' → <UR'	R' → >UR'				
U		U → VU'						U → VU'																	U → VU'	U → VU'	
U'			U' → λ						U' → λ							U' → -VU'			U' → λ		U' → λ	U' → λ	U' → λ	U' → +VU'			
V		V → id D						V → (E)																	V → entero	V → cadena	
D			D → λ					D → (L)	D → λ							D → λ			D → λ		D → λ	D → λ	D → λ	D → λ			

ANALIZADOR SEMÁNTICO

DISEÑO DE LA TRADUCCIÓN DIRIGIDA POR LA SINTAXIS

P' → {TSG:= creaTS(), TSactual:=TSG, DespG:= 0} P {DestruyeTS (TSG)}

P → BP {}

P → FP {}

P → λ {}

B → let {ZonaDecl:= true} T id; {ZonaDecl:= false,

InsertaTipoTS (id.pos, T.tipo),

If (TSL = NULL) Then

{InsertaDespTS (id.pos, DespG) DespG := DespG + T.ancho}

Else

{InsertaDespTS (id.pos, DespL) DespL := DespL + T.ancho},

B.tipo := tipo_ok,

B.tipoRet := vacío,

}

B → if (E) G {if (E.tipo != lógico) Then

{ B.tipo := tipo_error

B.tipoRet : = vacio }

Else

{ B.tipo := G.tipo

B.tipoRet := G.tipoRet }

}

B → S {B.tipo := S.tipo, B.tipoRet := S.tipoRet}

T → int {T.tipo := entero, T.ancho := 1}

T → string {T.tipo := cadena, T.ancho := 64}

T → boolean {T.tipo := lógico, T.ancho := 1}

G → S {G.tipo := S.tipo, G.tipoRet := S.tipoRet}

G → { C } O { If (C.tipo = O.tipo = tipo_ok) Then

{ G.tipo := tipo_ok }

Else

{ G.tipo := tipo_error }

If (C.tipoRet = O.tipoRet) Then

{ G.tipoRet := C.tipoRet }

Else If (O.tipoRet = vacío) Then

{ G.tipoRet := C.tipoRet }

Else If (C.tipoRet = vacío) Then

{ G.tipoRet := O.tipoRet }

Else

{ G.tipoRet := tipo_error }

C → $BC_{(1)}$ {If (B.tipo = C_1.tipo = tipo_ok) Then

{ C.tipo := tipo_ok }

Else

{ C.tipo := tipo_error }

If (O.tipoRet = C_1.tipoRet) Then

{ C.tipoRet := O.tipoRet }

Else If (C_1.tipoRet = vacío) Then

{ G.tipoRet := O.tipoRet }

Else If (O.tipoRet = vacío) Then

{ G.tipoRet := C_1.tipoRet }

Else

{ G.tipoRet := tipo_error }

C → λ {C.tipo := tipo_ok, C.tipoRet := vacio}

O → else { C } {O.tipo := C.tipo, O.tipoRet := C.tipoRet}

O → λ {O.tipo := tipo_ok, O.tipoRet := vacio}

S → id W { if (W.tipo = function) {

If (W.param = buscarParamTS(id.pos))Then

{ S.tipo := tipo_ok }

Else

{ S.tipo := tipo_error }

Else if (buscaTipoTS(id.pos) != null) Then

{ InsertaTipoTS (id.pos, entero)

If (TSL = NULL) Then

{InsertaDespTS (id.pos, DespG) DespG := DespG + 1}

Else

{InsertaDespTS (id.pos, DespL) DespL := DespL + 1},

S.tipo := tipo_ok,

}

Else if (W.tipo = buscaTipoTS(id.pos)) Then

{ S.tipo := tipo_ok }

Else

{ S.tipo := tipo_error }

}

S.tipoRet := vacío }

S → print (E); {S.tipo := if (E.tipo ∈ {entero, cadena})

Then tipo_ok

Else tipo_error,

S.tipoRet := vacío}

S → input (id); { if (BuscaTipoTS (id.pos) ∈ {entero, cadena})) Then

{ S.tipo := tipo_ok }

Else if (buscaTipoTS(id.pos) = lógico) Then

{ S.tipo := tipo_error }

Else

{ InsertaTipoTS (id.pos, entero),

If (TSL = NULL) Then

{InsertaDespTS (id.pos, DespG) DespG := DespG + 1}

Else

{InsertaDespTS (id.pos, DespL) DespL := DespL + 1},

S.tipo := tipo_ok,

S.tipoRet := vacío}

S → return X; {S.tipo := if (X.tipo != tipo_error)

Then tipo_ok

Else tipo_error,

S.tipoRet := X.tipo}

W → -=E; {W.tipo := E.tipo, W.param := vacío}

W → =E; {W.tipo := E.tipo, W.param := vacío}

W → (L); {W.tipo := function, W.param := L.tipo}

X → E {X.tipo := E.tipo}

X → λ {X.tipo := vacío}

L → EQ {If (Q.tipo = vacío)

Then L.tipo := E.tipo

Else L.tipo := E.tipo x Q.tipo}

L → λ {L.tipo := vacío}

Q → , $EQ_{(1)}$ {If (Q_1.tipo = vacío)

Then Q.tipo := E.tipo

Else Q.tipo := E.tipo x Q_1.tipo}

Q → λ {Q.tipo := vacío}

F → function id {TSL := CreaTS (), TSactual := TSL, DespL := 0, InsertaEtTS (id.pos, nuevaEt())}

H {InsertaTipoRet (id.pos, H.tipo), ZonaDecl : = True}

(A) { ZonaDecl : = False, InsertaTipoParam (id.pos, A.param) }

{ C }

{ If (C.tipoRet != H.tipo) Then Error

If (C.tipo = tipo_error) Then Error

DestruyeTS (TSL)

TSactual := TSG }

H → T {H.tipo := T.tipo}

H → λ {H.tipo := vacío}

A → T id K {InsertaTipoTS (id.pos, T.tipo)

InsertaDespTS (id.pos, DespL)

DespL := DespL + T.ancho

If (k.param = vacio) Then

{ A.param := T.tipo }

Else

{ A.param := T.tipo x K.param }

A.tipo := K.tipo

}

A → λ {A.tipo := tipo_ok, A.param := vacio}

K → , T id $K_{(1)}$ { InsertaTipoTS (id.pos, T.tipo)

InsertaDespTS (id.pos, DespL)

DespL := DespL + T.ancho

If (k_1.param = vacio) Then

{ K.param := T.tipo }

Else

{ K.param := T.tipo x K_1.param }

K.tipo := K_1.tipo

}

K → λ {K.tipo := tipo_ok, K.param := vacio}

E → RE' { E.tipo := If (R.tipo = E'.tipo = lógico)

Then lógico

Else If (E'.tipo = vacío)

Then R.tipo

Else tipo_error}

E' → &&RE'$_{(1)}$ {E'.tipo := If (R.tipo = lógico and E_1'.tipo != tipo_error) Then lógico Else tipo_error}

E' → λ {E'.tipo := vacío}

R → UR' {If (R'.tipo = lógico)

Then R.tipo := lógico

Else If (R'.tipo = vacío)

Then R.tipo := U.tipo

Else tipo_error}

R' → <UR'$_{(1)}$ {R'.tipo := If (U.tipo != tipo_error and R_1'.tipo != tipo_error) Then lógico Else tipo_error}

R' → >UR'$_{(1)}$ {R'.tipo := If (U.tipo != tipo_error and R_1'.tipo != tipo_error) Then lógico Else tipo_error}

R' → λ {R'.tipo := vacío}

U → VU' {If (U'.tipo = entero)

Then U.tipo := entero

Else If (U'.tipo = vacío)

Then U.tipo := V.tipo

Else tipo_error}

U' → +VU'$_{(1)}$ {U'.tipo := If (V.tipo = entero and U_1'.tipo = entero || U_1'.tipo = vacio) Then entero Else tipo_error}

U' → -VU'$_{(1)}$ {U'.tipo := If (V.tipo = entero and U_1'.tipo = entero || U_1'.tipo = vacio) Then entero Else tipo_error}

U' → λ {U'.tipo := vacío}

V → id D { if (D.tipo = function) {

If (D.param = buscarParamTS(id.pos)) Then

{ V.tipo := buscaTipoRet(id.pos)

V.ancho := anchoTipo(V.tipo) }

Else

{ V.tipo := tipo_error

V.ancho := 0 }

Else if (buscaTipoTS != null) Then

{ InsertaTipoTS (id.pos, entero),

If (TSL = NULL) Then

{InsertaDespTS (id.pos, DespG) DespG := DespG + 1}

Else

{InsertaDespTS (id.pos, DespL) DespL := DespL + 1},

V.tipo := entero

V.ancho : = 1}

Else

{ V.tipo := buscaTipoTS(id.pos)

V.ancho := anchoTipo(V.tipo) }

}

V → (E) {V.tipo := E.tipo, V.ancho := 0}

V → entero {V.tipo := entero, V.ancho := 1}

V → cadena {V.tipo := cadena, V.ancho := 64}

D → (L) {D.tipo := function, D.param := L.tipo}

D → λ {D.tipo := vacío, D.param := vacío}

TABLA DE SÍMBOLOS

DISEÑO TABLA DE SÍMBOLOS

En nuestro programa vamos a seguir la siguiente estructura para rellenar la tabla de símbolos:

Lexema	Tipo	Despl	numParam	TipoParamXX	TipoRetorno	EtiqFuncion

Siendo cada columna:

- Lexema: nombre identificador
- Tipo: representa el tipo del identificador.
- Despl: valor numérico que representa la dirección relativa que tendrá cada variable (el desplazamiento).
- numParam: valor numérico que representa el número de parámetros formales que tiene un identificador de tipo subprograma.
- TipoParamXX: representa el tipo del XXº parámetro de un subprograma. XX representa un número de hasta dos dígitos, cuyos valores irán desde el 1 hasta el valor del atributo numParam.
- TipoRetorno: representa el tipo que devuelve un identificador de tipo función.
- EtiqFuncion: representa la etiqueta que se asocia a un identificador de tipo función.

En nuestra tabla no añadimos ModoParamXX y Param debido a que vamos a realizar nuestro compilador en java y vamos a utilizar siempre el paso por valor.

Esta tabla la vamos a rellenar siguiendo el formato pedido, se pasará a un fichero externo (TS.txt) para poder ver todos los cambios mientras se va almacenando y destruyendo en memoria. También tendremos en cuenta las posibles anidaciones.

La tabla de símbolos funciona de la siguiente manera:

1. Al generar un identificador (Transición 1-7), se comprueba si ya está en la tabla.
2. En caso de que no estuviera, se añade una nueva entrada rellenando los datos de está con la información disponible.
3. Si se lee una nueva función se genera una nueva tabla de símbolos especifica. Esta tabla “hija” será destruida al terminar la función.

ANEXO

PRUEBA 1 (correcta):

```
let int n1;let int n2;
let boolean l1;let boolean l2;
let
        string
                cad
                        ;
input (n1);
l1 = l2;
if (l1&& l2) cad = 'hello';
n2 -= n1 - 378;

print(    33
                +
                n1
                +
                n2);

function ff boolean(boolean ss)
{
        varglobal = 8;
        if (l1) l2 = ff (ss);
        return ss;
}



if (ff(l1))
```

```
print (varglobal);
```

TOKENS:

<let,>

<int,>

<id,1>

<puntComa,>

<let,>

<int,>

<id,2>

<puntComa,>

<let,>

<int,>

<id,3>

<puntComa,>

<print,>

<pAbierto,>

<cadena,Introduce el primer operando>

<pCerrado,>

<puntComa,>

<input,>

<pAbierto,>

<id,1>

<pCerrado,>

<puntComa,>

<print,>

<pAbierto,>

<cadena,Introduce el segundo operando>

<pCerrado,>

<puntComa,>

<input,>

<pAbierto,>

<id,2>

<pCerrado,>

<puntComa,>

<function,>

<id,4>

<int,>

<pAbierto,>

<int,>

<id,1>

<coma,>

<int,>

<id,2>

<pCerrado,>

<kAbierta,>

<let,>

<int,>

<id,3>

<puntComa,>

<id,3>

<asig,>

<numEnt,8888>

<resta,>

<id,1>

<resta,>

<id,2>

<puntComa,>

<return,>

<id,3>

<puntComa,>

<kCerrada,>

<id,3>

<asig,>

<numEnt,0>

<puntComa,>

<print,>

<pAbierto,>

<id,4>

<pAbierto,>

<id,2>

<coma,>

<id,1>

<pCerrado,>

<pCerrado,>

<puntComa,>

TABLA DE SÍMBOLOS:

#1:

*'a'

+tipo:'entero'

+despl:0

*'b'

+tipo:'entero'

+despl:1

*'number'

+tipo:'entero'

+despl:2

*'operacion'

+tipo:'function'

+tipoRetorno:'entero'

+etiqFuncion:'Et1_operacion'

+numParam:2

+tipoParam1:'entero'

+tipoParam2:'entero'

#2:

*'num1_'

+tipo:'entero'

+despl:0

*'num2_'

+tipo:'entero'

+despl:1

*'number'

+tipo:'entero'

+despl:2

PARSE:

Descendente 1 4 7 1 4 7 1 4 7 1 6 17 36 39 43 50 46 42 38 1 6 18 1 6 17 36 39 43 50 46 42 38 1 6 18 2 29 30 7 32 7 34 7 35 12 4 7 12 6 16 21 36 39 43 49 45 47 52 45 47 52 46 42 38 12 6 19 23 36 39 43 47 52 46 42 38 13 1 6 16 21 36 39 43 49 46 42 38 1 6 17 36 39 43 47 51 25 36 39 43 47 52 46 42 38 27 36 39 43 47 52 46 42 38 28 46 42 38 3

ÁRBOL VAST:

- P (1)
    - B (4)
        - let
        - T (7)
            - int
        - id
        - ;
    - P (1)
        - B (4)
            - let
            - T (7)
                - int
            - id
            - ;
        - P (1)
            - B (4)
                - let
                - T (9)
                    - boolean
                - id

        - ;
    - P (1)
        - B (4)
            - let
            - T (9)
                - boolean
            - id
            - ;
        - P (1)
            - B (4)
                - let
                - T (8)
                    - string
                - id
                - ;
            - P (1)
                - B (6)
                    - S (18)
                        - input
                        - (
                        - id
                        -)
                        - ;
                - P (1)
                    - B (6)
                        - S (16)
- id

- W (21)
    - =
    - E (36)
        - R (39)
            - U (43)
                - V (47)
                    - id
                    - D (52)
                        - lambda
                - U_ (46)
                    - lambda
            - R_ (42)
                - lambda
        - E_ (38)
            - lambda
    - ;
                                - P (1)
                                    - B (5)
- if
- (
- E (36)
    - R (39)
        - U (43)
            - V (47)
                - id
                - D (52)
                    - lambda

            - U_ (46)
                - lambda
        - R_ (42)
            - lambda
    - E_ (37)
        - &&
        - R (39)
            - U (43)
                - V (47)
                    - id
                    - D (52)
                        - lambda
                - U_ (46)
                    - lambda
            - R_ (42)
                - lambda
        - E_ (38)
            - lambda
-)
- G (10)
    - S (16)
        - id
        - W (21)
            - =
            - E (36)
                - R (39)
                    - U (43)

- V (50)
  - cadena
- U_ (46)
  - lambda
- R_ (42)
  - lambda
- E_ (38)
  - lambda
- ;
- P (1)

- B (6)
  - S (16)
    - id
    - W (20)
      - -=
      - E (36)
        - R (39)
          - U (43)
            - V (47)
              - id
              - D (52)
                - lambda
            - U_ (45)
              - -
              - V (49)
                - entero
              - U_ (46)

- lambda
- R_ (42)
- lambda
- E_ (38)
- lambda
- ;

- P (1)
  - B (6)
    - S (17)
      - print
      - (
      - E (36)
        - R (39)
          - U (43)
            - V (49)
              - entero
            - U_ (44)
              - +
              - V (47)
                - id
                - D (52)

- lambda
  - U_ (44)
    - +
    - V (47)

- id
- D (52)

- lambda
  - U_ (46)
- lambda
  - R_ (42)
    - lambda
  - E_ (38)
    - lambda
  -)
  - ;
- P (2)
  - F (29)
    - function
    - id
    - H (30)
      - T (9)
        - boolean
    - (
    - A (32)
      - T (9)
        - boolean
      - id
      - K (35)
        - lambda
    -)
    - {
    - C (12)
      - B (6)

- S (16)
  - id
  - W (21)
    - =
    - E (36)
      - R (39)

- U (43)
  - V (49)
    - entero
  - U_ (46)
    - lambda
- R_ (42)
  - lambda
    - E_ (38)
- lambda
    - ;
- C (12)
  - B (5)
    - if
    - (
    - E (36)
      - R (39)
        - U (43)
- V (47)
  - id
  - D (52)
    - lambda

- U_ (46)
    - lambda
        - R_ (42)
- lambda
        - E_ (38)
            - lambda
    -)
    - G (10)
        - S (16)
            - id
            - W (21)
- =
- E (36)
    - R (39)
        - U (43)
            - V (47)
                - id
                - D (51)
                    - (
                    - L (25)
                        - E (36)
                            - R (39)
                                - U (43)
- V (47)
    - id
    - D (52)
        - lambda

- U_ (46)
    - lambda
        - R_ (42)
- lambda
        - E_ (38)
            - lambda
        - Q (28)
            - lambda
        -)
    - U_ (46)
        - lambda
    - R_ (42)
        - lambda
    - E_ (38)
        - lambda
- ;
        - C (12)
            - B (6)
                - S (19)
                    - return
                    - X (23)
- E (36)
    - R (39)
        - U (43)
            - V (47)
                - id
                - D (52)

                                        - lambda
                                - U_ (46)
                                    - lambda
                            - R_ (42)
                                - lambda
                    - E_ (38)
                        - lambda
                                                    - ;
                                            - C (13)
                                                - lambda
                        - }
                    - P (1)
                        - B (5)
                            - if
                            - (
                            - E (36)
                                - R (39)
                                    - U (43)
                                        - V (47)
                                            - id
                                            - D (51)
- (
- L (25)
    - E (36)
        - R (39)
            - U (43)
                - V (47)

                    - id
                    - D (52)
                        - lambda
                - U_ (46)
                    - lambda
            - R_ (42)
                - lambda
        - E_ (38)
            - lambda
    - Q (28)
        - lambda
-)
                    - U_ (46)
                        - lambda
                - R_ (42)
                    - lambda
            - E_ (38)
                - lambda
        -)
        - G (10)
            - S (17)
                - print
                - (
                - E (36)
                    - R (39)
                        - U (43)
- V (47)

- id
- D (52)
  - lambda

- U_ (46)
  - lambda
    - R_ (42)

- lambda
  - E_ (38)
    - lambda
  -)
  - ;
  - P (3)
    - lambda

PRUEBA 2 (correcta):

```
let int a;
let   int  b;
let boolean bbb;
a = 3;
b=a
                                                    ;
let boolean c;


c = a > b;


if (c) b -= 3333;
```

```
a = a + b;
print (a) ;
print(b);
```

PRUEBA 3 (correcta):

```
        let     boolean         b;let int x;
input (x);
print (x);
input (z);
print (x-z);
b=x>z;if (b)
x =
 x + 6
  + z
  + 1
  + (2
  - y
  - 7);
```

PRUEBA 4 (correcta):

```
let string texto;
function pideTexto ()
{
```

```
        print ('Introduce un texto');
        input (texto);
}
function alert (string msg)
{
        print ('Texto introducido:');
        print (msg);
}
pideTexto();
 alert
        (texto);
```

PRUEBA 5 (correcta):

```
let int   a   ;
let int    b  ;
let int number;
print ( 'Introduce el primer operando' );
input (a);
print ('Introduce el segundo operando');input(b);
function operacion
 int (int num1_, int num2_)
{
        let int number;
        number = 8888 - num1_-num2_;
        return number;
}
number = 0;
print(operacion(b,a));
```

PRUEBA 6 (incorrecta):

```
let int   a   ;

let int    b  ;

let int number;

print ( 'Introduce el primer operandoIntroduce el primer operandoIntroduce el primer
operandoIntroduce el primer operando' );

input (a);

print ('Introduce el segundo operando');input(b);

function  operacion       int(int num_1, int num_2)

{

          return  num_1 + num_2-77777;

}


number = operacion (a, b);

print (number);
```

SALIDA:

Error Léxico Linea 4: la cadena ha excedido el maximo de caracteres.

Error sintactico en la linea : 5 se encuentra el token input cuando deberia aparecer el token pCerrado

Error Lexico Linea 9: el valor supera el entero maximo del lenguaje

PRUEBA 7 (incorrecta):

```
let int z;

let boolean boolean_1;

let int x;

let string ss;

let int xx;

let boolean boolean_2;
```

```
function f1 int(int f1, boolean b1)
{
                print(ss);
                x = xx+f1;
                boolean_1 = boolean_1&& boolean_2;
                return (01234);
}


function f2 boolean( int f2 , boolean b1 )
{
                input (y);
                print ((4+5+77+(088-f2)));
                return (boolean_1&&boolean_2&&b1);
}

x =
  x + 6
    - z
    + 1
    + (2
    + y
    - 6)
          ;


print f1 (x, f2 (3, boolean_2)));
```

SALIDA:

Error sintactico en la linea : 35 se encuentra el token id cuando deberia aparecer el token pAbierto

PRUEBA 8 (incorrecta):

```
let int z;
let boolean boolean_1;
let int x;
let string ss;
let int xx;
let boolean boolean_2;

function f1 int(int f1, boolean b1)
{
		input (z);
		boolean_1 = boolean_1 + boolean_2;
		xx -= f1 && x;
		print//Alerta
		(ss);
		return ;
}



function f2 boolean( int f2 , boolean b1 )
{
		input (y);
		print ((4+5+77+(088-f2)));
		return ;
```

```
}


x =
 x + 6
  - z
  < 1
  + (2
  + y
  + 6)
        ;



boolean_2=(f2 (f1 (x, boolean_1), boolean_2));
```

SALIDA:

Error Semantico en la linea 11, la expresion no es correcta

Error Semantico en la linea 11, la asignacion no es correcta

Error Semantico en la linea 12, la expresion no es correcta

Error Semantico en la linea 12, la asignacion no es correcta

Error Semantico en el retorno de la funcion de la linea 8

Error Semantico en el cuerpo de la funcion de la linea 8

Error Semantico en el retorno de la funcion de la linea 20

Error Semantico en la linea 27, la asignacion no es correcta

PRUEBA 9 (incorrecta):

```
let int   a   ;
let boolean    b  ;
let int number;
print ( 'Introduce el primer operando' );
```

```
input (a);
print ('Introduce el segundo operando');input(b);
function  operacion       int(int num_1, int num_2)
{
        return  num_1 + num_2-77;
}

number = operacion (a, b);
print (b);
```

SALIDA:

Error Semantico en la linea 6, no se puede realizar el input de una variable logica

Error Semantico en la linea 12, los parametros de esta funcion no son correctos

Error Semantico en la linea 12, la expresion no es correcta

Error Semantico en la linea 12, la asignacion no es correcta

Error Semantico en la linea 13, no se ha podido realizar el print

PRUEBA 10 (incorrecta):

```
let int   a   ;
let boolean    b  ;
let int number;
print ( 'Introduce el primer operando' );
input (a);
print ('Introduce el segundo operando');input(b);
function operacion
 int (int num1_, int num2_)
{
        let int number;
        number = 88 && num1_-num2_;
```

```
        return b;
}
number = 0;
print(operacion(b,a));
```

SALIDA:

Error Semantico en la linea 6, no se puede realizar el input de una variable logica

Error Semantico en la linea 11, la expresion no es correcta

Error Semantico en la linea 11, la asignacion no es correcta

Error Semantico en el retorno de la funcion de la linea 7

Error Semantico en el cuerpo de la funcion de la linea 7

Error Semantico en la linea 15, los parametros de esta funcion no son correctos

Error Semantico en la linea 15, la expresion no es correcta

Error Semantico en la linea 15, no se ha podido realizar el print