Procesadores de Lenguajes

Descendente Recursivo

Este documento contiene una breve descripción del Analizador Léxico, Sintáctico, Semántico y Tabla de símbolos del lenguaje JavaScript-PL.

Autores-Grupo70

Angélica María Guamán Albarracín, t110212

Robinson Fernando Ganchala Benitez, t110377

Índice

Introducción 3
Analizador Léxico 4
Identificar los tokens 4
Construir la gramática para JavaScript-PL 5
Construir Autómata Finito Determinista (AFD) 6
Acciones Semánticas 6
Analizador Sintáctico 8
Elementos de la Gramática 8
Símbolos Terminales 8
Símbolos No Terminales 8
Gramática 8
Analizador Semántico 11
Traducción Dirigida por la Sintaxis 11
Tabla de Símbolos 13
Casos de Pruebas Correctos 14
Prueba 1 14
Código Fuente 14
Tokens 14
Tabla de Símbolos 15
Parser 15
Árbol 16
Prueba 2 16
Código Fuente 16
Tokens 17
Tabla de Símbolos 17
Parser 18
Árbol 18
Prueba 3 19
Código Fuente 19
Tokens 19
Tabla de Símbolos 20
Parser 20
Árbol 21
Prueba 4 21
Código Fuente 21

Tokens 22
Tabla de Símbolos 22
Parser 23
Árbol 23
Prueba 5 24
Código Fuente 24
Tokens 24
Tabla de Símbolos 25
Parser 25
Árbol 26
Casos de Pruebas Incorrectos 27
Prueba 1 27
Código Fuente 27
Errores 27
Prueba 2 27
Código Fuente 27
Errores 28
Prueba 3 28
Código Fuente 28
Errores 29
Prueba 4 29
Código Fuente 29
Errores 29
Prueba 5 30
Código Fuente 30
Errores 30

Introducción

Se plantea realizar el diseño e implementación de un procesador de lenguajes que realiza un análisis léxico, sintáctico y semántico del lenguaje JavaScript-PL, incluyendo la tabla de símbolos local y global.

Para el desarrollo de este analizador nos apoyamos de la herramienta JavaCC que es un generador de analizadores sintácticos descendente. El código fuente está escrito en el lenguaje de Java, la entrada de este programa es un código fuente JavaScript(.js) y si el análisis es correcto genera 3 ficheros(tokens.txt, parser.txt y TablaDeSimbolos.txt) en el mismo directorio del código. Si este análisis no es correcto se muestran los errores léxico y sintácticos en la consola, y los errores semánticos en el fichero log.txt.

Para comprobar que el análisis es correcto construimos el árbol sintáctico con el parser, para esto nos apoyamos de la herramienta proporcionada por la asignatura(VASt).

Para la compilación de nuestra práctica se debe añadir lo siguiente:

- Para que funcione JavaCC se debe incluir al PATH la siguiente ruta del disco PROYECTO-PDL/`javacc/bin`
- Desde el cmd se ejecuta lo siguiente: `java LexicoSintactico < fichero.js > log.txt`, y esto genera todos los ficheros(tokens.txt, TablaDeSimbolos.txt y parser.tx)

Analizador Léxico

Un analizador léxico o analizador lexicográfico es la primera fase de un compilador, consiste en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos.

Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada del analizador sintáctico.

Las funciones que tiene el Analizador Léxico son:

1. Tiene acceso al Programa Fuente.
2. Leer el Programa Fuente desde el principio hasta el final y nunca retrocede.
3. Formar una palabra válida (tokens), si es así se lo envía al Analizador Sintáctico, sino al Gestor de Errores
4. Elimina todos los caracteres que no valen para el ejecutable como comentarios, espacios en blanco, etc.
5. Informa al Gestor de Errores por la línea que va.

Pasos para el diseño de un Analizador Léxico:

1. Identificar los tokens.
2. Construir la gramática para JavaScript-PL.
3. Construir el AFD según el paso anterior.
4. Completar el AFD con las acciones semánticas.
5. Incluir los mensajes de error.

Identificar los tokens

Un token o también llamado componente léxico es una cadena de caracteres que tiene un significado propio en cierto lenguaje de programación.

Se nos ha asignado los siguientes tokens:

- Palabras Reservadas: (PR, posTS)

posTS	Palabra Reservada
1	var
2	function
3	prompt
4	write
5	while
6	if
7	return
8	int

9	chars
10	bool
11	true
12	false

- Entero → <Numero, valor>
- Cadena → <Cadena, posMem>
- Operador asignación → <Asignacion, =>
- Operador suma → <Suma, +>
- Operador lógico and → <And, &&>
- Operador relacional → <MayorQue, >>
- Símbolo coma → <Coma, ,>
- Símbolo llave abierta → <LlaveAbierta, {>
- Símbolo llave cerrada → <LlaveCerrada, }>
- Símbolo paréntesis abierto → <ParentesisAbierto, (>
- Símbolo paréntesis cerrado → <ParentesisCerrado,)>
- Símbolo post-decremento → <PostDecremento, -->
- Identificadores → <id, posTS>

Construir la gramática para JavaScript-PL

S → lA | dD| + | (|) | { | } | > | , | &G | -H| = | “E | del S

A → lA | dA | _A | λ

D → dD | λ

E → c1F

F → E

F → “

G → &

H → -

Donde:

c1 = cualquier caracter

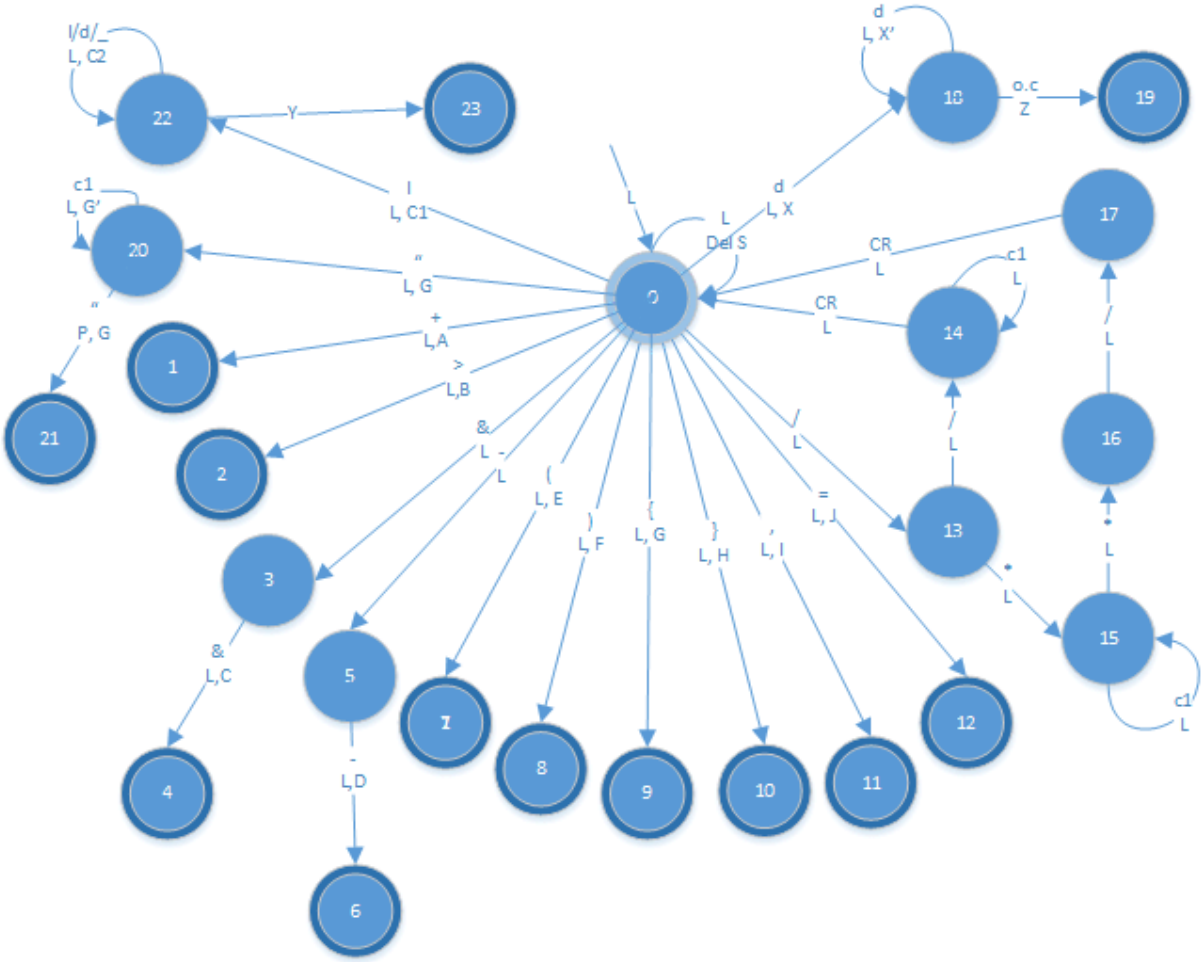
Numero → $valor \in [0 - 32767]$

$Identificadores = l\{l / d / \_\}^{*}$

$l = letras\_del\_Alfabeto$

$d = digito[0...9]$

Construir Autómata Finito Determinista (AFD)



Acciones Semánticas

cad = “”;

num=0;

L= Leer

A = GenToken(suma,)

B = GenToken(mayorQue,)

C = GenToken(and,)

C1 = concatenar

cad += l

C2 = concatenar

cad += {l/d/_}

D = GenToken(postDecremento,)

E = GenToken(parentesisAbierto,)

F = GenToken(parentesisCerrado,)

G = GenToken(LLaveAbierta,)

H = GenToken(LLaveCerrada,)

I = GenToken(Coma,)

J = GenToken(Asignacion,)

K = concatenar

cad += “;

K’ = concatenar

cad += c1

P = posMem:=buscaTS(cad)

GenToken(cadena,posMem)

M = GenToken(eof,)

X = num := d

X’ = num := num * 10 + d

Y = pos := buscaTS(cad);

If(pos <= 0) then

pos := guardarTS(cad)

GenToken(id, pos)

else

GenToken(pr,pos)

Z = if(num<= 32767) then

GenToken(entero,num)

Analizador Sintáctico

Un analizador sintáctico es una de las partes del compilador que transforma su entrada en un árbol de derivación, que es un parser para posteriormente construir el árbol según la entrada.

Un parser es una secuencia de números que identifican las reglas de la gramática para el analizador sintáctico.

Es el proceso de comprobar si una determinada secuencia de tokens pertenece al lenguaje generado por la gramática, si no somos capaces de construir el árbol es porque la gramática es incorrecta.

Elementos de la Gramática

A continuación se muestra la gramática para el Analizador Descendente Recursivo, junto con los símbolos terminales, no terminales y producciones.

Símbolos Terminales

```
Terminales = { var id prompt write return if while function eof salto ( )
{ }  =  + && > -- , num cadena true false int chars bool }
```

Símbolos No Terminales

```
NoTerminales = { PP P B F A H HH W T TP E EP U UP V VP L Q S SP X C R }
```

Gramática

```
Producciones = {

PP -> P ////1

P -> B P ////2

P -> F P ////3

P -> eof ////4


F -> function HH id ( A ) { C } ////5


HH -> int ////6

HH -> chars ////7

HH -> bool ////8

HH -> lambda ////9
```

```
A -> H id R ////10
A -> lambda ////11

R -> , H id R ////12
R -> lambda ////13

C -> B C ////14
C -> lambda ////15

B -> var H id W ////16
B -> if ( E ) S ////17
B -> while ( E ) { C } ////18
B -> S ////19

S -> prompt ( id ) ////20
S -> write ( E ) ////21
S -> return X  ////22
S -> id SP ////23
SP -> ( L ) ////24
SP -> = E ////25
SP -> -- ////26

X -> E ////27
X -> lambda ////28

H -> int ////29
H -> chars ////30
H -> bool ////31

W -> = E ////32
W -> lambda ////33
```

```
E -> T EP ////34
EP -> && T EP ////35
EP -> lambda ////36
T -> U TP ////37
TP -> > U TP ////38
TP -> lambda ////39
U -> V UP ////40
UP -> + V UP ////41
UP -> lambda ////42
V -> ( E ) ////43
V -> num ////44
V -> cadena ////45
V -> true ////46
V -> false ////47
V -> id VP ////48
VP -> ( L ) ////49
VP -> -- ////50
VP -> = V ////51
VP -> lambda ////52

L -> E Q ////53
L -> lambda ////54

Q -> , E Q ////55
Q -> lambda ////56
}
```

Analizador Semántico

La fase de análisis semántico revisa el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En ella se utiliza la estructura jerárquica determinada por la fase de análisis sintáctico para identificar los operadores y operandos de expresiones.

Un componente importante del análisis semántico es la verificación de tipos. Aquí el compilador verifica si cada operador tiene operandos permitidos por la especificación del lenguaje fuente.

Traducción Dirigida por la Sintaxis

```
PP -> P {TSG=crearTS()}

P -> B P1 {if(B.tipo=P1.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR }

P -> F P2 {if(F.tipo=P2.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR }

P -> eof {if(P.tipo=EOF) then TIPO_OK else TIPO_ERROR }

F -> function HH id ( A ) { C } {TSL=crearTSL();esLocal=true, insertarIdTS=(TSG,id.entrada,HH.tipo),
if(C.tipo=HH.tipo) then TIPO_OK}

HH -> int       {HH.tipo=int, HH.tam=2}

HH -> chars     {HH.tipo=chars, HH.tam=1}

HH -> bool      {HH.tipo=bool, HH.tam=1}

HH -> lambda    {HH.tipo=lambda}

A -> H id R     {if(H.tipo=TIPO_OK)  then  añadirTSL(TSL,id.entrada,H.tipo),  if(R.tipo=TIPO_OK)  then
TIPO_OK else TIPO_ERROR }

A -> lambda      {A.tipo=lambda}

R -> , H id R   {if(H.tipo=TIPO_OK)  then  añadirTSL(TSL,id.entrada,H.tipo),  if(R.tipo=TIPO_OK)  then
TIPO_OK else TIPO_ERROR }

R -> lambda     {R.tipo=lambda}

C -> B C        {if(B.tipo=C.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR}

C -> lambda     {C.tipo=lambda}

B -> var H id W {if(H.tipo=TIPO_OK  &&  esLocal==true)  then  añadirTSL(TSL,id.entrada,H.tipo)  else
if(H.tipo=TIPO_OK)  then  añadirTSG(TSG,id.entrada,H.tipo);  if(W.tipo=TIPO_OK)then  TIPO_OK  else
TIPO_ERROR}

B -> if ( E ) S {B.tipo:=(if (E.tipo=bool) then S.tipo else TIPO_ERROR)}

B -> while ( E ) { C }  {B.tipo:=(if (E.tipo=TIPO_OK) then C.tipo else TIPO_ERROR)}

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

S -> prompt ( id )      {S.tipo:=(añadirTSG(TSG,id.entrada,TIPO_OK))}

S -> write ( E )        {S.tipo:=(if(E.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR)}

S -> return X   {S.tipo:=(if(X.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR)}

S -> id SP      {if(esLocal=true)then               añadirTSL(TSL,id.entrada,H.tipo)               else
añadirTSG(TSG,id.entrada,H.tipo)}

SP -> ( L )     {SP.tipo:=(if (L.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR)}

SP -> = E       {SP.tipo:=(if(E.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR)}
```

```
SP -> --

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

X -> lambda     {X.tipo=lambda}

H -> int{H.tipo=int}

H -> chars      {H.tipo=chars}

H -> bool       {H.tipo=bool}

W -> = E        {W.tipo:=(if(E.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR)}

W -> lambda     {W.tipo=lambda}

E -> T EP       {E.tipo:=(if(T.tipo=EP.tipo=TIPO_OK)) then TIPO_OK else TIPO_ERROR}

EP -> && T EP   {EP.tipo:=(if(T.tipo=EP1.tipo=TIPO_OK)) then TIPO_OK else TIPO_ERROR}

EP -> lambda    {EP.tipo=lambda}

T -> U TP       {T.tipo:=(if(U.tipo=TP.tipo=TIPO_OK)) then TIPO_OK else TIPO_ERROR}

TP -> > U TP    {TP.tipo:=(if(U.tipo=TP1.tipo=TIPO_OK)) then TIPO_OK else TIPO_ERROR}

TP -> lambda    {TP.tipo=lambda}

U -> V UP       {U.tipo:=(if(V.tipo=UP.tipo=TIPO_OK)) then TIPO_OK else TIPO_ERROR}

UP -> + V UP    {UP.tipo:=(if(V.tipo=UP1.tipo=TIPO_OK)) then TIPO_OK else TIPO_ERROR}

UP -> lambda    {Up.tipo=lambda}

V -> ( E )      {V.tipo:=(if (E.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR)}

V -> num        {V.tipo=int}

V -> cadena     {V.tipo=chars}

V -> true       {V.tipo=bool}

V -> false      {V.tipo=bool}

V -> id VP      {if(esLocal=true)then               añadirTSL(TSL,id.entrada,H.tipo)               else
añadirTSG(TSG,id.entrada,H.tipo)}

VP -> ( L )     {VP.tipo:=(if (L.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR}

VP -> --

VP -> = V       {VP.tipo:=(if(V.tipo=TIPO_OK) then TIPO_OK else TIPO_ERROR)}

VP -> lambda    {VP.tipo=lambda}

L -> E Q        {L.tipo:=(if(E.tipo=Q.tipo=TIPO_OK)) then TIPO_OK else TIPO_ERROR}

L -> lambda     {L.tipo=lambda}

Q -> , E Q      {Q.tipo:=(if(E.tipo=Q.tipo=TIPO_OK)) then TIPO_OK else TIPO_ERROR}

Q -> lambda     {Q.tipo=lambda}
```

Tabla de Símbolos

El diseño de la tabla de símbolos es el siguiente (FormatoTS.java):

```
///ESTA ESTRUCTURA ES DE LA TABLA DE SIMBOLOS
String lexema;
String tipo;
int nParam;
String kind; //var o function
ArrayList<String> tipoParams=new ArrayList<String>();


public FormatoTS(String kind, String lexema, String tipo, int nParam, ArrayList<String> tipoParams)
{
	this.kind=kind;
	this.lexema=lexema;
	this.tipo=tipo;
	this.nParam=nParam;
	this.tipoParams=tipoParams;
}
```

Para el desarrollo de la práctica utilizamos 2 listas de tipo FormatoTS, estas listas son para la tabla de símbolos global y local.

```
public static ArrayList<FormatoTS> tsg = new ArrayList<FormatoTS>();
public static ArrayList<FormatoTS> tsl = new ArrayList<FormatoTS>();
```

Casos de Pruebas Correctos

A continuación se muestran algunos ejemplos correctos

Prueba 1

Código Fuente

ejem.js

```
1  //PRIMERA PRUEBA BUENA
2  //Variables globales
3  var int a
4  var int b
5  var chars s = "APROBADO =)"
6  function int sumar(int num1, int num2)
7  {
8      while(true){
9          if(a>b)
10         return num1+num2
11     }
12     return 0
13 }
14
15 write(s)
```

Tokens

tokens.txt

```
1  <PalabraReservada,var>
2  <PalabraReservada,int>
3  <Identificador,a>
4  <PalabraReservada,var>
5  <PalabraReservada,int>
6  <Identificador,b>
7  <PalabraReservada,var>
8  <PalabraReservada,chars>
9  <Identificador,s>
10 <Asignacion,=>
11 <Cadena,"APROBADO =)">
12 <PalabraReservada,function>
13 <PalabraReservada,int>
14 <Identificador,sumar>
15 <ParentesisAbierto,(>
16 <PalabraReservada,int>
17 <Identificador,num1>
18 <Coma,,>
19 <PalabraReservada,int>
20 <Identificador,num2>
21 <ParentesisCerrado,)>
22 <LlaveAbierta,{>
```

tokens.txt

```
22 <LlaveAbierta,{>
23 <PalabraReservada,while>
24 <ParentesisAbierto,(>
25 <PalabraReservada,true>
26 <ParentesisCerrado,)>
27 <LlaveAbierta,{>
28 <PalabraReservada,if>
29 <ParentesisAbierto,(>
30 <Identificador,a>
31 <MayorQue,>>
32 <Identificador,b>
33 <ParentesisCerrado,)>
34 <PalabraReservada,return>
35 <Identificador,num1>
36 <Suma,+>
37 <Identificador,num2>
38 <LlaveCerrada,}>
39 <PalabraReservada,return>
40 <Numero,0>
41 <LlaveCerrada,}>
42 <PalabraReservada,write>
43 <ParentesisAbierto,(>
44 <Identificador,s>
45 <ParentesisCerrado,)>
```

Tabla de Símbolos

TablaDeSimbolos.txt

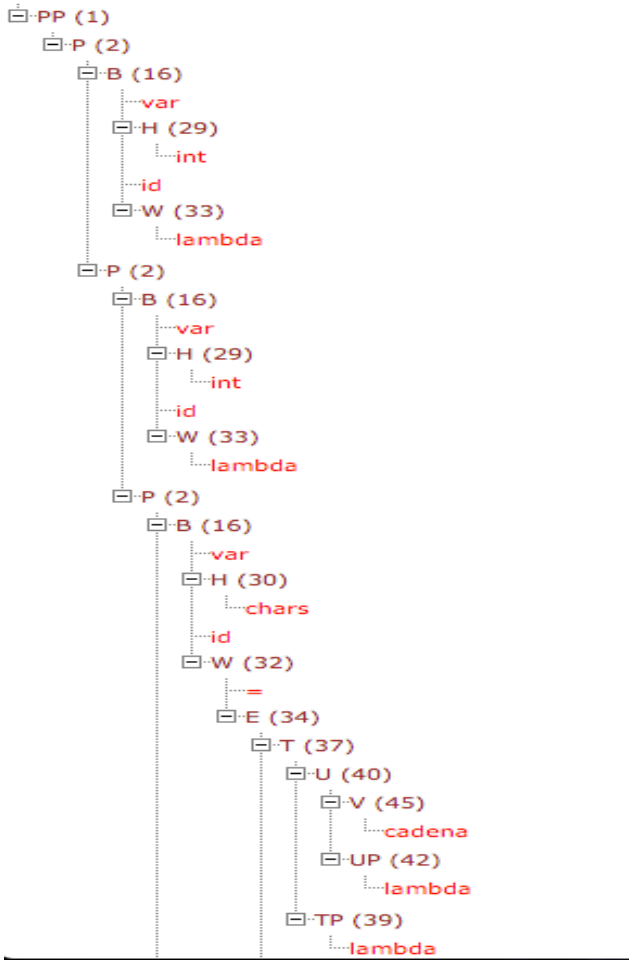
```
TABLA PRINCIPAL PALABRAS RESERVADAS #1:
* LEXEMA: 'var'
* LEXEMA: 'function'
* LEXEMA: 'prompt'
* LEXEMA: 'write'
* LEXEMA: 'while'
* LEXEMA: 'if'
* LEXEMA: 'return'
* LEXEMA: 'int'
* LEXEMA: 'chars'
* LEXEMA: 'bool'
* LEXEMA: 'true'
* LEXEMA: 'false'
------------------------------------------------------------
TABLA PRINCIPAL CON VARIABLES #2:
* LEXEMA: 'a'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #3:
* LEXEMA: 'b'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #4:
* LEXEMA: 's'
 +tipo:'chars'
TABLA PRINCIPAL #5:
* (Esto es una funcion) LEXEMA: 'sumar'
 ATRIBUTOS:
 +tipo:'int'
  +parametros:2
   +tipoParam1:'int'
    +tipoParam2:'int'
CONTENIDO DE LA FUNCION sumar #6:
* LEXEMA:'num1' (tipo de entrada 'parametro')
 +tipo:'int'
* LEXEMA:'num2' (tipo de entrada 'parametro')
 +tipo:'int'

```

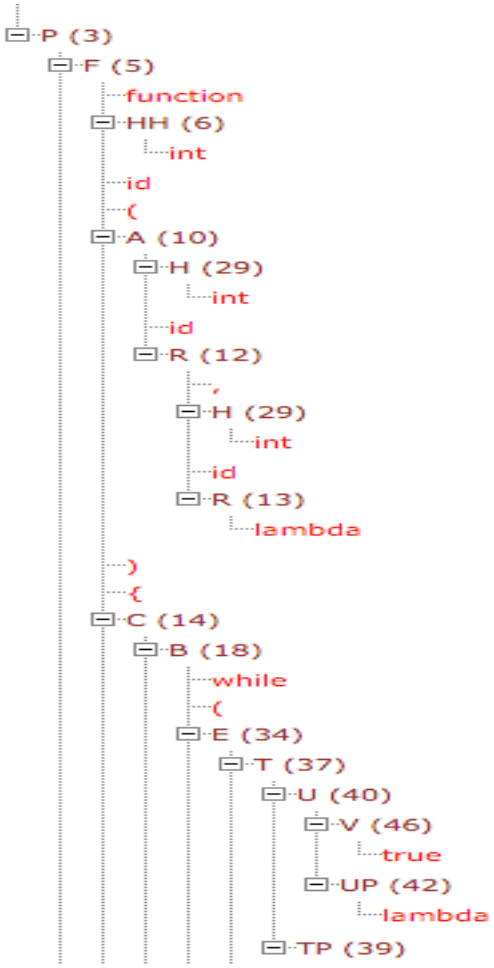
Parser

parser.txt

```
Descendente 1 2 16 29 33 2 16 29 33 2 16 30 32 34 37 40 45 42 39 36
3 5 6 10 29 12 29 13 14 18 34 37 40 46 42 39 36 14 17 34 37 40 48 52
42 38 40 48 52 42 39 36 22 27 34 37 40 48 52 41 48 52 42 39 36 15 14
19 22 27 34 37 40 44 42 39 36 15 2 19 21 34 37 40 48 52 42 39 36 4
```

Árbol





Prueba 2

Código Fuente

ejem.js

```
1   //PRIMERA PRUEBA BUENA
2   //Variables globales
3   var int a = 2
4   var int b = 3
5   var chars s = "APROBADO =)"
6   function bool verdad (int num1, int num2){
7
8       if(num1>num2) return true
9       prompt(num1)
10      prompt(num2)
11      return false
12  }
13
14  function int sumar()
15  {
16      //Variables locales
17      var bool c
18      c = verdad(a,b)
19      if(c)
20          write(s)
21      return 10
22  }
23
24  nota = sumar()
```

Tokens

tokens.txt

```
<PalabraReservada,var>
<PalabraReservada,int>
<Identificador,a>
<PalabraReservada,var>
<PalabraReservada,int>
<Identificador,b>
<PalabraReservada,var>
<PalabraReservada,chars>
<Identificador,s>
<Asignacion,=>
<Cadena,"APROBADO =)">
<PalabraReservada,function>
<PalabraReservada,int>
<Identificador,sumar>
<ParentesisAbierto,(>
<PalabraReservada,int>
<Identificador,num1>
<Coma,,>
<PalabraReservada,int>
<Identificador,num2>
<ParentesisCerrado,)>
<LlaveAbierta,{>
```

tokens.txt

```
<LlaveAbierta,{>
<PalabraReservada,while>
<ParentesisAbierto,(>
<PalabraReservada,true>
<ParentesisCerrado,)>
<LlaveAbierta,{>
<PalabraReservada,if>
<ParentesisAbierto,(>
<Identificador,a>
<MayorQue,>>
<Identificador,b>
<ParentesisCerrado,)>
<PalabraReservada,return>
<Identificador,num1>
<Suma,+>
<Identificador,num2>
<LlaveCerrada,}>
<PalabraReservada,return>
<Numero,0>
<LlaveCerrada,}>
<PalabraReservada,write>
<ParentesisAbierto,(>
<Identificador,s>
<ParentesisCerrado,)>
```

Tabla de Símbolos

TablaDeSimbolos.txt

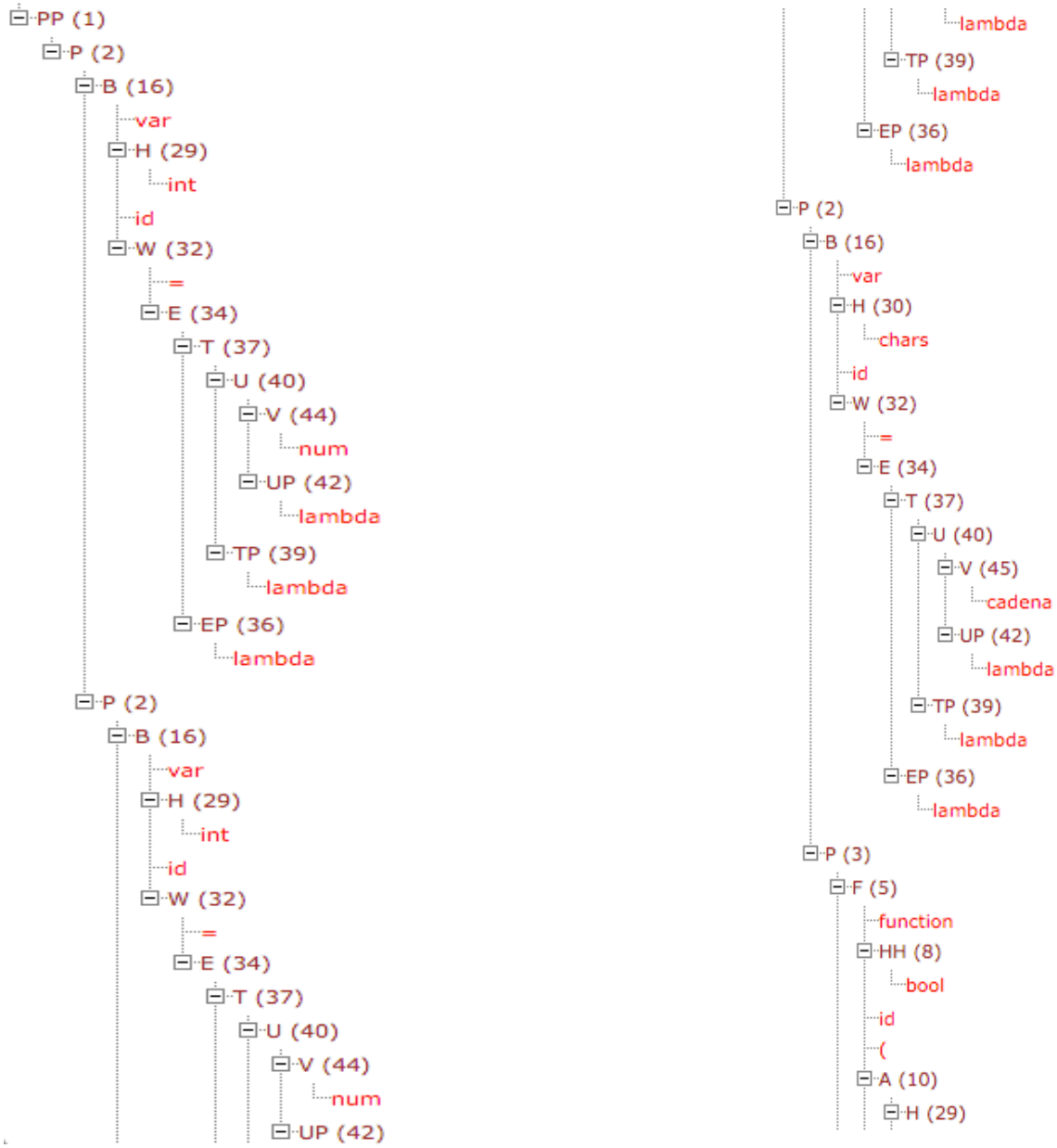
```
TABLA PRINCIPAL PALABRAS RESERVADAS #1:
* LEXEMA: 'var'
* LEXEMA: 'function'
* LEXEMA: 'prompt'
* LEXEMA: 'write'
* LEXEMA: 'while'
* LEXEMA: 'if'
* LEXEMA: 'return'
* LEXEMA: 'int'
* LEXEMA: 'chars'
* LEXEMA: 'bool'
* LEXEMA: 'true'
* LEXEMA: 'false'
------------------------------------------------------------
TABLA PRINCIPAL CON VARIABLES #2:
* LEXEMA: 'a'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #3:
* LEXEMA: 'b'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #4:
* LEXEMA: 's'
 +tipo:'chars'
TABLA PRINCIPAL #5:
* (Esto es una funcion) LEXEMA: 'verdad'
 ATRIBUTOS:
 +tipo:'bool'
  +parametros:2
   +tipoParam1:'int'
    +tipoParam2:'int'
CONTENIDO DE LA FUNCION verdad #6:
* LEXEMA:'num1' (tipo de entrada 'parametro')
 +tipo:'int'
* LEXEMA:'num2' (tipo de entrada 'parametro')
 +tipo:'int'
TABLA PRINCIPAL #7:
* (Esto es una funcion) LEXEMA: 'sumar'
 ATRIBUTOS:
 +tipo:'int'
  +parametros:0
CONTENIDO DE LA FUNCION sumar #8:
* LEXEMA: 'c' (tipo de entrada 'variable')
 +tipo:'bool'
TABLA PRINCIPAL CON VARIABLES #9:
* LEXEMA: 'nota'
 +tipo:'int'
```

Parser

parser.txt

```
Descendente 1 2 16 29 32 34 37 40 44 42 39 36 2 16 29 32 34 37 40 44
42 39 36 2 16 30 32 34 37 40 45 42 39 36 3 5 8 10 29 12 29 13 14 17
34 37 40 48 52 42 38 40 48 52 42 39 36 22 27 34 37 40 46 42 39 36 14
19 20 14 19 20 14 19 22 27 34 37 40 47 42 39 36 15 3 5 6 11 14 16 31
33 14 19 23 25 34 37 40 48 49 53 34 37 40 48 52 42 39 36 55 34 37 40
48 52 42 39 36 56 42 39 36 14 17 34 37 40 48 52 42 39 36 21 34 37 40
48 52 42 39 36 14 19 22 27 34 37 40 44 42 39 36 15 2 19 23 25 34 37
40 48 49 54 42 39 36 4
```

Árbol



Prueba 3

Código Fuente

ejem.js

```
1  //PRIMERA PRUEBA BUENA
2
3  function bool verdad (int num1, int num2){
4
5      if(num1>num2) return true
6
7      return false
8  }
9
10 function int sumar(int a, int b)
11 {
12     //Variables locales
13     var int c = a + b
14     verdad(z,x)
15     return c
16 }
17
18 sumar(a,b)
19
```

Tokens

tokens.txt

```
1  <PalabraReservada,function>
2  <PalabraReservada,bool>
3  <Identificador,verdad>
4  <ParentesisAbierto,(>
5  <PalabraReservada,int>
6  <Identificador,num1>
7  <Coma,,>
8  <PalabraReservada,int>
9  <Identificador,num2>
10 <ParentesisCerrado,)>
11 <LlaveAbierta,{>
12 <PalabraReservada,if>
13 <ParentesisAbierto,(>
14 <Identificador,num1>
15 <MayorQue,>>
16 <Identificador,num2>
17 <ParentesisCerrado,)>
18 <PalabraReservada,return>
19 <PalabraReservada,true>
20 <PalabraReservada,return>
21 <PalabraReservada,false>
22 <LlaveCerrada,}>
23 <PalabraReservada,function>
24 <PalabraReservada,int>
25 <Identificador,sumar>
26 <ParentesisAbierto,(>
27 <PalabraReservada,int>
```

tokens.txt

```
26 <ParentesisAbierto,(>
27 <PalabraReservada,int>
28 <Identificador,a>
29 <Coma,,>
30 <PalabraReservada,int>
31 <Identificador,b>
32 <ParentesisCerrado,)>
33 <LlaveAbierta,{>
34 <PalabraReservada,var>
35 <PalabraReservada,int>
36 <Identificador,c>
37 <Asignacion,=>
38 <Identificador,a>
39 <Suma,+>
40 <Identificador,b>
41 <Identificador,verdad>
42 <ParentesisAbierto,(>
43 <Identificador,z>
44 <Coma,,>
45 <Identificador,x>
46 <ParentesisCerrado,)>
47 <PalabraReservada,return>
48 <Identificador,c>
49 <LlaveCerrada,}>
50 <Identificador,sumar>
51 <ParentesisAbierto,(>
52 <Identificador,a>
53 <Coma,,>
54 <Identificador,b>
55 <ParentesisCerrado,)>
56
```

Tabla de Símbolos

TablaDeSimbolos.txt

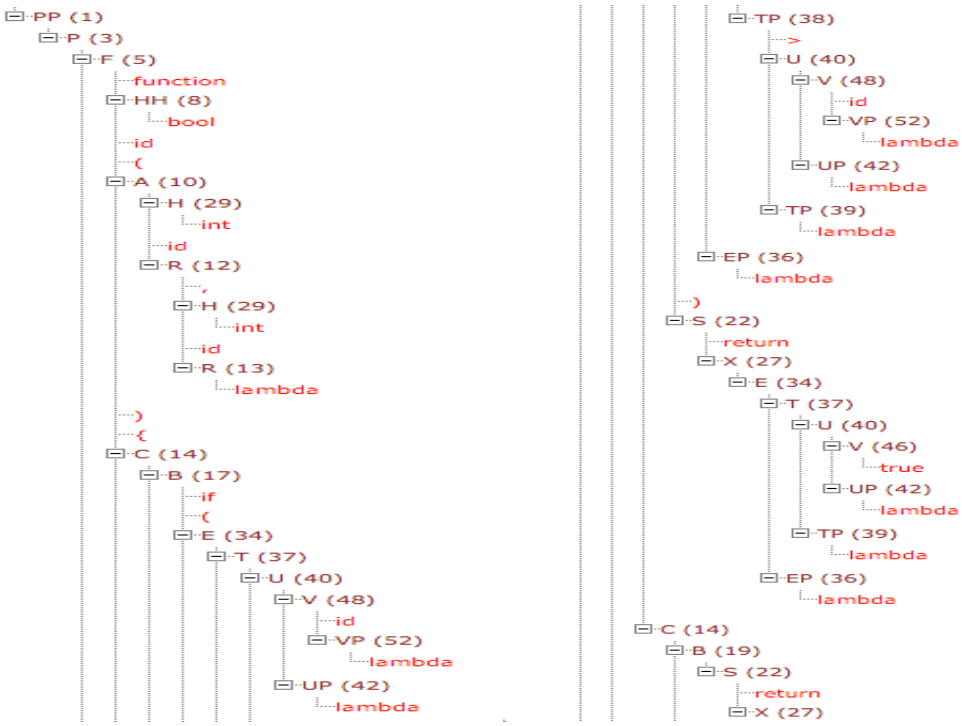
```
TABLA PRINCIPAL PALABRAS RESERVADAS #1:
* LEXEMA: 'var'
* LEXEMA: 'function'
* LEXEMA: 'prompt'
* LEXEMA: 'write'
* LEXEMA: 'while'
* LEXEMA: 'if'
* LEXEMA: 'return'
* LEXEMA: 'int'
* LEXEMA: 'chars'
* LEXEMA: 'bool'
* LEXEMA: 'true'
* LEXEMA: 'false'
------------------------------------------------------------
TABLA PRINCIPAL #2:
* (Esto es una funcion) LEXEMA: 'verdad'
 ATRIBUTOS:
 +tipo:'bool'
  +parametros:2
   +tipoParam1:'int'
    +tipoParam2:'int'
CONTENIDO DE LA FUNCION verdad #3:
* LEXEMA:'num1' (tipo de entrada 'parametro')
 +tipo:'int'
* LEXEMA:'num2' (tipo de entrada 'parametro')
 +tipo:'int'
TABLA PRINCIPAL #4:
* (Esto es una funcion) LEXEMA: 'sumar'
 ATRIBUTOS:
 +tipo:'int'
  +parametros:2
   +tipoParam1:'int'
    +tipoParam2:'int'
CONTENIDO DE LA FUNCION sumar #5:
* LEXEMA:'a' (tipo de entrada 'parametro')
 +tipo:'int'
* LEXEMA:'b' (tipo de entrada 'parametro')
 +tipo:'int'
* LEXEMA: 'c' (tipo de entrada 'variable')
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #6:
* LEXEMA: 'z'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #7:
* LEXEMA: 'x'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #8:
* LEXEMA: 'a'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #9:
* LEXEMA: 'b'
 +tipo:'int'
```

Parser

parser.txt

```
Descendente 1 3 5 8 10 29 12 29 13 14 17 34 37 40 48 52 42 38 40 48
52 42 39 36 22 27 34 37 40 46 42 39 36 14 19 22 27 34 37 40 47 42 39
36 15 3 5 6 10 29 12 29 13 14 16 29 32 34 37 40 48 52 41 48 52 42 39
36 14 19 23 24 53 34 37 40 48 52 42 39 36 55 34 37 40 48 52 42 39 36
56 14 19 22 27 34 37 40 48 52 42 39 36 15 2 19 23 24 53 34 37 40 48
52 42 39 36 55 34 37 40 48 52 42 39 36 56 4
```

Árbol



Prueba 4

Código Fuente

ejem.js

```
1  //SOLO ERA UN COMENTARIO
2
3  function chars escribir (chars hola, int mundo){
4      var chars c
5      while(mundo>0){
6          mundo--
7          write(hola)
8      }
9      c="Fin"
10     return c
11 }
12
13 function int sumar(int a, int b)
14 {
15     //Variables locales
16     var int c = (4+5+a)+b+sumar(a,b)
17
18     return c
19 }
20
21 sumar(a,b)
```

Tokens

tokens.txt

```
<PalabraReservada,function>
<PalabraReservada,chars>
<Identificador,escribir>
<ParentesisAbierto,(>
<PalabraReservada,chars>
<Identificador,hola>
<Coma,,>
<PalabraReservada,int>
<Identificador,mundo>
<ParentesisCerrado,)>
<LlaveAbierta,{>
<PalabraReservada,var>
<PalabraReservada,chars>
<Identificador,c>
<PalabraReservada,while>
<ParentesisAbierto,(>
<Identificador,mundo>
<MayorQue,>>
<Numero,0>
<ParentesisCerrado,)>
<LlaveAbierta,{>
<Identificador,mundo>
<PostDecremento,-->
<PalabraReservada,write>
<ParentesisAbierto,(>
<Identificador,hola>
<ParentesisCerrado,)>
<LlaveCerrada,}>
<Identificador,c>
<Asignacion,=>
<Cadena,"Fin">
<PalabraReservada,return>
<Identificador,c>
<LlaveCerrada,}>
<PalabraReservada,function>
<PalabraReservada,int>
<Identificador,sumar>
```

tokens.txt

```
<ParentesisAbierto,(>
<PalabraReservada,int>
<Identificador,a>
<Coma,,>
<PalabraReservada,int>
<Identificador,b>
<ParentesisCerrado,)>
<LlaveAbierta,{>
<PalabraReservada,var>
<PalabraReservada,int>
<Identificador,c>
<Asignacion,=>
<ParentesisAbierto,(>
<Numero,4>
<Suma,+>
<Numero,5>
<Suma,+>
<Identificador,a>
<ParentesisCerrado,)>
<Suma,+>
<Identificador,b>
<Suma,+>
<Identificador,sumar>
<ParentesisAbierto,(>
<Identificador,a>
<Coma,,>
<Identificador,b>
<ParentesisCerrado,)>
<PalabraReservada,return>
<Identificador,c>
<LlaveCerrada,}>
<Identificador,sumar>
<ParentesisAbierto,(>
<Identificador,a>
<Coma,,>
<Identificador,b>
<ParentesisCerrado,)>
```

Tabla de Símbolos

TablaDeSimbolos.txt

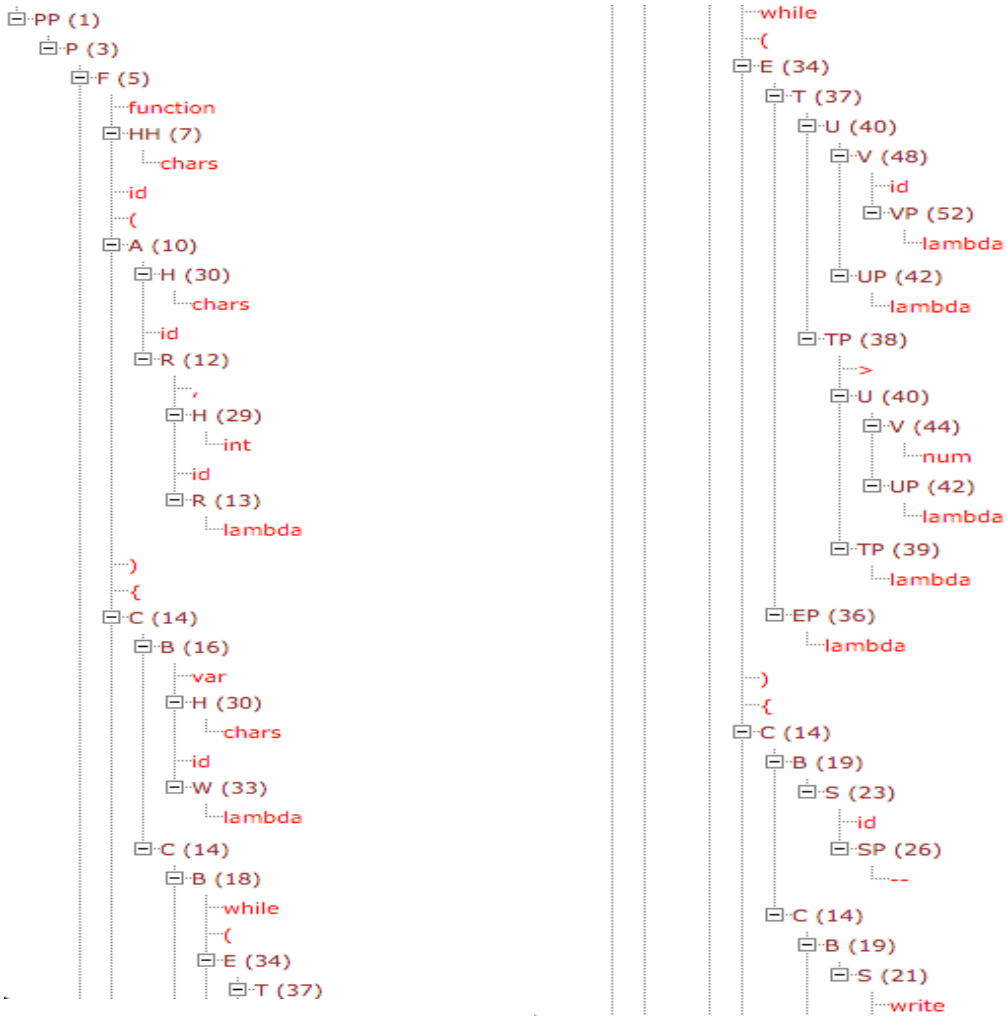
```
TABLA PRINCIPAL PALABRAS RESERVADAS #1:
* LEXEMA: 'var'
* LEXEMA: 'function'
* LEXEMA: 'prompt'
* LEXEMA: 'write'
* LEXEMA: 'while'
* LEXEMA: 'if'
* LEXEMA: 'return'
* LEXEMA: 'int'
* LEXEMA: 'chars'
* LEXEMA: 'bool'
* LEXEMA: 'true'
* LEXEMA: 'false'
------------------------------------------------------------
TABLA PRINCIPAL #2:
* (Esto es una funcion) LEXEMA: 'escribir'
 ATRIBUTOS:
 +tipo:'chars'
  +parametros:2
   +tipoParam1:'chars'
    +tipoParam2:'int'
CONTENIDO DE LA FUNCION escribir #3:
* LEXEMA:'hola' (tipo de entrada 'parametro')
 +tipo:'chars'
* LEXEMA:'mundo' (tipo de entrada 'parametro')
 +tipo:'int'
* LEXEMA: 'c' (tipo de entrada 'variable')
 +tipo:'chars'
TABLA PRINCIPAL #4:
* (Esto es una funcion) LEXEMA: 'sumar'
 ATRIBUTOS:
 +tipo:'int'
  +parametros:2
   +tipoParam1:'int'
    +tipoParam2:'int'
CONTENIDO DE LA FUNCION sumar #5:
* LEXEMA:'a' (tipo de entrada 'parametro')
 +tipo:'int'
* LEXEMA:'b' (tipo de entrada 'parametro')
 +tipo:'int'
* LEXEMA: 'c' (tipo de entrada 'variable')
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #6:
* LEXEMA: 'a'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #7:
* LEXEMA: 'b'
 +tipo:'int'
```

Parser

parser.txt

```
Descendente 1 3 5 7 10 30 12 29 13 14 16 30 33 14 18 34 37 40 48 52 42 38 40 44 42 39 36 14 19 23 26 14 19
21 34 37 40 48 52 42 39 36 15 14 19 23 25 34 37 40 45 42 39 36 14 19 22 27 34 37 40 48 52 42 39 36 15 3 5 6
10 29 12 29 13 14 16 29 32 34 37 40 43 34 37 40 44 41 44 41 48 52 42 39 36 41 48 52 41 48 49 53 34 37 40 48
52 42 39 36 55 34 37 40 48 52 42 39 36 56 42 39 36 14 19 22 27 34 37 40 48 52 42 39 36 15 2 19 23 24 53 34
37 40 48 52 42 39 36 55 34 37 40 48 52 42 39 36 56 4
```

Árbol



Prueba 5

Código Fuente

ejem.js

```
/*Varias lineas
 de comentarios
 en el codigo*/
var int num = 30
var chars yes
 function int sumaSimple(int b){
    return num + b
 }

var bool esFlag = true

/*Varias lineas
 de comentarios
 en cualquier parte del codigo*/

while(true){
    write("hola")
}
if(esFlag && true) write("mundo")

auxiliar = 300+sumaSimple(num)+2

if(num>2) yes = "Verdad"

/*Varias lineas
 de comentarios
 en el codigo*/
```

Tokens

tokens.txt

```
<PalabraReservada,var>
<PalabraReservada,int>
<Identificador,num>
<Asignacion,=>
<Numero,30>
<PalabraReservada,var>
<PalabraReservada,chars>
<Identificador,yes>
<PalabraReservada,function>
<PalabraReservada,int>
<Identificador,sumaSimple>
<ParentesisAbierto,(>
<PalabraReservada,int>
<Identificador,b>
<ParentesisCerrado,)>
<LlaveAbierta,{>
<PalabraReservada,return>
<Identificador,num>
<Suma,+>
<Identificador,b>
<LlaveCerrada,}>
<PalabraReservada,var>
<PalabraReservada,bool>
<Identificador,esFlag>
<Asignacion,=>
<PalabraReservada,true>
<PalabraReservada,while>
<ParentesisAbierto,(>
<PalabraReservada,true>
<ParentesisCerrado,)>
<LlaveAbierta,{>
<PalabraReservada,write>
<ParentesisAbierto,(>
```

tokens.txt

```
<Cadena,"hola">
<ParentesisCerrado,)>
<LlaveCerrada,}>
<PalabraReservada,if>
<ParentesisAbierto,(>
<Identificador,esFlag>
<And,&&>
<PalabraReservada,true>
<ParentesisCerrado,)>
<PalabraReservada,write>
<ParentesisAbierto,(>
<Cadena,"mundo">
<ParentesisCerrado,)>
<Identificador,auxiliar>
<Asignacion,=>
<Numero,300>
<Suma,+>
<Identificador,sumaSimple>
<ParentesisAbierto,(>
<Identificador,num>
<ParentesisCerrado,)>
<Suma,+>
<Numero,2>
<PalabraReservada,if>
<ParentesisAbierto,(>
<Identificador,num>
<MayorQue,>>
<Identificador,Z>
<ParentesisCerrado,)>
<Identificador,yes>
<Asignacion,=>
<Cadena,"Verdad">
```

Tabla de Símbolos

TablaDeSimbolos.txt

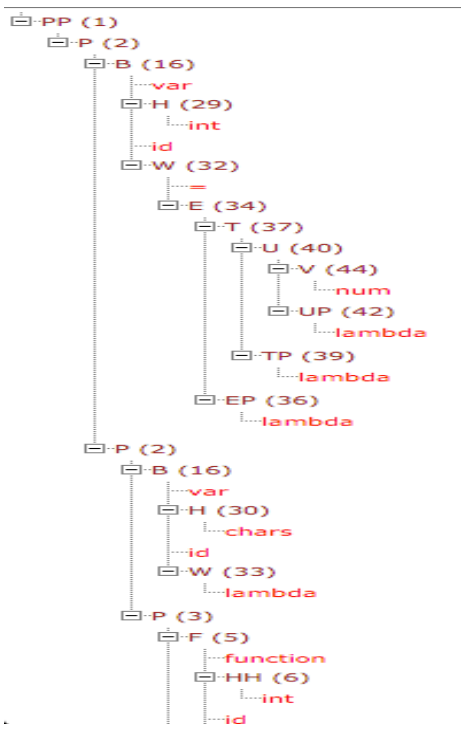
```
TABLA PRINCIPAL PALABRAS RESERVADAS #1:
* LEXEMA: 'var'
* LEXEMA: 'function'
* LEXEMA: 'prompt'
* LEXEMA: 'write'
* LEXEMA: 'while'
* LEXEMA: 'if'
* LEXEMA: 'return'
* LEXEMA: 'int'
* LEXEMA: 'chars'
* LEXEMA: 'bool'
* LEXEMA: 'true'
* LEXEMA: 'false'
-----------------------------------------------------
TABLA PRINCIPAL CON VARIABLES #2:
* LEXEMA: 'num'
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #3:
* LEXEMA: 'yes'
 +tipo:'chars'
TABLA PRINCIPAL #4:
* (Esto es una funcion) LEXEMA: 'sumaSimple'
 ATRIBUTOS:
 +tipo:'int'
  +parametros:1
   +tipoParam1:'int'
CONTENIDO DE LA FUNCION sumaSimple #5:
* LEXEMA:'b' (tipo de entrada 'parametro')
 +tipo:'int'
TABLA PRINCIPAL CON VARIABLES #6:
* LEXEMA: 'esFlag'
 +tipo:'bool'
TABLA PRINCIPAL CON VARIABLES #7:
* LEXEMA: 'auxiliar'
 +tipo:'int'
```

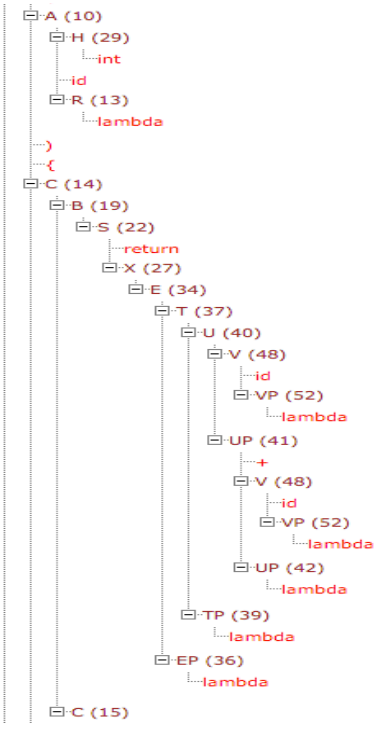
Parser

parser.txt

```
Descendente 1 2 16 29 32 34 37 40 44 42 39 36 2 16 30 33 3 5 6 10 29 13 14 19 22
27 34 37 40 48 52 41 48 52 42 39 36 15 2 16 31 32 34 37 40 46 42 39 36 2 18 34
37 40 46 42 39 36 14 19 21 34 37 40 45 42 39 36 15 2 17 34 37 40 48 52 42 39 35
37 40 46 42 39 36 21 34 37 40 45 42 39 36 2 19 23 25 34 37 40 44 41 48 49 53 34
37 40 48 52 42 39 36 56 41 44 42 39 36 2 17 34 37 40 48 52 42 38 40 48 52 42 39
36 23 25 34 37 40 45 42 39 36 4
```

Árbol


PP (1)
P (2)
B (16)
var
H (29)
int
id
W (32)
=
E (34)
T (37)
U (40)
V (44)
num
UP (42)
lambda
TP (39)
lambda
EP (36)
lambda
P (2)
B (16)
var
H (30)
chars
id
W (33)
lambda
P (3)
F (5)
function
HH (6)
int
id


A (10)
H (29)
int
id
R (13)
lambda
)
{
C (14)
B (19)
S (22)
return
X (27)
E (34)
T (37)
U (40)
V (48)
id
VP (52)
lambda
UP (41)
+
V (48)
id
VP (52)
lambda
UP (42)
lambda
TP (39)
lambda
EP (36)
lambda
C (15)

Casos de Pruebas Incorrectos

Prueba 1

Código Fuente

ejem.js

```
1  //PRIMERA PRUEBA BUENA
2  //Variables globales
3  var int a
4  var int b
5  var chars s = "APROBADO =)"
6  function int sumar(chars num1, int num2)
7  {
8      while(true){
9          if(a>b)
10         return num1+num2
11     }
12     num1--
13     return true
14 }
15
16 write(s)
```

Errores

log.txt

```
1 Error semántico: el valor de retorno no es correcto --> int != chars: num1 en la línea 10
2 Error semantico: la comprobacion de postDecremento no es correcta Linea: 12
3 Error semántico: el valor de retorno no es correcto --> int != bool: true en la línea 13
4 Fin de ejecuciOn
```

Prueba 2

Código Fuente

ejem.js

```
1  //PRIMERA PRUEBA BUENA
2  //Variables globales
3  var int a = 2
4  var bool b = 3
5  var chars s = "APROBADO =)"
6  function bool verdad (int num1, int num2){
7
8      if(num1>num2) return true
9      prompt(true)
10     prompt(num2)
11     return false
12 }
13
14 function int sumar()
15 {
16     //Variables locales
17     var int c
18     c = verdad(a,b)
19     if(c)
20         write(s)
21     return "Ups"
22 }
23
24 nota = sumar()
```

Errores

Error Semántico

```
log.txt
1  Error semántico: tipo izqE(bool) no coincide tipo der (int) --> 3 en la línea 4
2
```

Error Sintáctico en la línea 9

```
//PRIMERA PRUEBA BUENA
//Variables globales
var int a = 2
var bool b = 3
var chars s = "APROBADO =)"
function bool verdad (int num1, int num2){

    if(num1>num2) return true
    prompt(true)
    prompt(num2)
    return false
}

function int sumar()
{
    //Variables local
    var int c
    c = verdad(a,b)
    if(c)
        write(s)
    return "Ups"
}

nota = sumar()
```

```
Administrador: C:\Windows\System32\cmd.exe

C:\Users\Robinson\Documents\PDL>java LexicoSintactico < ejem.txt > log.txt

C:\Users\Robinson\Documents\PDL>java LexicoSintactico < ejem.txt > log.txt

C:\Users\Robinson\Documents\PDL>java LexicoSintactico < ejem.txt > log.txt
Exception in thread "main" ParseException: Encountered " "true" "true "" at line 9, column 16.
Was expecting:
    <ID> ...

        at LexicoSintactico.generateParseException(LexicoSintactico.java:1642)
        at LexicoSintactico.jj_consume_token(LexicoSintactico.java:1524)
        at LexicoSintactico.S(LexicoSintactico.java:322)
        at LexicoSintactico.B(LexicoSintactico.java:306)
        at LexicoSintactico.C(LexicoSintactico.java:235)
        at LexicoSintactico.C(LexicoSintactico.java:236)
        at LexicoSintactico.F(LexicoSintactico.java:124)
        at LexicoSintactico.P(LexicoSintactico.java:80)
        at LexicoSintactico.P(LexicoSintactico.java:76)
        at LexicoSintactico.P(LexicoSintactico.java:76)
        at LexicoSintactico.P(LexicoSintactico.java:76)
        at LexicoSintactico.Iniciar(LexicoSintactico.java:61)
        at LexicoSintactico.main(LexicoSintactico.java:51)

C:\Users\Robinson\Documents\PDL>
```

Prueba 3

Código Fuente

```
ejem.js
1  //PRIMERA PRUEBA BUENA
2
3  function bool verdad (int num1, int num2){
4
5      if(num1>num2) return true
6
7      return "UPS"
8  }
9
10 function int sumar(int a, bool b)
11 {
12     //Variables locales
13     var int c = "hola" + b
14     verdad(z,x)
15     return c
16 }
17
18 sumar(a,b)
19
```

Errores

log.txt

```
Error semántico: el valor de retorno no es correcto --> bool != chars: "UPS" en la línea 7
Error semántico: tipoX izq(int) no coincide tipo der (chars) --> "hola" en la línea 13
Error semantico: tipo izqD(int) no coincide tipo der (bool) --> b en la línea 13
Error semantico: la comprobacion de suma no es correcta Linea: 13
Error Semático: la función sumar no está declarada con nº de parametros: 2 o con tipo de parametros distintos
Fin de ejecuciOn
```

Prueba 4

Código Fuente

ejem.js

```
/*Varias lineas
 de comentarios
 en el codigo*/

function chars escribir (chars hola, int mundo){
    var bool c
    while(mundo>0){
        mundo--
        write(hola)
    }
    c="Fin"
    return c
}

function int sumar(bool a, int b)
{
    //Variables locales
    var int c = (4+5+a)+b+sumar(a,b)

    return c
}

sumar(a,b)
llamadaInexistente()
```

Errores

log.txt

```
Error semántico: tipoX izq(bool) no coincide tipo der (chars) --> "Fin" en la línea 11
Error semántico: el valor de retorno no es correcto --> chars != bool: c en la línea 12
Error semantico: tipo izqD(int) no coincide tipo der (bool) --> a en la línea 18
Error semantico: la comprobacion de suma no es correcta Linea: 18
Error Semático: la función sumar no está declarada con nº de parametros: 2 o con tipo de parametros distintos en la línea 23
Error Semático: la función llamadaInexistente no está declarada con nº de parametros: 0 o con tipo de parametros distintos en
Fin de ejecuciOn
```

Prueba 5

Código Fuente

```
ejem.js
/*Varias lineas
 de comentarios
 en el codigo*/
var int num = 30
var chars num = true

 function int sumaSimple(int b){
    var int b
    return num + b
 }

var bool esFlag = true + 2

/*Varias lineas
 de comentarios
 en cualquier parte del codigo*/

while("f"){
    write("hola")
}
if(esFlag && true) write("mundo")

auxiliar = 303330+sumaSimple(num)+2

if(num>Z) yes = "Verdad"

prompt(num)

/*Varias lineas
 de comentarios
 en el codigo*/
```

Errores

Errores Semánticos

```
log.txt
Ya existe la variable--> num Linea: 5
Error semántico: tipoY izq(chars) no coincide tipo der (bool) --> true en la línea 5
Ya existe la variable local--> b Linea: 8
Error semántico: tipo izqE(bool) no coincide tipo der (int) --> 2 en la línea 12
Error semantico: la comprobacion de suma no es correcta Linea: 12
Error semántico: tipoX izq(int) no coincide tipo der (chars) --> "Verdad" en la línea 25
Fin de ejecuciOn
```

Error Léxico

```
Ya existe la variable--> num Linea: 5
Error semántico: tipoY izq(chars) no coincide tipo der (bool) --> true en la línea 5
Ya existe la variable local--> b Linea: 8
Error semántico: tipo izqE(bool) no coincide tipo der (int) --> 2 en la línea 12
Error semantico: la comprobacion de suma no es correcta Linea: 12
Error semántico: tipoX izq(int) no coincide tipo der (chars) --> "Verdad" en la línea 25
Fin de ejecuciOn
```

```
Administrador: C:\Windows\System32\cmd.exe
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>
C:\Users\Robinson\Documents\PDL>java LexicoSintactico < ejem.txt > log.txt
Numero mayor que 32767

C:\Users\Robinson\Documents\PDL>
```