**PROCESADORES DE LENGUAJES**



**Grupo 64**

# Christian Paniagua Paniagua x150380

# Javier Pérez Martín x150147

# Pablo Heras Aranzana x150046

# **Diseño Analizador Léxico**

## 1.1.Tokens

**<cteent, número>**

**<CAD, “lexema”>**

**<ctebool, 1>** (true)

**<ctebool, 0>** (false)

**<MAS, ->** (+)

**<MENOS, ->** (-)

**<MUL, ->** (\*)

**<DIV, ->** (/)

**<MOD, ->** (%)

**<AND, ->** (&&)

**<LLAVA, ->** ( { )

**<LLAVC, ->** ( } )

**<PARA, ->** ( ( )

**<PARC, ->** ( ) )

**<FIN, - >** ( ; )

**<SIG, - >** ( , )

**<OR, ->** (||)

**<NOT, ->** ( ! )

**<ASIGR, ->** (-=)

**<ASIG, - >** (=)

**<OPMAY, - >** (>)

**<OPMEN, - >** (<)

**<OPIG, - >** (==)

**<OPDISTINTO,- >** (!=)

**<DOSPUNTOS, - >** (:)

**<PR, 1 >** (int)

**<PR, 2 >** (bool)

**<PR, 3 >** (string)

**<PR, 4 >** (if)

**<PR, 5 >** (default)

**<PR, 6 >** (break)

**<PR, 7 >** (return)

**<PR, 8 >** (function)

**<PR, 9 >** (var)

**<PR, 10 >** (switch)

**<PR, 11 >** (case)

**<PR, 12 >** (print)

**<PR, 13 >** (promt)

**<Id, pos\_ts>** (siendo “pos\_ts” la posicion que ocupa el id dentro de la tabla de simbolos)

# 1.2.Gramática

S --> del S | d E | 'C | l I | / N | = D | : | + | - D1 | & J | '|' K | \* | [ | ] | ; | , | ! D2 | < D3 | D4 >

C --> \ C' | oc1 C | '

C' --> 't' C | 'n' C | ' C | " C | \ C

D --> = | λ

D1 --> = | λ

D2 --> = | λ

D3 --> = | λ

D4 --> = | λ

E --> d E | λ

I --> l I | d I | \_ I | λ

J --> &

K --> '|'

N --> \* N' | λ

N' --> \* N'' | oc2 N'

N''--> \* N'' | / S | oc3 N'

Definiciones:

del={esp, tab, eol}

d = {0..9}

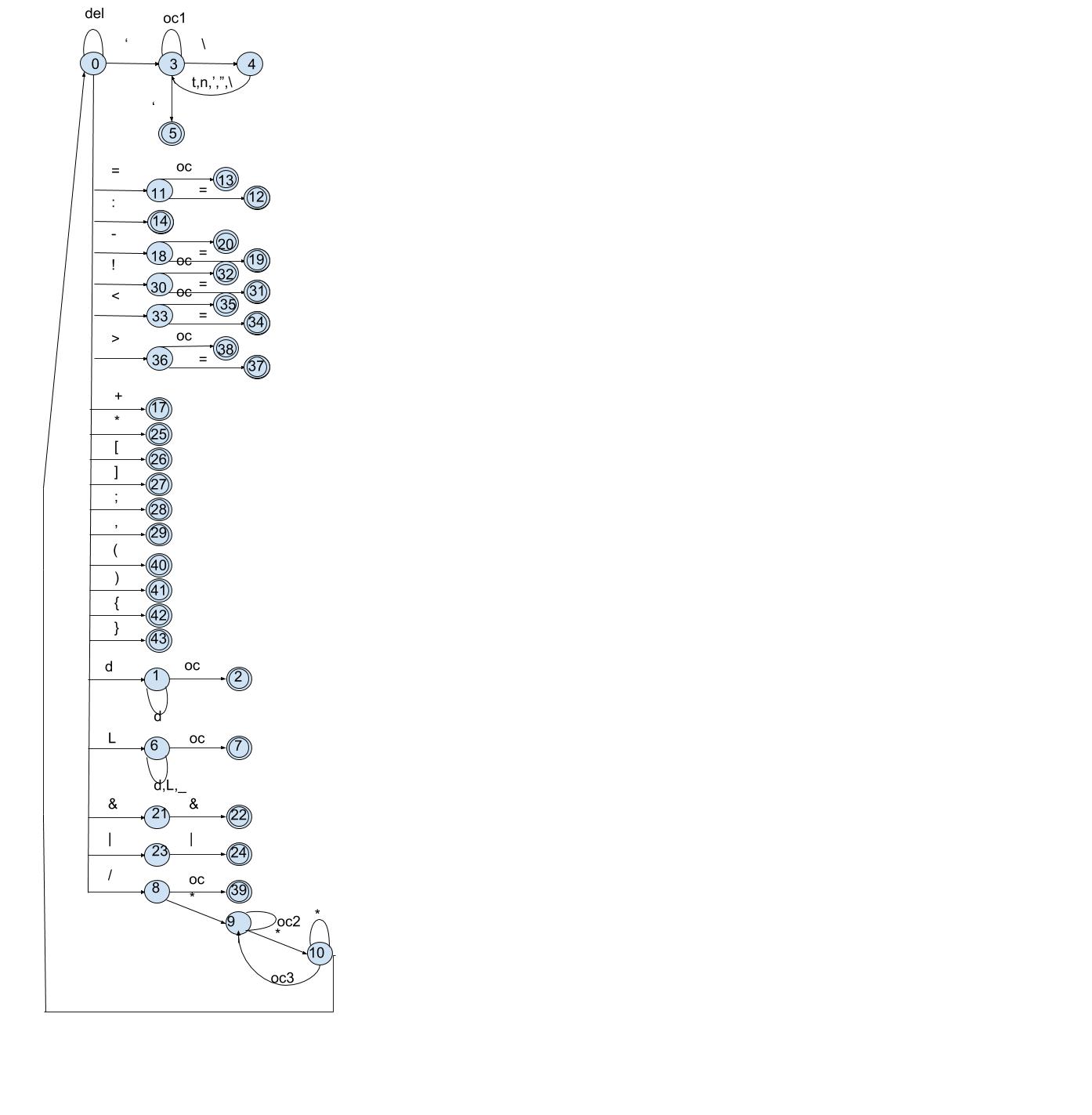
l = {a..z, A..Z}

oc1 = {cualquier carácter imprimible} - {\}

oc2 = {cualquier carácter imprimible} - {\*}

oc3 = {cualquier carácter imprimible} - {/,\*}

## 1.3.Autómata



1.4.Acciones semánticas

L=Leer\_carácter()

0:1 -->L, valor = valor\_ascii(d)

1:1 -->L, valor = valor\*10 + valor\_ascii(d)

1:2 -->if( valor > 32767) then **ERROR**

else **G.Token(<ENT, valor>)**

0:3 -->L,lexema = carácter\_ascii( ‘ )

3:3 -->L,lexema = lexema ⊕ carácter ascii(oc1)

3:4 -->L,lexema = lexema ⊕ carácter ascii( \ )

4:3 --> L,lexema = lexema ⊕ {

carácter ascii( t )

carácter ascii( n )

carácter ascii( ‘ )

carácter ascii( “ )

carácter ascii( \ )

}

3:5-->L, lexema = lexema ⊕ carácter ascii( ‘ )

0:6-->L, lexema = carácter\_asciii( l )

6:6-->L, lexema = lexema ⊕ {

carácter ascii( l )

carácter ascii( d )

carácter ascii( -)

}

6:7--> If (pos = TS(lexema)

then G.token(< Id , pos>)

Else if ( pos = T.palreservada(lexema))

then **G.token(<PR , pos>)**

else pos = Insertar Ts(lexema),

Gtoken(< Id, pos>)

0:8-->L

0:39-->**G.token(<DIV, ~ >)**

8:9-->L

9:9-->L

9:10-->L

10:10-->L

10:0-->L

0:11-->L

11:12-->L, **G.token(<OP\_IG, pos>)**

11:13-->**G.token(<Asig, ~ >)**

0:14-->L, **G.token(<Dos\_puntos, ~ >)**

0:17-->L, **G.token(<Mas, ~ >)**

0:18-->L

18:19-->L, **G.token(<Asig\_R, ~ >)**

18:20--> **G.token(<Menos, ~ >)**

0:21-->L

21:22-->L, **G.token(<AND, ~ >)**

0:23-->L

23:24-->L, **G.token(<OR, ~ >)**

0:25-->L, **G.token(<MUL, ~ >)**

0:26-->L, **G.token(<CORA, ~ >)**

0:27-->L, **G.token(<CORC, ~ >)**

0:28-->L, **G.token(<FIN, ~ >)**

0:29-->L,

0:30-->L

30:31-->L, **G.token(<OP\_DISTINTO, ~ >)**

30:32-->L, **G.token(<NOT, ~ >)**

0:33-->L

33:34-->L, **G.token(<OP\_MEN\_IG, ~ >)**

33:35-->L, **G.token(<OP\_MEN, ~ >)**

0:36-->L

36:38-->L, **G.token(<OP\_MAY\_IG, ~ >)**

36:37-->L, **G.token(<OP\_MAY, ~ >)**

0:40-->L, **G.token(<PARA, ~ >)**

0:41-->L, **G.token(<PARC, ~ >)**

0:42-->L, **G.token(<LLAVA, ~ >)**

0:43-->L, **G.token(<LLAVC, ~ >)**

# **Diseño Analizador Sintáctico**

## Gramática

Terminales = { var id ; if ( ) while } { switch case break default int bool string return print

prompt function , || && == != < > -= ! = + - \* / % CAD : cteent ctebool }

NoTerminales = { P B T S \_S F H L Q A K X C E \_E R \_R U \_U V \_V W \_W Z \_Z G \_G D }

Axioma = P

Producciones = {

P -> B P

P -> F P

P -> lambda

B -> var T id ;

B -> if ( E ) S

B -> while ( E ) { C }

B -> switch ( E ) { D }

B -> S

T -> int

T -> bool

T -> string

S -> id \_S

S -> return X ;

S -> print ( E ) ;

S -> prompt ( id ) ;

S -> break ;

\_S -> = E ;

\_S -> -= E ;

\_S -> ( L ) ;

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

H -> T

H -> lambda

L -> E Q

L -> lambda

Q -> , E Q

Q -> lambda

A -> T id K

A -> lambda

K -> , T id K

K -> lambda

X -> E

X -> lambda

C -> B C

C -> lambda

E -> R \_E

\_E -> || R \_E

\_E -> lambda

R -> U \_R

\_R -> && U \_R

\_R -> lambda

U -> V \_U

\_U -> == V \_U

\_U -> != V \_U

\_U -> lambda

V -> W \_V

\_V -> > W \_V

\_V -> < W \_V

\_V -> lambda

W -> Z \_W

\_W -> + Z \_W

\_W -> - Z \_W

\_W -> lambda

Z -> G \_Z

\_Z -> \* G \_Z

\_Z -> / G \_Z

\_Z -> % G \_Z

\_Z -> lambda

G -> ! G

G -> id \_G

G -> ( E )

G -> cteent

G -> CAD

G -> ctebool

\_G -> ( L )

\_G -> lambda

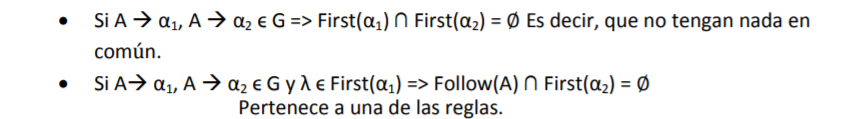
D -> case cteent : C D

D -> default : C

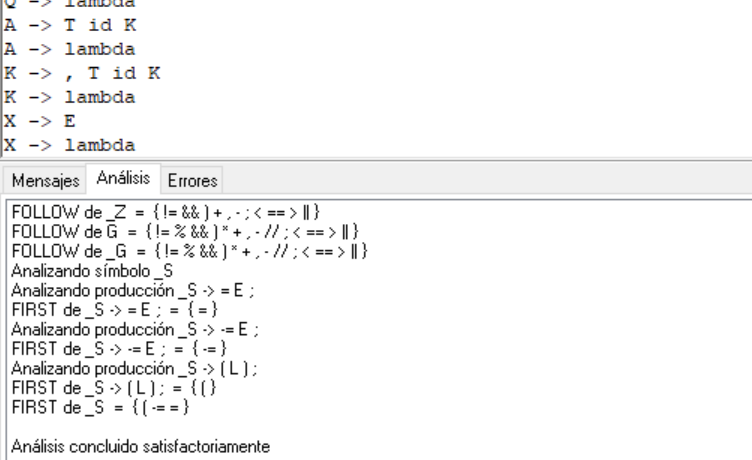
}

## Demostración LL(1)

Puesto que nuestro método de análisis sintáctico es el “Descendente Recursivo” nuestra gramática debe ser una gramática LL(1). Este tipo de gramáticas son no ambiguas, no recursivas por la izquierda y factorizadas por la izquierda. A parte de estas condiciones necesarias, todas las producciones de la gramática con mismo antecedente deben cumplir (dos a dos) la condición LL(1):



Tras eliminar las recursividades por la izquierda y factorizar nuestra gramática empleamos la herramienta SDGLL1 proporcionada en la web de la asignatura para comprobar que nuestra gramática es LL(1). Tras analizar la gramática en la herramienta obtuvimos los First y Follows de las distintas reglas, así como la confirmación de que la gramática es LL(1).



## 2.3.Procedimientos

A continuación, mostramos el código de los procedimientos que realizarán el análisis sintáctico. El resultado de éste se volcará en el fichero de parse.

print("Des ");

Proc P;

if(st ϵ First(B P)={ break id if print prompt return switch var while }) then

{

print("1");

B();

P();

}

else if(st ϵ First(F P)={ function }) then

{

print("2");

F();

P();

}

else if(st ϵ Follow(P)={ $ })

{

print("3");

}

else ERROR;

END;

Proc B;

if(st ϵ First(var T id ;)={ var }) then

{

print("4");

EqT(var);

T();

EqT(id);

EqT(;);

}

else if(st ϵ First(if ( E ) S)={ if }) then

{

print("5");

EqT(if);

EqT(();

E();

EqT());

S();

}

else if(st ϵ First(while ( E ) { C })={ while }) then

{

print("6");

EqT(while);

EqT(();

E();

EqT());

EqT({);

C();

EqT(});

}

else if(st ϵ First(switch ( E ) { D })={ switch }) then

{

print("7");

EqT(switch);

EqT(();

E();

EqT());

EqT({);

D();

EqT(});

}

else if(st ϵ First(S)={ break id print prompt return }) then

{

print("8");

S();

}

else ERROR;

END;

Proc T;

if(st ϵ First(int)={ int }) then

{

print("9");

EqT(int);

}

else if(st ϵ First(bool)={ bool }) then

{

print("10");

EqT(bool);

}

else if(st ϵ First(string)={ string }) then

{

print("11");

EqT(string);

}

else ERROR;

END;

Proc S;

if(st ϵ First(id S')={ id }) then

{

print("12");

EqT(id);

S'();

}

else if(st ϵ First(return X ;)={ return }) then

{

print("13");

EqT(return);

X();

EqT(;);

}

else if(st ϵ First(print ( E ) ;)={ print }) then

{

print("14");

EqT(print);

EqT(();

E();

EqT());

EqT(;);

}

else if(st ϵ First(prompt ( id ) ;)={ prompt }) then

{

print("15");

EqT(prompt);

EqT(();

EqT(id);

EqT());

EqT(;);

}

else if(st ϵ First( break ;)={ break }) then

{

print("16");

EqT(break);

EqT(;);

}

else ERROR;

END;

Proc S';

if(st ϵ First(= E ;)={ = }) then

{

print("17");

EqT(=);

E();

EqT(;);

}

else if(st ϵ First(-= E ;)={ -= }) then

{

print("18");

EqT(-=);

E();

EqT(;);

}

else if(st ϵ First(( L ) ;)={ ( }) then

{

print("19");

EqT(();

L();

EqT());

EqT(;);

}

else ERROR;

END;

Proc F;

if(st ϵ First(function H id ( A ) { C })={ function }) then

{

print("20");

EqT(function);

H();

EqT(id);

EqT(();

A();

EqT());

EqT({);

C();

EqT(});

}

else ERROR;

END;

Proc H;

if(st ϵ First(T)={ int bool string }) then

{

print("21");

T();

}

else if(st ϵ Follow(H)={ id }) then

{

print("22");

}

else ERROR;

END;

Proc L;

if(st ϵ First(E Q)={ ! ( cte\_bool CAD cte\_ent id }) then

{

print("23");

E();

Q();

}

else if(st ϵ Follow(L)={ ) }) then

{

print("24");

}

else ERROR;

END;

Proc Q;

if(st ϵ First(, E Q)={ , }) then

{

print("25");

EqT(,);

E();

Q();

}

else if(st ϵ Follow(Q)={ ) }) then

{

print("26");

}

else ERROR;

END;

Proc A;

if(st ϵ First(T id K)={ int bool string }) then

{

print("27");

T();

EqT(id);

K();

}

else if(st ϵ Follow(A)={ ) ; }) then

{

print("28");

}

else ERROR;

END;

Proc K;

if(st ϵ First(, T id K)={ , }) then

{

print("29");

EqT(,);

T();

EqT(id);

K();

}

else if(st ϵ Follow(K)={ ) }) then

{

print("30");

}

else ERROR;

END;

Proc X;

if(st ϵ First(E)={ ! ( cte\_bool CAD cte\_ent id }) then

{

print("31");

E();

}

else if(st ϵ Follow(X)={ ; }) then

{

print("32");

}

else ERROR;

END;

Proc C;

if(st ϵ First(B C)={ break id if print prompt return switch var while }) then

{

print("33");

B();

C();

}

else if(st ϵ Follow(C)={ case } default }) then

{

print("34");

}

else ERROR;

END;

Proc E;

if(st ϵ First(R E')={ ! ( cte\_bool CAD cte\_ent id }) then

{

print("35");

R();

E'();

}

else ERROR;

END;

Proc E';

if(st ϵ First(|| R E')={ || }) then

{

print("36");

EqT(||);

R();

E'();

}

else if(st ϵ Follow(E')={ ) , ; }) then

{

print("37");

}

else ERROR;

END;

Proc R;

if(st ϵ First(U R')={ ! ( cte\_bool CAD cte\_ent id }) then

{

print("38");

U();

R'();

}

else ERROR;

END;

Proc R';

if(st ϵ First(&& U R')={ && }) then

{

print("39");

EqT(&&);

U();

R'();

}

else if(st ϵ Follow(R')={ ) , ; || }) then

{

print("40");

}

else ERROR;

END;

Proc U;

if(st ϵ First(V U')={ ! ( cte\_bool CAD cte\_ent id }) then

{

print("41");

V();

U'();

}

else ERROR;

END;

Proc U';

if(st ϵ First(== V U')={ == }) then

{

print("42");

EqT(==);

V();

U'();

}

else if(st ϵ First(!= V U')={ != }) then

{

print("43");

EqT(!=);

V();

U'();

}

else if(st ϵ Follow(U')={ && ) , ; || }) then

{

print("44");

}

else ERROR;

END;

Proc V;

if(st ϵ First(W V')={ ! ( cte\_bool CAD cte\_ent id }) then

{

print("45");

W();

V'();

}

else ERROR;

END;

Proc V';

if(st ϵ First(> W V')={ > }) then

{

print("46");

EqT(>);

W();

V'();

}

else if(st ϵ First(< W V')={ < }) then

{

print("47");

EqT(<);

W();

V'();

}

else if(st ϵ Follow(V')={ != && ) , ; == || }) then

{

print("48");

}

else ERROR;

END;

Proc W;

if(st ϵ First(Z W')={ ! ( cte\_bool CAD cte\_ent id }) then

{

print("49");

Z();

W'();

}

else ERROR;

END;

Proc W';

if(st ϵ First(+ Z W')={ + }) then

{

print("50");

EqT(+);

Z();

W'();

}

else if(st ϵ First(- Z W')={ - }) then

{

print("51");

EqT(-);

Z();

W'();

}

else if(st ϵ Follow(W')={ != && ) , ; < == > || }) then

{

print("52");

}

else ERROR;

END;

Proc Z;

if(st ϵ First(G Z')={ ! ( cte\_bool CAD cte\_ent id }) then

{

print("53");

G();

Z'();

}

else ERROR;

END;

Proc Z';

if(st ϵ First(\* G Z')={ \* }) then

{

print("54");

EqT(\*);

G();

Z'();

}

else if(st ϵ First(/ G Z')={ / }) then

{

print("55");

EqT(/);

G();

Z'();

}

else if(st ϵ First(% G Z')={ % }) then

{

print("56");

EqT(%);

G();

Z'();

}

else if(st ϵ Follow(Z')={ != && ) + , - ; < == > || }) then

{

print("57");

}

else ERROR;

END;

Proc G;

if(st ϵ First(! G)={ ! }) then

{

print("58");

EqT(!);

G();

}

else if(st ϵ First(id G')={ id }) then

{

print("59");

EqT(id);

G'();

}

else if(st ϵ First(( E ))={ ( }) then

{

print("60");

EqT(();

E();

EqT());

}

else if(st ϵ First(cte\_ent)={ cte\_ent }) then

{

print("61");

EqT(cte\_ent);

}

else if(st ϵ First(CAD)={ CAD }) then

{

print("62");

EqT(CAD);

}

else if(st ϵ First(cte\_bool)={ cte\_bool }) then

{

print("63");

EqT(cte\_bool);

}

else ERROR;

END;

Proc G';

if(st ϵ First(( L ))={ ( }) then

{

print("64");

EqT(();

L();

EqT());

}

else if(st ϵ Follow(G')={ != % && ) \* + , - / ; < == > || }) then

{

print("65");

}

else ERROR;

END;

Proc D;

if(st ϵ First(case cte\_ent : C D)={ case }) then

{

print("66");

EqT(case);

EqT(cte\_ent);

EqT(:);

C();

D();

}

else if(st ϵ ) then

{

print("67");

EqT(default);

EqT(:);

C();

}

else ERROR;

END;