UNIVERSIDAD POLITECNICA DE MADRID

**PRACTICA: ANALIZADOR LEXICO Y SINTACTICO**

Autores: Jiménez Pérez Juan, N:190204

Hernandez Pérez Jesus, N: 190295

Yanez Soffia Miguel, N: 190322

25 de octubre del 2021

# Índice

Contenido

[Índice 2](#_Toc89102274)

[Diseño de la practica 3](#_Toc89102275)

[Consideraciones 3](#_Toc89102276)

[Analizador Léxico 3](#_Toc89102277)

[Tokens 3](#_Toc89102278)

[Autómata 4](#_Toc89102279)

[Gramática 4](#_Toc89102280)

[Acciones semánticas 4](#_Toc89102281)

[Error 5](#_Toc89102282)

[Tabla de símbolos 5](#_Toc89102283)

[Analizador sintáctico 5](#_Toc89102284)

[Gramática 5](#_Toc89102285)

[Condiciones LL(1) 6](#_Toc89102286)

[Pseudo-Código 7](#_Toc89102287)

[Analizador Sintáctico 7](#_Toc89102288)

[Función Equipara 7](#_Toc89102289)

[Producciones 7](#_Toc89102290)

[Función Error 10](#_Toc89102291)

[Casos de prueba 11](#_Toc89102292)

[Caso 1 (correcto) 11](#_Toc89102293)

[Codigo: 11](#_Toc89102294)

[Tokens: 13](#_Toc89102295)

[Tabla de símbolos 18](#_Toc89102296)

[Caso 2 (correcto) 19](#_Toc89102297)

[Código 19](#_Toc89102298)

[Tokens 20](#_Toc89102299)

[Tabla de símbolos 25](#_Toc89102300)

[Caso 3 (correcto) 26](#_Toc89102301)

[Código 26](#_Toc89102302)

[Tokens 27](#_Toc89102303)

[Tabla de símbolos 32](#_Toc89102304)

[Caso 4 (error) 34](#_Toc89102305)

[Código 34](#_Toc89102306)

[Tokens 35](#_Toc89102307)

[Tabla de símbolos 40](#_Toc89102308)

[Caso 5 (error) 42](#_Toc89102309)

[Código 42](#_Toc89102310)

[Tokens 43](#_Toc89102311)

[Tabla de símbolos 48](#_Toc89102312)

[Caso 6 (error) 50](#_Toc89102313)

[Código 50](#_Toc89102314)

[Tokens 51](#_Toc89102315)

[Tabla de símbolos 57](#_Toc89102316)

# Diseño de la practica

Para el desarrollo del analizador léxico, hemos escogido el lenguaje de programación Java con la librería JFlex. Para el desarrollo del analizador sintáctico, hemos escogido el lenguaje de programación Java.

## Consideraciones

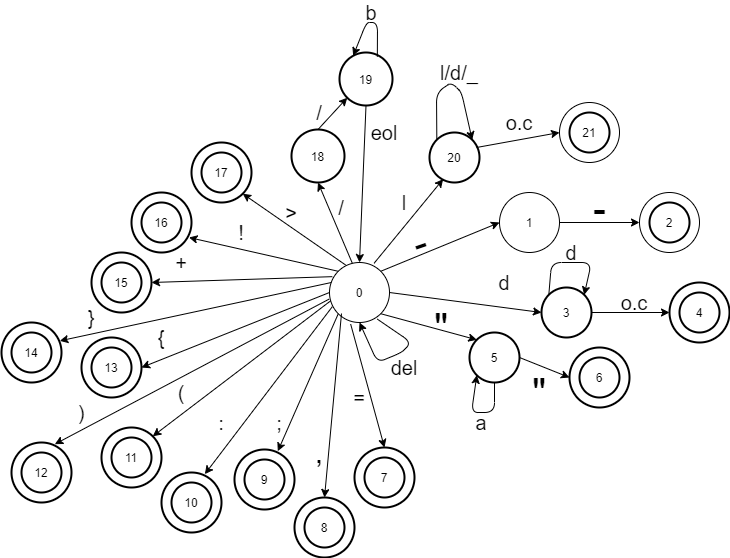
* Hemos utilizado los tokens obligatorios y de tokens de grupo la negación (!) y el mayor (>).
* Los comentarios se hacen con el doble barra diagonal (//) y finalizan con un salto de línea.
* Los enteros están en el rango [0 , 32767].
* Las cadenas están dentro de comillas.
* Un identificador es cualquier combinación de letras, números y barra baja (\_) que comience con una letra.

# Analizador Léxico

## Tokens

* Boolean: <boolean , >
* Break: <break, >
* Case: <case, >
* Function: <function, >
* If: <if, >
* Input: <input, >
* Int: <int, >
* Let: <let, >
* Print: <print, >
* Return: <return, >
* String: <string, >
* Switch: <switch, >
* Autodecremento: <opAutodecremento, >
* Constante entera: <entero, numero >
* Cadena: <cadena, lexema >
* Identificador: <id, numero >
* “=”: <opAsignacion, >
* “,”: <coma, >
* “;”: <puntoComa, >
* “:”: <dosPuntos, >
* “(“: <parentesisAbierto, >
* “)”: < parentesisCerrado, >
* “{“: <llaveAbierta, >
* “}”: <llaveCerrada,

## Autómata



## Gramática

l = letras minúsculas y mayúsculas

d = {0..9}

c = cualquier carácter

del = {espacio, eol, tab …}

a = c – { ”}

b = c – {eol}

o.c = otro caracter (diferente a letra y digito)

S 🡪 del S | l A | - C | d E | “ G | / K | = | , | ; | : | ( | ) | { | } | + | ! | >

A 🡪 l A | d A | \_ A | o.c

C 🡪 -

E 🡪 d E | o.c

G 🡪 a G | “

K 🡪 / L

L 🡪 b L | eol S

## Acciones semánticas

**0-0:** Leer.

**0-1:** Leer.

**1-2:** Leer; Gen\_Token(opAutodecremento, - ).

**0-3:** Numero = valor\_ASCII(d); Leer.

**3-3:** Numero = numero\*10 + valor\_ASCII(d); Leer.

**3-4:** Leer;IF(numero > 32767) THEN error(“Numero fuera de rango.”)

ELSE Gen\_Token(entero, numero).

**0-5:** Lexema := “;Leer

**5-5:** Lexema := concatenar(Lexema, a); Leer

**5-6:** Lexema := concatenar(Lexema, “); Leer; Gen\_Token(cadena, lexema)

**0-7:** Leer; Gen\_Token(opAsignacion, -)

**0-8:** Leer; Gen\_Token(coma, -)

**0-9:** Leer; Gen\_Token(puntoComa, -)

**0-10:** Leer; Gen\_Token(dosPuntos, -)

**0-11:** Leer; Gen\_Token(parentesisAbierto, -)

**0-12:** Leer; Gen\_Token(parentesisCerrado, -)

**0-13:** Leer; Gen\_Token(llaveAbierta, -)

**0-14:** Leer; Gen\_Token(llaveCerrada, -)

**0-15:** Leer; Gen\_Token(suma, -)

**0-16:** Leer; Gen\_Token(negacion, -)

**0-17:** Leer; Gen\_Token(mayor, -)

**0-18:** Leer;

**18-19:** Leer;

**19-19:** Leer;

**19-0:** Leer;

**0-20:** Lexema := l;Leer

**20-20:** Lexema := concatenar(Lexema, l/d,\_); Leer

**20-21:**

IF lexema == boolean THEN leer; Gen\_Token(Boolean, -)

ELSE IF lexema == break THEN leer; Gen\_Token(break, -)

ELSE IF lexema == case THEN leer; Gen\_Token(case, -)

ELSE IF lexema == function THEN leer; Gen\_Token(function, -)

ELSE IF lexema == if THEN leer; Gen\_Token(if, -)

ELSE IF lexema == input THEN leer; Gen\_Token(input, -)

ELSE IF lexema == int THEN leer; Gen\_Token(int, -)

ELSE IF lexema == print THEN leer; Gen\_Token(print, -)

ELSE IF lexema == return THEN leer; Gen\_Token(return, -)

ELSE IF lexema == string THEN leer; Gen\_Token(string, -)

ELSE IF lexema == switch THEN leer; Gen\_Token(switch, -)

ELSE IF Zona\_Declaracion = true THEN

{ p=buscar\_TS (lexema)

IF p=null THEN {p:=insertar\_TS(lexema) Gen\_Token (identificador, p) }

ELSE error (“identificador ya declarado”) }

ELSE { p=buscar\_TS (lexema)

IF p=null THEN error (“identificador NO declarado”)

ELSE Gen\_Token (identificador, p) }

## Error

Todas las transiciones no consideradas corresponden a casos de error.

# Tabla de símbolos

Para esta primera entrega, existirá solamente una tabla de símbolos (ya que es solo el analizador léxico el que está trabajando). Así que lo único que sabremos de esta tabla es el lexema.

# Analizador sintáctico

## Gramática

P -> B Z | F Z

B -> let T id; | if ( E ) S | switch ( Y ) { W } | S

T -> int | string | boolean

S -> idS’| print ( E ) ; | input ( id ); | return X;

S’ -> =E; | ( L ) ;

X -> E | lambda

C -> BC | lambda

L -> EQ | lambda

Q -> ,EQ | lambda

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

H -> T | lambda

A -> T id K | lambda

K -> ,T id K | lambda

E -> UE’

E’ -> >UE’ | lambda

U -> VU’

U’ -> +VU’ | lambda

V -> G | entero | cadena | --id | !G

G -> idV’ | ( E )

V’ -> ( L ) | lambda

Y -> id | entero

W -> case entero : C W | break ; W | lambda

Z -> P | lambda

## Condiciones LL(1)

Solo vamos a realizar el análisis de producciones que poseen más de 2 reglas, ya que las producciones que poseen solo 1 regla siempre cumplen esta condición.

* Producción P:

First(BZ)= {let, if, switch, id, print, input, return}

First(FZ) = {function}

* Producción B:

First(let T id; ) = {let }

First(if ( E ) S) ={if}

First(switch ( Y ) { W }) ={switch}

First(S) = {id, print, return, input}

* Producción T:

First(int)={ int }

First(string) ={ string }

First(boolean) = { boolean }

* Producción S:

First(idS’)= {id}

First(print ( E ) ; )= {print}

First(input ( id ); )= {input}

First(return X;) = { return}

* Producción S’:

First(=E;)= { = }

First( ( L ) ) = { ( }

* Producción X:

First(E)= {id, (, entero, cadena, --, !}

Follow( X ) = { ; }

* Producción C:

First(BC)= {let, if, switch, id, print, input, return }

Follow( C ) = { }, case, break }

* Producción L:

First(EQ)= {id, (, entero, cadena, --, !}

Follow( L ) = { ) }

* Producción Q:

First(,EQ)= { , }

Follow( Q ) = { ) }

* Producción H:

First(T)= { int, string, boolean }

Follow( H ) = { ( }

* Producción A:

First(T id K ) ={ int, string, boolean }

Follow( A ) = { ) }

* Producción K:

First(,T id K )= { , }

Follow( K ) = { ) }

* Producción E’:

First(>UE’ ) ={ > }

Follow( E’ ) = { , , ; ,) }

* Producción U’:

First(+VU’ )= { + }

Follow( U’ ) = {>, , , ; , ) }

* Producción V:

First(G) ={id, (}

First(entero)= {entero}

First(cadena)= {cadena }

First(--id)= { -- }

First(!G) = { ! }

* Producción G:

First(idV’)= { id }

First( ( E ) ) = { ( }

* Producción V’:

First( (L) )= { ( }

Follow( V’ ) = {+, >, , , ; , ) }

* Producción Y:

First(id ) ={ id}

First( entero ) = {entero }

* Producción W:

First(case entero : C W)= { case }

First(break ; W)= { break}

Follow(W) = { } }

* Producción Z:

First(P) ={ let, if, switch, id, print, input, return, function}

Follow(Z) = {$}

## Pseudo-Código

### Analizador Sintáctico

Function A\_Sint (){

sig\_tok=ALex();

P();

if (sig\_tok!="$") then error ()

}

### Función Equipara

Function equipara (t){

if (sig\_tok==t)

then sig\_tok=ALex();

else error();

}

### Producciones

Function P() {

if(sig\_tok=={let, if, switch, id, print, input, return}) {

B();

Z();

}

else if(sig\_tok==function) {

F();

Z();

}

else {

Error("Token no esperado");

}

}

Function B() {

if(sig\_tok==let) {

equipara(let);

T();

equipara(id);

}

else if(sig\_tok==f) {

equipara(if);

equipara(();

E();

equipara());

S();

}

else if(sig\_tok==switch){

equpiara(switch);

equipara(();

Y();

equipara());

equipara({);

W();

equipara(});

}

else if(sig\_tok= {id, print, input, return}) {

S();

}

else {

Error("Token no esperado");

}

}

Function T() {

if(sig\_tok==int) {

equipara(int);

}

else if(sig\_tok==String) {

equipara(String);

}

else if(sig\_tok==Boolean) {

equipara(Boolean);

}

else {

Error("Token no esperado");

}

}

Function S()  {

        if(sig\_tok=="id") {

             print(10);

            equipara("id");

            S'();

        }

        else if(sig\_tok=="print") {

             print(11);

            equipara("print");

            equipara("(");

            E();

            equipara(")");

            equipara(";");

        }

        else if(sig\_tok=="input") {

             print(12);

            equipara("input");

            equipara("(");

            equipara("id");

            equipara(")");

            equipara(";");

        }

        else if(sig\_tok=="return") {

             print(13);

            equipara("return");

            X();

            equipara(";");

        }

        else {

            error();

        }

    }

    Function S'()  {

        if(sig\_tok=="=") {

             print(14);

            equipara("=");

            E();

            equipara(";");

        }

        else if(sig\_tok=="(") {

             print(15);

            equipara("(");

            L();

            equipara(")");

            equipara(";");

        }

        else {

            error();

        }

    }

    Function X()  {

        if(sig\_tok=="!" || sig\_tok=="id" || sig\_tok=="(" || sig\_tok=="entero"|| sig\_tok=="cadena" || sig\_tok=="--") {

             print(16);

            E();

        }

        else if(sig\_tok==";") {

             print(17);

        }

        else {

            error();

        }

    }

    Function C()  {

        if(sig\_tok=="let" || sig\_tok=="if" || sig\_tok=="switch" || sig\_tok=="id" ||

                sig\_tok==("print") || sig\_tok==("input") || sig\_tok==("return")) {

             print(18);

            B();

            C();

        }

        else if(sig\_tok==("}") || sig\_tok==("case") || sig\_tok==("break")) {

             print(19);

        }

        else {

            error();

        }

    }

    Function L()  {

        if(sig\_tok=="id" || sig\_tok=="(" || sig\_tok=="entero" || sig\_tok=="cadena" || sig\_tok=="--") {

             print(20);

            E();

            Q();

        }

        else if(sig\_tok==")") {

             print(21);

        }

        else {

            error();

        }

    }

    Function Q()  {

        if(sig\_tok==",") {

             print(22);

            equipara(",");

            E();

            Q();

        }

        else if(sig\_tok==(")")) {

             print(23);

        }

        else {

            error();

        }

    }

    Function F()  {

        if(sig\_tok=="function") {

             print(24);

            equipara("function");

            equipara("id");

            H();

            equipara("(");

            A();

            equipara(")");

            equipara("{");

            C();

            equipara("}");

        }

        else {

            error();

        }

    }

    Function H()  {

        if(sig\_tok=="int" || sig\_tok=="string" || sig\_tok=="boolean") {

             print(25);

            T();

        }

        else if(sig\_tok=="(") {

             print(26);

        }

        else {

            error();

        }

    }

    Function A()  {

        if(sig\_tok=="int" || sig\_tok=="string" || sig\_tok=="boolean") {

             print(27);

            T();

            equipara("id");

            K();

        }

        else if(sig\_tok==")") {

             print(28);

        }

        else {

            error();

        }

    }

    Function K()  {

        if(sig\_tok==",") {

             print(29);

            equipara(",");

            T();

            equipara("id");

            K();

        }

        else if(sig\_tok==")") {

             print(30);

        }

        else {

            error();

        }

    }

    Function E()  {

        if(sig\_tok=="id" || sig\_tok=="("|| sig\_tok=="entero" || sig\_tok=="cadena"|| sig\_tok=="--"|| sig\_tok=="!") {

             print(31);

            U();

            E'();

        }

        else {

            error();

        }

    }

    Function E'()  {

        if(sig\_tok==">") {

             print(32);

            equipara(">");

            U();

            E'();

        }

        else if(sig\_tok==")" || sig\_tok==";" || sig\_tok==",") {

             print(33);

        }

        else {

            error();

        }

    }

    Function U()  {

        if(sig\_tok=="id" || sig\_tok=="(" || sig\_tok=="entero" || sig\_tok=="cadena"|| sig\_tok=="--" || sig\_tok=="!") {

             print(34);

            V();

            U'();

        }

        else {

            error();

        }

    }

    Function U'()  {

        if(sig\_tok=="+") {

             print(35);

            equipara("+");

            V();

            U'();

        }

        else if(sig\_tok==">" || sig\_tok==")" || sig\_tok==";" || sig\_tok==",") {

             print(36);

        }

        else {

            error();

        }

    }

    Function V()  {

        if(sig\_tok=="id") {

             print(37);

            equipara("id");

            V'();

        }

        else if(sig\_tok=="(") {

             print(38);

            equipara("(");

            E();

            equipara(")");

        }

        else if(sig\_tok==("entero")) {

             print(39);

            equipara("entero");

        }

        else if(sig\_tok==("cadena")) {

             print(40);

            equipara("cadena");

        }

        else if(sig\_tok=="--") {

             print(41);

            equipara("--");

            equipara("id");

        }

        else if(sig\_tok=="!") {

             print(42);

            equipara("!");

            E();

        }

        else {

            error();

        }

    }

    Function V'()  {

        if(sig\_tok=="(") {

             print(43);

            equipara("(");

            L();

            equipara(")");

        }

        else if(sig\_tok=="+" || sig\_tok==">" || sig\_tok==")" || sig\_tok==";" || sig\_tok==",") {

             print(44);

        }

        else {

            error();

        }

    }

    Function Y()  {

        if(sig\_tok=="id") {

             print(45);

            equipara("id");

        }

        else if(sig\_tok=="entero") {

             print(46);

            equipara("entero");

        }

        else {

            error();

        }

    }

    Function W()  {

        if(sig\_tok=="case") {

             print(47);

            equipara("case");

            equipara("entero");

            equipara(":");

            C();

            W();

        }

        else if(sig\_tok=="break") {

             print(48);

            equipara("break");

            equipara(";");

            W();

        }

        else if(sig\_tok==("}")) {

             print(49);

        }

        else {

            error();

        }

    }

Function Z()  {

if(sig\_tok=={let, if, switch, id, print, input, return, function}) {

             print(51);

            P();

        }

        else if(sig\_tok=="$") {

             print(52);

        }

        else {

            error();

        }

    }

### Función Error

    Function error() {

            print("Error, el token no es el esperado");

            sig\_tok="";

    }

# Casos de prueba Analizador léxico

Anexamos a la entrega los casos de prueba, de los cuales tres funcionan correctamente y tres devuelven error. De cada uno podremos ver el código que utilizamos, los tokens que produce y la tabla de símbolos.

## Caso 1 (correcto)

### Codigo:

let string cadena;

input(cadena);

let boolean logico1;

let boolean logico2;

let int int2;

int1 = 873;

int2 = 378;

if (! logico2) cadena = " hello";

function ff string(string sss)

{

global = 33;

logico1 = logico2;

if (logico1) sss = ff (cadena);

return sss;

}

function funcion string (string logico2)

{ let int var;

switch (int1){

case 0: logico1 = int1 > int2;break;

case 8888: print(0);

case 3333: logico2="";

}

return logico2;

}

print(((ff((funcion(cadena))))));

### Tokens:

<let , >

<string , >

<id , 0>

<puntoComa , >

<input , >

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<puntoComa , >

<let , >

<boolean , >

<id , 1>

<puntoComa , >

<let , >

<boolean , >

<id , 2>

<puntoComa , >

<let , >

<int , >

<id , 3>

<puntoComa , >

<id , 4>

<opAsignacion , >

<entero , 873>

<puntoComa , >

<id , 3>

<opAsignacion , >

<entero , 378>

<puntoComa , >

<if , >

<parentesisAbierto , >

<negacion , >

<id , 2>

<parentesisCerrado , >

<id , 0>

<opAsignacion , >

<cadena , " hello">

<puntoComa , >

<function , >

<id , 5>

<string , >

<parentesisAbierto , >

<string , >

<id , 6>

<parentesisCerrado , >

<llaveAbierta , >

<id , 7>

<opAsignacion , >

<entero , 33>

<puntoComa , >

<id , 1>

<opAsignacion , >

<id , 2>

<puntoComa , >

<if , >

<parentesisAbierto , >

<id , 1>

<parentesisCerrado , >

<id , 6>

<opAsignacion , >

<id , 5>

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<puntoComa , >

<return , >

<id , 6>

<puntoComa , >

<llaveCerrada , >

<function , >

<id , 8>

<string , >

<parentesisAbierto , >

<string , >

<id , 2>

<parentesisCerrado , >

<llaveAbierta , >

<let , >

<int , >

<id , 9>

<puntoComa , >

<switch , >

<parentesisAbierto , >

<id , 4>

<parentesisCerrado , >

<llaveAbierta , >

<case , >

<entero , 0>

<dosPuntos , >

<id , 1>

<opAsignacion , >

<id , 4>

<mayor , >

<id , 3>

<puntoComa , >

<break , >

<puntoComa , >

<case , >

<entero , 8888>

<dosPuntos , >

<print , >

<parentesisAbierto , >

<entero , 0>

<parentesisCerrado , >

<puntoComa , >

<case , >

<entero , 3333>

<dosPuntos , >

<id , 2>

<opAsignacion , >

<cadena , "">

<puntoComa , >

<llaveCerrada , >

<return , >

<id , 2>

<puntoComa , >

<llaveCerrada , >

<print , >

<parentesisAbierto , >

<parentesisAbierto , >

<parentesisAbierto , >

<id , 5>

<parentesisAbierto , >

<parentesisAbierto , >

<id , 8>

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<puntoComa , >

### Tabla de símbolos

CONTENIDOS DE LA TABLA #0 :

\* lexema: 'ff'

Atributos:

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

\* lexema: 'int2'

Atributos:

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

\* lexema: 'sss'

Atributos:

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

\* lexema: 'int1'

Atributos:

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

\* lexema: 'logico2'

Atributos:

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

\* lexema: 'var'

Atributos:

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

\* lexema: 'cadena'

Atributos:

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

\* lexema: 'logico1'

Atributos:

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

\* lexema: 'global'

Atributos:

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

\* lexema: 'funcion'

Atributos:

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

## Caso 2 (correcto)

### Código

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\_2;

xx = f1+x;

print//Alerta

(ss);

return xx;

}

function f2 boolean( int f2 , boolean b1 )

{

input (y);

print ((4+5+77+(088+f2)));

return (!!b1);

}

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

print (f1 (x, f2 (3, boolean\_2)));

### Tokens

<let , >

<int , >

<id , 0>

<puntoComa , >

<let , >

<boolean , >

<id , 1>

<puntoComa , >

<let , >

<int , >

<id , 2>

<puntoComa , >

<let , >

<string , >

<id , 3>

<puntoComa , >

<let , >

<int , >

<id , 4>

<puntoComa , >

<let , >

<boolean , >

<id , 5>

<puntoComa , >

<function , >

<id , 6>

<int , >

<parentesisAbierto , >

<int , >

<id , 6>

<coma , >

<boolean , >

<id , 7>

<parentesisCerrado , >

<llaveAbierta , >

<input , >

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<puntoComa , >

<id , 1>

<opAsignacion , >

<negacion , >

<id , 5>

<puntoComa , >

<id , 4>

<opAsignacion , >

<id , 6>

<suma , >

<id , 2>

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 3>

<parentesisCerrado , >

<puntoComa , >

<return , >

<id , 4>

<puntoComa , >

<llaveCerrada , >

<function , >

<id , 8>

<boolean , >

<parentesisAbierto , >

<int , >

<id , 8>

<coma , >

<boolean , >

<id , 7>

<parentesisCerrado , >

<llaveAbierta , >

<input , >

<parentesisAbierto , >

<id , 9>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<parentesisAbierto , >

<entero , 4>

<suma , >

<entero , 5>

<suma , >

<entero , 77>

<suma , >

<parentesisAbierto , >

<entero , 088>

<suma , >

<id , 8>

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<puntoComa , >

<return , >

<parentesisAbierto , >

<negacion , >

<negacion , >

<id , 7>

<parentesisCerrado , >

<puntoComa , >

<llaveCerrada , >

<id , 2>

<opAsignacion , >

<id , 2>

<suma , >

<entero , 6>

<suma , >

<id , 0>

<suma , >

<entero , 1>

<suma , >

<parentesisAbierto , >

<entero , 2>

<suma , >

<id , 9>

<suma , >

<entero , 6>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 6>

<parentesisAbierto , >

<id , 2>

<coma , >

<id , 8>

<parentesisAbierto , >

<entero , 3>

<coma , >

<id , 5>

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<puntoComa , >

### Tabla de símbolos

CONTENIDOS DE LA TABLA #0 :

\* lexema: 'ss'

Atributos:

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

\* lexema: 'xx'

Atributos:

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

\* lexema: 'x'

Atributos:

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

\* lexema: 'y'

Atributos:

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

\* lexema: 'z'

Atributos:

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

\* lexema: 'boolean\_2'

Atributos:

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

\* lexema: 'boolean\_1'

Atributos:

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

\* lexema: 'f1'

Atributos:

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

\* lexema: 'f2'

Atributos:

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

\* lexema: 'b1'

Atributos:

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

## Caso 3 (correcto)

### Código

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 num\_1, int num\_2)

{

return num\_1 + num\_2+77;

}

number = 0;

print(operacion(b,a));

}

function f2 boolean( int f2 , boolean b1 )

{

input (y);

print ((4+5+77+(088+f2)));

return (!!b1);

}

z=

x + 6 + z + 1+ (2 + 6);

let 30;

print (f5 (x, f2 (3, boolean\_2)));

### Tokens

<let , >

<int , >

<id , 0>

<puntoComa , >

<let , >

<int , >

<id , 1>

<puntoComa , >

<let , >

<int , >

<id , 2>

<puntoComa , >

<print , >

<parentesisAbierto , >

<cadena , "Introduce el primer operando">

<parentesisCerrado , >

<puntoComa , >

<input , >

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<cadena , "Introduce el segundo operando">

<parentesisCerrado , >

<puntoComa , >

<input , >

<parentesisAbierto , >

<id , 1>

<parentesisCerrado , >

<puntoComa , >

<function , >

<id , 3>

<int , >

<parentesisAbierto , >

<int , >

<id , 4>

<coma , >

<int , >

<id , 5>

<parentesisCerrado , >

<llaveAbierta , >

<return , >

<id , 4>

<suma , >

<id , 5>

<suma , >

<entero , 77>

<puntoComa , >

<llaveCerrada , >

<id , 2>

<opAsignacion , >

<entero , 0>

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 3>

<parentesisAbierto , >

<id , 1>

<coma , >

<id , 0>

<parentesisCerrado , >

<parentesisCerrado , >

<puntoComa , >

<llaveCerrada , >

<function , >

<id , 6>

<boolean , >

<parentesisAbierto , >

<int , >

<id , 6>

<coma , >

<boolean , >

<id , 7>

<parentesisCerrado , >

<llaveAbierta , >

<input , >

<parentesisAbierto , >

<id , 8>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<parentesisAbierto , >

<entero , 4>

<suma , >

<entero , 5>

<suma , >

<entero , 77>

<suma , >

<parentesisAbierto , >

<entero , 088>

<suma , >

<id , 6>

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<puntoComa , >

<return , >

<parentesisAbierto , >

<negacion , >

<negacion , >

<id , 7>

<parentesisCerrado , >

<puntoComa , >

<llaveCerrada , >

<id , 9>

<opAsignacion , >

<id , 10>

<suma , >

<entero , 6>

<suma , >

<id , 9>

<suma , >

<entero , 1>

<suma , >

<parentesisAbierto , >

<entero , 2>

<suma , >

<entero , 6>

<parentesisCerrado , >

<puntoComa , >

<let , >

<entero , 30>

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 11>

<parentesisAbierto , >

<id , 10>

<coma , >

<id , 6>

<parentesisAbierto , >

<entero , 3>

<coma , >

<id , 12>

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<puntoComa , >

### Tabla de símbolos

CONTENIDOS DE LA TABLA #0 :

\* lexema: 'a'

Atributos:

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

\* lexema: 'b'

Atributos:

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

\* lexema: 'boolean\_2'

Atributos:

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

\* lexema: 'operacion'

Atributos:

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

\* lexema: 'f2'

Atributos:

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

\* lexema: 'b1'

Atributos:

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

\* lexema: 'f5'

Atributos:

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

\* lexema: 'number'

Atributos:

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

\* lexema: 'x'

Atributos:

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

\* lexema: 'y'

Atributos:

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

\* lexema: 'z'

Atributos:

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

\* lexema: 'num\_1'

Atributos:

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

\* lexema: 'num\_2'

Atributos:

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

## Caso 4 (error)

### Código

// Ejemplo de caso de error

let int66 = 32768;

let string cadena="Las constantes cadena van encerradas entre comillas dobles o entre comillas simples";

input(str);

let boolean bool1;

let boolean logico2;

let int int2;

int1 = 32;

int2 = 5;

if (! logico2 && bool1) cadena = " analizador";

function ff string(string sss)

{

global = 33/5;

logico1 = logico2;

logico1;

if (logico1 || logico2) sss = ff (cadena);

return sss;

}

function funcion string (string logico2)

{ let int var;

switch (int1){

case 0: logico1 = int1 < int2;break;

case 8888: print(0);

case 3333: logico2="";

}

return logico2;

}

### Tokens

<let , >

<id , 0>

<opAsignacion , >

ERROR: Entero 32768 esta fuera de rango

<puntoComa , >

<let , >

<string , >

<id , 1>

<opAsignacion , >

ERROR: Cadena "Las constantes cadena van encerradas entre comillas dobles o entre comillas simples" esta fuera de rango

<puntoComa , >

<input , >

<parentesisAbierto , >

<id , 2>

<parentesisCerrado , >

<puntoComa , >

<let , >

<boolean , >

<id , 3>

<puntoComa , >

<let , >

<boolean , >

<id , 4>

<puntoComa , >

<let , >

<int , >

<id , 5>

<puntoComa , >

<id , 6>

<opAsignacion , >

<entero , 32>

<puntoComa , >

<id , 5>

<opAsignacion , >

<entero , 5>

<puntoComa , >

<if , >

<parentesisAbierto , >

<negacion , >

<id , 4>

ERROR: Simbolo & no definido

ERROR: Simbolo & no definido

<id , 3>

<parentesisCerrado , >

<id , 1>

<opAsignacion , >

<cadena , " analizador">

<puntoComa , >

<function , >

<id , 7>

<string , >

<parentesisAbierto , >

<string , >

<id , 8>

<parentesisCerrado , >

<llaveAbierta , >

<id , 9>

<opAsignacion , >

<entero , 33>

ERROR: Simbolo / no definido

<entero , 5>

<puntoComa , >

<id , 10>

<opAsignacion , >

<id , 4>

<puntoComa , >

<id , 10>

<opAsignacion , >

<puntoComa , >

<if , >

<parentesisAbierto , >

<id , 10>

ERROR: Simbolo | no definido

ERROR: Simbolo | no definido

<id , 4>

<parentesisCerrado , >

<id , 8>

<opAsignacion , >

<id , 7>

<parentesisAbierto , >

<id , 1>

<parentesisCerrado , >

<puntoComa , >

<return , >

<id , 8>

<puntoComa , >

<llaveCerrada , >

<function , >

<id , 12>

<string , >

<parentesisAbierto , >

<string , >

<id , 4>

<parentesisCerrado , >

<llaveAbierta , >

<let , >

<int , >

<id , 13>

<puntoComa , >

<switch , >

<parentesisAbierto , >

<id , 6>

<parentesisCerrado , >

<llaveAbierta , >

<case , >

<entero , 0>

<dosPuntos , >

<id , 10>

<opAsignacion , >

<id , 6>

ERROR: Simbolo < no definido

<id , 5>

<puntoComa , >

<break , >

<puntoComa , >

<case , >

<entero , 8888>

<dosPuntos , >

<print , >

<parentesisAbierto , >

<entero , 0>

<parentesisCerrado , >

<puntoComa , >

<case , >

<entero , 3333>

<dosPuntos , >

<id , 4>

<opAsignacion , >

<cadena , "">

<puntoComa , >

<llaveCerrada , >

<return , >

<id , 4>

<puntoComa , >

<llaveCerrada , >

### Tabla de símbolos

CONTENIDOS DE LA TABLA #0 :

\* lexema: 'ff'

Atributos:

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

\* lexema: 'logico2'

Atributos:

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

\* lexema: 'var'

Atributos:

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

\* lexema: 'cadena'

Atributos:

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

\* lexema: 'logico1'

Atributos:

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

\* lexema: 'global'

Atributos:

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

\* lexema: 'str'

Atributos:

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

\* lexema: 'int2'

Atributos:

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

\* lexema: 'sss'

Atributos:

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

\* lexema: 'int1'

Atributos:

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

\* lexema: 'int66'

Atributos:

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

\* lexema: 'true'

Atributos:

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

\* lexema: 'funcion'

Atributos:

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

\* lexema: 'bool1'

Atributos:

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

## Caso 5 (error)

### Código

let int z=4%2;

function f1 int(int z)

{

input (x);

let string = "cadena superior a los caracteres permitidos por javaScript";

x = 5624642;

print(z)//Imprimimos z

}

let string ss;

let int xx;

let boolean boolean\_2;

function funcionF2 boolean(int f1, boolean b1)

{

input (z);

boolean\_1 = ! boolean\_2;

xx = f1+x;

print//Alerta

(ss);

return xx;

}

function funcionF2 string( int f2 , boolean b1 )

{

input (y);

print ((4+5+77+(088+f2)));

return (b1<b2);

}

x =x + 6458554+ z \* 1/(2+ aux\* 6);

print (x);

### Tokens

<let , >

<int , >

<id , 0>

<opAsignacion , >

<entero , 4>

ERROR: Simbolo % no definido

<entero , 2>

<puntoComa , >

<function , >

<id , 1>

<int , >

<parentesisAbierto , >

<int , >

<id , 0>

<parentesisCerrado , >

<llaveAbierta , >

<input , >

<parentesisAbierto , >

<id , 2>

<parentesisCerrado , >

<puntoComa , >

<let , >

<string , >

<opAsignacion , >

<cadena , "cadena superior a los caracteres permitidos por javaScript">

<puntoComa , >

<id , 2>

<opAsignacion , >

ERROR: Entero 5624642 esta fuera de rango

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<llaveCerrada , >

<let , >

<string , >

<id , 3>

<puntoComa , >

<let , >

<int , >

<id , 4>

<puntoComa , >

<let , >

<boolean , >

<id , 5>

<puntoComa , >

<function , >

<id , 6>

<boolean , >

<parentesisAbierto , >

<int , >

<id , 1>

<coma , >

<boolean , >

<id , 7>

<parentesisCerrado , >

<llaveAbierta , >

<input , >

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<puntoComa , >

<id , 8>

<opAsignacion , >

<negacion , >

<id , 5>

<puntoComa , >

<id , 4>

<opAsignacion , >

<id , 1>

<suma , >

<id , 2>

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 3>

<parentesisCerrado , >

<puntoComa , >

<return , >

<id , 4>

<puntoComa , >

<llaveCerrada , >

<function , >

<id , 6>

<string , >

<parentesisAbierto , >

<int , >

<id , 9>

<coma , >

<boolean , >

<id , 7>

<parentesisCerrado , >

<llaveAbierta , >

<input , >

<parentesisAbierto , >

<id , 10>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<parentesisAbierto , >

<entero , 4>

<suma , >

<entero , 5>

<suma , >

<entero , 77>

<suma , >

<parentesisAbierto , >

<entero , 088>

<suma , >

<id , 9>

<parentesisCerrado , >

<parentesisCerrado , >

<parentesisCerrado , >

<puntoComa , >

<return , >

<parentesisAbierto , >

<id , 7>

ERROR: Simbolo < no definido

<id , 11>

<parentesisCerrado , >

<puntoComa , >

<llaveCerrada , >

<id , 2>

<opAsignacion , >

<id , 2>

<suma , >

ERROR: Entero 6458554 esta fuera de rango

<suma , >

<id , 0>

ERROR: Simbolo \* no definido

<entero , 1>

ERROR: Simbolo / no definido

<parentesisAbierto , >

<entero , 2>

<suma , >

<id , 12>

ERROR: Simbolo \* no definido

<entero , 6>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 2>

<parentesisCerrado , >

<puntoComa , >

### Tabla de símbolos

CONTENIDOS DE LA TABLA #0 :

\* lexema: 'ss'

Atributos:

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

\* lexema: 'xx'

Atributos:

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

\* lexema: 'aux'

Atributos:

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

\* lexema: 'boolean\_2'

Atributos:

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

\* lexema: 'f1'

Atributos:

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

\* lexema: 'boolean\_1'

Atributos:

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

\* lexema: 'f2'

Atributos:

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

\* lexema: 'b1'

Atributos:

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

\* lexema: 'b2'

Atributos:

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

\* lexema: 'funcionF2'

Atributos:

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

\* lexema: 'x'

Atributos:

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

\* lexema: 'y'

Atributos:

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

\* lexema: 'z'

Atributos:

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

## Caso 6 (error)

### Código

let int x;

let boolean b;

let int z= 55346436;

input (x);

print (x);

input (z);

print (x+z);

print("cadena de caracteres superior al permitido por el lenguaje javaScript");

b=x<z;if (b||z)

x =x - 6\* z+ 1+ [(2%y)/7)];

let int n1;

let boolean l1;

let string cad;

let boolean l2;

input (n1);

l1 = l2;

if (! l2) cad = "hello";

n2 = n1 + 378;

print(6534756);

function ff boolean(boolean ss)

{

l2 = ! l1 && l2;

if (l2) l1 = ff (ss);

varglobal =7567844;

return (ss);

}

if (ff(l1))

print (varglobal);

### Tokens

<let , >

<int , >

<id , 0>

<puntoComa , >

<let , >

<boolean , >

<id , 1>

<puntoComa , >

<let , >

<int , >

<id , 2>

<opAsignacion , >

ERROR: Entero 55346436 esta fuera de rango

<puntoComa , >

<input , >

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<puntoComa , >

<input , >

<parentesisAbierto , >

<id , 2>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 0>

<suma , >

<id , 2>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

ERROR: Cadena "cadena de caracteres superior al permitido por el lenguaje javaScript" esta fuera de rango

<parentesisCerrado , >

<puntoComa , >

<id , 1>

<opAsignacion , >

<id , 0>

ERROR: Simbolo < no definido

<id , 2>

<puntoComa , >

<if , >

<parentesisAbierto , >

<id , 1>

ERROR: Simbolo | no definido

ERROR: Simbolo | no definido

<id , 2>

<parentesisCerrado , >

<id , 0>

<opAsignacion , >

<id , 0>

ERROR: Simbolo - no definido

<entero , 6>

ERROR: Simbolo \* no definido

<id , 2>

<suma , >

<entero , 1>

<suma , >

ERROR: Simbolo [ no definido

<parentesisAbierto , >

<entero , 2>

ERROR: Simbolo % no definido

<id , 3>

<parentesisCerrado , >

ERROR: Simbolo / no definido

<entero , 7>

<parentesisCerrado , >

ERROR: Simbolo ] no definido

<puntoComa , >

<let , >

<int , >

<id , 4>

<puntoComa , >

<let , >

<boolean , >

<id , 5>

<puntoComa , >

<let , >

<string , >

<id , 6>

<puntoComa , >

<let , >

<boolean , >

<id , 7>

<puntoComa , >

<input , >

<parentesisAbierto , >

<id , 4>

<parentesisCerrado , >

<puntoComa , >

<id , 5>

<opAsignacion , >

<id , 7>

<puntoComa , >

<if , >

<parentesisAbierto , >

<negacion , >

<id , 7>

<parentesisCerrado , >

<id , 6>

<opAsignacion , >

<cadena , "hello">

<puntoComa , >

<id , 8>

<opAsignacion , >

<id , 4>

<suma , >

<entero , 378>

<puntoComa , >

<print , >

<parentesisAbierto , >

ERROR: Entero 6534756 esta fuera de rango

<parentesisCerrado , >

<puntoComa , >

<function , >

<id , 9>

<boolean , >

<parentesisAbierto , >

<boolean , >

<id , 10>

<parentesisCerrado , >

<llaveAbierta , >

<id , 7>

<opAsignacion , >

<negacion , >

<id , 5>

ERROR: Simbolo & no definido

ERROR: Simbolo & no definido

<id , 7>

<puntoComa , >

<if , >

<parentesisAbierto , >

<id , 7>

<parentesisCerrado , >

<id , 5>

<opAsignacion , >

<id , 9>

<parentesisAbierto , >

<id , 10>

<parentesisCerrado , >

<puntoComa , >

<id , 11>

<opAsignacion , >

ERROR: Entero 7567844 esta fuera de rango

<puntoComa , >

<return , >

<parentesisAbierto , >

<id , 10>

<parentesisCerrado , >

<puntoComa , >

<llaveCerrada , >

<if , >

<parentesisAbierto , >

<id , 9>

<parentesisAbierto , >

<id , 5>

<parentesisCerrado , >

<parentesisCerrado , >

<print , >

<parentesisAbierto , >

<id , 11>

<parentesisCerrado , >

<puntoComa , >

### Tabla de símbolos

CONTENIDOS DE LA TABLA #0 :

\* lexema: 'ff'

Atributos:

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

\* lexema: 'ss'

Atributos:

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

\* lexema: 'b'

Atributos:

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

\* lexema: 'n1'

Atributos:

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

\* lexema: 'n2'

Atributos:

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

\* lexema: 'l1'

Atributos:

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

\* lexema: 'l2'

Atributos:

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

\* lexema: 'varglobal'

Atributos:

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

\* lexema: 'cad'

Atributos:

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

\* lexema: 'x'

Atributos:

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

\* lexema: 'y'

Atributos:

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

\* lexema: 'z'

Atributos:

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

# Casos de prueba Analizador sintáctico

## Caso 1