**GRUPO 63** 

# Procesador de Lenguajes: JS--

Procesadores de Lenguajes

Ignacio García Fernández José Ruiz Esteban Eduardo Gil Alba 4-10-2024

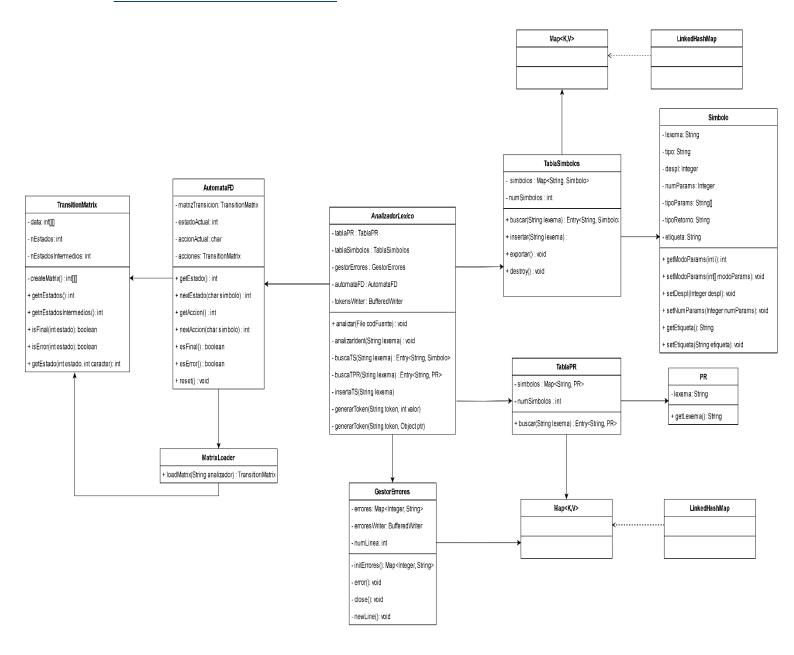
# Tabla de contenido

1.	Prop	ósito	1
2.	Dise	ño de estructura	1
3.	Anal	izador Léxico	2
3	.1.	Tokens	2
3	.2.	Gramática	3
3	.3.	Autómata	4
	3.3.1	Acciones Semánticas	5
	3.3.2	2. Errores	6
4.	Tabla	a de Símbolos	7
4	.1.	Estructura y organización	7
AN	EXO		8
C	Casos	de prueba	8
	Caso	o de prueba 1	8
	Caso	o de prueba 2	9
	Caso	o de prueba 3 1	0
	Caso	o de Prueba 4 1	1
	Caso	o de prueba 5 1	2
	Caso	o de prueba 6 1	3

# 1. Propósito

En este documento se detalla un primer diseño que se ha empleado en construir un analizador léxico para el lenguaje Javascript JS--. Para ello, se ha decidido utilizar como lenguaje de soporte Java, implementando un diseño orientado a objetos. El diagrama de clases se detalla en el siguiente apartado.

# 2. Diseño de estructura



# 3. Analizador Léxico

El analizador léxico es el encargado de comprobar si una expresión forma parte del lenguaje.

#### 3.1. Tokens

Son cada uno de los elementos del lenguaje con significado propio. Estos son los tokens que se ha decidido que pueda generar el analizador léxico.

<pr, postpr=""> Palabra reservada</pr,>	<cadena, lexema="">  Cadenas de caracteres</cadena,>	<id, posts=""></id,>
<entero, valor=""> Constantes enteras</entero,>	<suma, -=""> + Operador aritmético</suma,>	<asig, -=""> =  Operador asignación</asig,>
< ASIGAND , -> &=  Operador y lógico	<pre><dobleeq,-> == Operador relacional</dobleeq,-></pre>	<pre><dobleand, -=""> &amp;&amp;    Operador lógico</dobleand,></pre>
< PARENTIZDA , -> ( Apertura paréntesis	< PARENTDCHA , -> )  Cierre paréntesis	<pre><llaveizda, -=""> {    Apertura bloque</llaveizda,></pre>
< LLAVEDCHA , -> } Cierre bloque	<pyc, -=""> ; Punto y coma</pyc,>	<coma, -=""> , Coma</coma,>
	<eof, -=""></eof,>	
	Fin de fichero	

En principio, se ha decidido implementar un operador por cada tipo: aritmético, la suma (+); relacional, la equivalencia (==); lógico y la operación AND (&&). Además, nuestro grupo debe analizar el operador de asignación lógico, &=. Las palabras reservadas estarán indexadas en una tabla hash y al generar el token se devuelve un puntero a la posición en la Tabla de Palabras Reservadas. De igual manera, con los identificadores. Los tokens que se visualizan en la tabla, aparecerán de esa misma forma en el fichero generado 'tokens'. De momento, el analizador vuelca los tokens en un fichero y se deja planteado un objeto Token que pueda ser proporcionado al analizador sintáctico, más adelante.

#### 3.2. Gramática

```
T: Cjto. de terminales

l: letra d: digito c<sub>1</sub>:T-{'} c<sub>2</sub>:T-{*} c<sub>3</sub>:T-{/,*}

cr: salto de linea del: {esp, tab} eof: end of file

0: A → lp|d N|'ST|& Y|=I|/C|,|;|(|)|{|}|+|eof|del A|cr A

1: P → lp|d P|_P| λ

2: N → d N| λ

3: ST → c<sub>1</sub>ST| '

4: Y → &|=

5: I → =|λ

6: C → *D

7: D → c<sub>2</sub>D|*E

8: E → /A|c<sub>3</sub>D|*E
```

La gramática que emplea el analizador léxico es una gramática regular, de tipo 3 según establece la jerarquía de Chomsky. Con esta gramática se establece cuáles son las expresiones que aceptará nuestro analizador léxico, es decir, que forman parte de nuestro lenguaje. Concretamente, abarcará las características generales de JS - - , aquellas obligatorias de implementación para nuestro grupo y las que hemos decidido implementar.

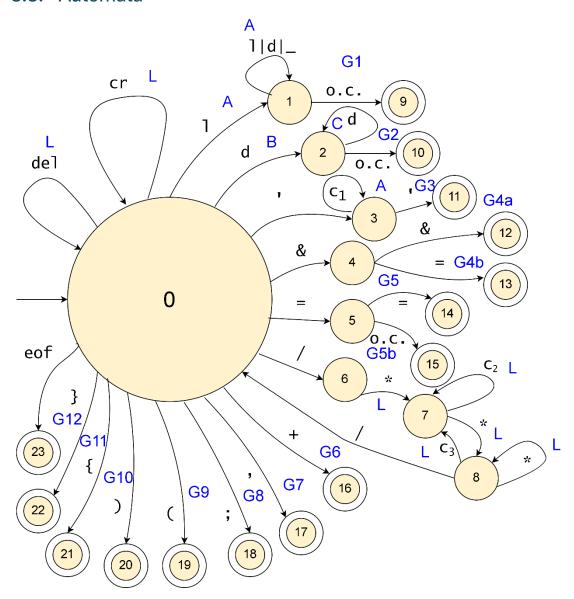
El comportamiento del analizador léxico, va determinado por el autómata generado por esta grámatica (siguiente apartado). La columna de la izquierda indica el estado del autómata. El axioma A corresponde al estado 0 o inicial.

Se puede observar que: la trasición al estado 1, corresponde al procesado de identificadores válidos para funciones y variables; la transición al estado 2, al procesado de números enteros; la transción al estado 3, al procesado de los caracteres que conformarán una cadena; la transicion al estado 4, comprueba si se trata del operador lógico o el operador de asignación lógico; la transición al estado 5, comprueba si se trata del operador de asignación o el operador relacional; las transiciones 6, 7 y 8, al procesado de los comentarios.

En los estados finales, nuestro analizador generará los tokens. También se realizan las comprobación de las restricciones. Los comentarios y delimitadores no conducen a estados finales, por lo que, no generarán token.

Cualquier transición no prevista, se considera un error, se informa al Gestor de Errores que presentará un mensaje descriptivo al usuario.

#### 3.3. Autómata



El automata generado por la gramática es finito determinista (AFD), esto quiere decir que, para una misma entrada, siempre se produce el mismo resultado. La notación en color azul, denota las acciones semánticas que se realizan en las transiciones.

En nuestra implementación, existe una clase AutomataFD, que posee el Analizador Lexico, contiene toda la información del automata: el estado actual, estados finales, dos matrices de transición, TransitionMatrix. Cada una de las matrices se cargan de un fichero, que contiene una cabecera seguido de los elementos de la matriz tabulados. Las columnas de la matriz que corresponden a los caracteres de entrada, van numeradas desde el 0 al 127 ( codificación ASCII). Una de las matrices determina el estado siguiente y la otra la acción siguiente.

Esta clase contiene métodos para conocer si se encuentra en un estado final o en un estado de error.

#### 3.3.1. Acciones Semánticas

Nótese que se han descrito las acciones semánticas en las transiciones finales como G\*, ya que efectúan la generación de token, entre otras acicones. En el código, por comodidad se han asignado las letras del alfabeto a partir de la G, exceptúando la letra L que está reservada. Se ha tomado la decisión de que si se encuentra un error, el analizador léxico reporta el error y continua leyendo el siguiente carácter.

Acción   en el código		Acción			
A	Acción	en el	Descripción		
A		código			
LEE	Λ	۸	CONCAT		
C	A	A	LEE		
L   L   LEE	В	В	num := valor(d)		
D	С	С	num := num*10 + valor(d)		
G1   G	L	L	LEE		
G1			p := BUSCATPR(lex)		
G2			if (p != NULL) GENTOKEN(PR, p)		
GENTOKEN(ID, p)   if (num > 32767)	G1	G	p := BUSCATS(lex)		
G2			if $(p == NULL)$ $p := INSERTATS(lex)$		
G2			GENTOKEN(ID, p)		
GENTOKEN(ENTERO, num)   GENTOKEN(ENTERO, num)   GENTOKEN(ENTERO, num)   GENTOKEN(ENTERO, num)   GENTOKEN(CADENA, lex)   GENTOKEN(CADENA, lex)   GENTOKEN(DOBLEAND, -)   LEE   GENTOKEN(BOBLEAND, -)   LEE   GENTOKEN(BOBLEAND, -)   LEE   GENTOKEN(DOBLEEQ, -)   LEE   GENTOKEN(DOBLEEQ, -)   LEE   GENTOKEN(ASIG, -)   LEE   GENTOKEN(SUMA, -)   LEE   GENTOKEN(COMA, -)   LEE   GENTOKEN(PYC, -)   LEE   GENTOKEN(PYC, -)   LEE   GENTOKEN(PYC, -)   LEE   GENTOKEN(PYC, -)   LEE   GENTOKEN(PARENTIZDA, -)   LEE   GENTOKEN(PARENTOCHA, -)   LEE   GENTOKEN(LLAVEIZDA, -)   LEE   GENTOKEN(LAVEIZDA, -)   LEE   GENTOKEN(LAVEIZDA, -)   LEE			if (num > 32767)		
GENTOKEN(ENTERO, num)   GENTOKEN(ENTERO, num)   if (longitud(lex) > 64)   { ERROR(37, lex) }     LEE	C2		{ ERROR(36, num)		
I	G2	П	LEE }		
G3			GENTOKEN(ENTERO, num)		
Gab   Gentoken(cadena, lex)   Gentoken(cadena, lex)   Gentoken(dobleand, -)   Lee   Gentoken(dobleand, -)   Lee   Gentoken(asigand, -)   Lee   Gentoken(dobleeq, -)   Lee   Gentoken(dobleeq, -)   Lee   Gentoken(asig, -)   Lee   Gentoken(asig, -)   Lee   Gentoken(suma, -)   Lee   Gentoken(coma, -)   Lee   Gentoken(coma, -)   Lee   Gentoken(pyc, -)   Lee   Gentoken(pyc, -)   Lee   Gentoken(pyc, -)   Lee   Gentoken(parentizda, -)   Lee   Gentok			if (longitud(lex) > 64)		
GENTOKEN(CADENA, lex)   GENTOKEN(CADENA, lex)   GENTOKEN(DOBLEAND, -)   LEE   GENTOKEN(ASIGAND, -)   LEE   GENTOKEN(DOBLEEQ, -)   LEE   GENTOKEN(DOBLEEQ, -)   LEE   GENTOKEN(ASIG, -)   LEE   GENTOKEN(SUMA, -)   LEE   GENTOKEN(COMA, -)   LEE   GENTOKEN(PYC, -)   LEE   GENTOKEN(PYC, -)   LEE   GENTOKEN(PYC, -)   LEE   GENTOKEN(PARENTIZDA, -)   LEE   GENTOKEN(PARENTIZDA, -)   LEE   GENTOKEN(PARENTICOHA, -)   LEE   GENTOKEN(LLAVEIZDA, -)   LEE   L	02		{ ERROR(37, lex)		
G4a         J         GENTOKEN(DOBLEAND, -)             LEE           G4b         K         GENTOKEN(ASIGAND, -)             LEE           G5a         M         GENTOKEN(DOBLEEQ, -)             LEE           G5b         N         GENTOKEN(ASIG, -)             LEE           G6         O         GENTOKEN(SUMA, -)             LEE           G7         P         GENTOKEN(COMA, -)             LEE           G8         Q         GENTOKEN(PYC, -)             LEE           G9         R         GENTOKEN(PARENTIZDA, -)             LEE           G10         S         GENTOKEN(PARENTIDCHA, -)             LEE           G11         T         GENTOKEN(LLAVEIZDA, -)             LEE           G11         T         GENTOKEN(LLAVEIZDA, -)              LEE	G3	l	LEE }		
G48			GENTOKEN(CADENA, lex)		
G4b   K   GENTOKEN(ASIGAND, -)   LEE	0.4=	a J	GENTOKEN(DOBLEAND, -)		
G50	G4a		LEE		
G50	0.415	К	GENTOKEN(ASIGAND, -)		
G5a   M	G40				
G5b	05-	.   <sub>M</sub>	GENTOKEN(DOBLEEQ, -)		
G50 N LEE  G6 O GENTOKEN(SUMA, -) LEE  G7 P GENTOKEN(COMA, -) LEE  G8 Q GENTOKEN(PYC, -) LEE  G9 R GENTOKEN(PARENTIZDA, -) LEE  G10 S GENTOKEN(PARENTDCHA, -) LEE  G11 T GENTOKEN(LLAVEIZDA, -) LEE  G12 U GENTOKEN(LLAVEDCHA, -)	Goa	I <sup>V</sup> I	LEE		
G6 O GENTOKEN(SUMA, -)  LEE  G7 P GENTOKEN(COMA, -)  LEE  G8 Q GENTOKEN(PYC, -)  LEE  G9 R GENTOKEN(PARENTIZDA, -)  LEE  G10 S GENTOKEN(PARENTDCHA, -)  LEE  G11 T GENTOKEN(LLAVEIZDA, -)  LEE  G12 U GENTOKEN(LLAVEDCHA, -)	OFh	N.	GENTOKEN(ASIG, -)		
GF D LEE  GF P GENTOKEN(COMA, -)  LEE  GENTOKEN(PYC, -)  LEE  GENTOKEN(PARENTIZDA, -)  LEE  GENTOKEN(PARENTDCHA, -)  LEE  GENTOKEN(LLAVEIZDA, -)  LEE  GENTOKEN(LLAVEIZDA, -)  LEE  GENTOKEN(LLAVEDCHA, -)	Gob	IN	LEE		
G7 P GENTOKEN(COMA, -) LEE  G8 Q GENTOKEN(PYC, -) LEE  G9 R GENTOKEN(PARENTIZDA, -) LEE  G10 S GENTOKEN(PARENTDCHA, -) LEE  G11 T GENTOKEN(LLAVEIZDA, -) LEE  G12 U GENTOKEN(LLAVEDCHA, -)	00	0	GENTOKEN(SUMA, -)		
G7 P LEE  G8 Q GENTOKEN(PYC, -) LEE  G9 R GENTOKEN(PARENTIZDA, -) LEE  G10 S GENTOKEN(PARENTDCHA, -) LEE  G11 T GENTOKEN(LLAVEIZDA, -) LEE  G12 U GENTOKEN(LLAVEDCHA, -)	Gb		LEE		
GENTOKEN(PYC, -) LEE  GENTOKEN(PARENTIZDA, -) LEE  GIO S GENTOKEN(PARENTDCHA, -) LEE  GII T GENTOKEN(LLAVEIZDA, -) LEE  GENTOKEN(LLAVEIZDA, -) LEE  GENTOKEN(LLAVEDCHA, -)	07	27 D	GENTOKEN(COMA, -)		
G12 LEE  GENTOKEN(PARENTIZDA, -) LEE  GENTOKEN(PARENTDCHA, -) LEE  GENTOKEN(LLAVEIZDA, -) LEE  GENTOKEN(LLAVEIZDA, -) LEE	G/	P	LEE		
G12 LEE  GENTOKEN(PARENTIZDA, -) LEE  GENTOKEN(PARENTDCHA, -) LEE  GENTOKEN(LLAVEIZDA, -) LEE  GENTOKEN(LLAVEIZDA, -) LEE	00		GENTOKEN(PYC, -)		
G10 S GENTOKEN(PARENTDCHA, -)  LEE  G10 T GENTOKEN(LLAVEIZDA, -)  LEE  GENTOKEN(LLAVEDCHA, -)	G8	Q	LEE		
G10 S GENTOKEN(PARENTDCHA, -)  LEE  G10 T GENTOKEN(LLAVEIZDA, -)  LEE  GENTOKEN(LLAVEDCHA, -)		_	GENTOKEN(PARENTIZDA, -)		
G10 S LEE  G11 T GENTOKEN(LLAVEIZDA, -) LEE  G12 U GENTOKEN(LLAVEDCHA, -)	G9	K	·		
G10 S LEE  G11 T GENTOKEN(LLAVEIZDA, -) LEE  G12 U GENTOKEN(LLAVEDCHA, -)	010		GENTOKEN(PARENTDCHA, -)		
G11 LEE  GENTOKEN(LLAVEDCHA, -)	G10	5	,		
G11 LEE  GENTOKEN(LLAVEDCHA, -)	041	_	GENTOKEN(LLAVEIZDA, -)		
G12 U GENTOKEN(LLAVEDCHA, -)	G11		· · · · · · · · · · · · · · · · · · ·		
	010	1	GENTOKEN(LLAVEDCHA, -)		
	G12	U	LEE		

#### 3.3.2. Errores

Código de error	Mensaje de error	Posible causa
30	No se reconoce el carácter LEXEMA en ASCII	Se ha introducido un carácter que su codificación no se encuentra en el rango [0, 127]
31	Existe un carácter ASCII de control <sup>1</sup>	Se ha introducido un carácter de control que su codificación se encuentra en el rango [0, 31], exceptuando los códigos 10 y 13 que corresponden al salto de línea.
32	No se reconoce el operador LEXEMA	El carácter introducido no se reconoce en este lenguaje como operador, palabra reservada o identificador.
33	LEXEMA no es un carácter ASCII imprimible	Se ha introducido un carácter no imprimible en el lexema de una cadena o un comentario.
34	No se reconoce el operador &LEXEMA	Se ha introducido un carácter que no corresponde con los operadores &= o &&
35	No se reconoce el operador /LEXEMA	Se ha introducido un carácter que no corresponde con el operador /*
36	El numero LEXEMA ha excedido los 16 bits	El número no es representable en este lenguaje para un entero. Se exceden los 2B para su representación.
37	La cadena LEXEMA ha excedido los 64 caracteres	El lexema excede el máximo representable para una cadena en este lenguaje.

El módulo de Gestor de Errores, contiene el número de línea en el fichero fuente, un descriptor de fichero a un archivo 'errores' para el volcado de errores y una tabla indexada por código de error de los mensajes de error al usuario. Los estados de error, se encuentran codificados en la matriz de transición de estados para cada transición no prevista.

6

<sup>&</sup>lt;sup>1</sup>. Se ha tomado la decisión de ignorar estos caracteres y considerarlos como delimitadores en ciertas transiciones.

#### 4. Tabla de Símbolos

#### 4.1. Estructura y organización

En nuestro procesador, para implementar el módulo de la Tabla de Símbolos, hemos decidido que su estructura sea una tabla hash. En este tipo de tabla, cada elemento es indexado por una clave que, internamente corresponde a otro índice resultado de una función hash sobre esa clave. Por esta misma razón, no pueden existir dos elementos con la misma clave, en nuestro caso, dos identificadores con un mismo lexema. La búsqueda por lexema es inmediata.

En nuestra implementación existe una clase TablaSimbolos que contiene la tabla hash de objetos Simbolo indexados por su lexema y tiene las operaciones necesarias: insertar, buscar, añadir atributo, consulta de atributos y vaciar la tabla. Cada TablaSimbolos contiene un descriptor de fichero a un archivo 'simbolos', para el volcado de los datos. Además de un identificador único, por claridad en el volcado.

Cada vez que se realice una búsqueda a la tabla de símbolos, se retornará un puntero a esa entrada de la tabla. La clase Símbolo contiene todos los atributos que pueden ser accedidos y modificados, más adelante, por los demás analizadores.

# **ANEXO**

# Casos de prueba

# Caso de prueba 1

Fichero fuente	Fichero tokens		Fichero simbolos	
function void main (void) {	<function, -=""></function,>	<if, -=""></if,>	CONTENIDOS DE LA	
	<void, -=""></void,>	<parentizda, -=""></parentizda,>	TABLA # 0:	
input a;	<id, 0=""></id,>	<id, 6=""></id,>	* LEXEMA : 'main'	
input b;	<parentizda, -=""></parentizda,>	<dobleeq, -=""></dobleeq,>	Atributos :	
	<void, -=""></void,>	<entero, 0=""></entero,>	* LEXEMA : 'a'	
ret_suma_ = suma(a, b);	<llaveizda, -=""></llaveizda,>	<llaveizda, -=""></llaveizda,>	Atributos :	
output ret_suma_;	<input, -=""></input,>	<return, -=""></return,>	* LEXEMA : 'b'	
	<id, 1=""></id,>	<id, 7=""></id,>	Atributos :	
return ret;	<input, -=""></input,>	<llavedcha, -=""></llavedcha,>	* LEXEMA : 'ret_suma_'	
}	<id, 2=""></id,>	<else, -=""></else,>	Atributos :	
	<id, 3=""></id,>	<if, -=""></if,>	* LEXEMA : 'suma'	
function int suma (int num1, int num2) {	<asig, -=""></asig,>	<parentizda, -=""></parentizda,>	Atributos :	
if (num1 == 0) {	<id, 4=""></id,>	<id, 7=""></id,>	* LEXEMA : 'ret'	
return num2;	<id, 1=""></id,>	<dobleeq, -=""></dobleeq,>	Atributos :	
} else if (num2 == 0) {	<id, 2=""></id,>	<entero, 0=""></entero,>	* LEXEMA : 'num1'	
return num1;	<pyc, -=""></pyc,>	<llaveizda, -=""></llaveizda,>	Atributos :	
}	<output, -=""></output,>	<return, -=""></return,>	* LEXEMA : 'num2'	
	<id, 3=""></id,>	<id, 6=""></id,>	Atributos :	
return num1+num2;	<return, -=""></return,>	<llavedcha, -=""></llavedcha,>		
}	<id, 5=""></id,>	<return, -=""></return,>		
	<llavedcha, -=""></llavedcha,>	<id, 6=""></id,>		
	<function, -=""></function,>	<id, 7=""></id,>		
	<int, -=""></int,>	<eof, -=""></eof,>		
	<id, 4=""></id,>			
	<parentizda, -=""></parentizda,>			
	<int, -=""></int,>			
	<id, 6=""></id,>			
	<int, -=""></int,>			
	<id, 7=""></id,>			
	<llaveizda, -=""></llaveizda,>			
	Fichero errores			

Fichero fuente	Fichero	tokens	Fichero simbolos
public int void main (string args) {	<id, 0=""></id,>	<output,></output,>	CONTENIDOS DE LA
int a, b;	<int,></int,>	<cadena, se="" td="" va<=""><td>TABLA # 0:</td></cadena,>	TABLA # 0:
/* TODO:	<void,></void,>	realizar	* LEXEMA : 'public'
Interfaz chula */	<id, 1=""></id,>	operaci�n AND>	Atributos :
output 'Se va realizar un pequeño test'	<parentizda,></parentizda,>	<id, 5=""></id,>	* LEXEMA : 'main'
output 'Introduzca número 1: '	<string,></string,>	<asig,></asig,>	Atributos :
input a;	<id, 2=""></id,>	<id, 6=""></id,>	* LEXEMA : 'args'
	<llaveizda,></llaveizda,>	<id, 3=""></id,>	Atributos :
output 'Introduzca número 2: '	<int,></int,>	<id, 4=""></id,>	* LEXEMA : 'a'
input b;	<id, 3=""></id,>	<pyc,></pyc,>	Atributos :
	<id, 4=""></id,>	<output,></output,>	* LEXEMA : 'b'
output 'Se va realizar operación AND'	<output,></output,>	<id, 5=""></id,>	Atributos :
	<cadena, se="" td="" va<=""><td><llavedcha,></llavedcha,></td><td>* LEXEMA : 'c'</td></cadena,>	<llavedcha,></llavedcha,>	* LEXEMA : 'c'
c = operac10n_AnD(a,b);	realizar un	<boolean,></boolean,>	Atributos :
	peque�o test>	<id, 7=""></id,>	* LEXEMA :
output c	<output,></output,>	<int,></int,>	'operac10n_AnD'
	<cadena,< td=""><td><id, 3=""></id,></td><td>Atributos :</td></cadena,<>	<id, 3=""></id,>	Atributos :
}	Introduzca n�mero	<int,></int,>	* LEXEMA :
	1:>	<id, 4=""></id,>	'operacion_AND'
boolean operacion_AND(int a, int b) {	<input,></input,>	<llaveizda,></llaveizda,>	Atributos :
return a && b;	<id, 3=""></id,>	<return,></return,>	
}	<output,></output,>	<id, 3=""></id,>	
	<cadena,< td=""><td><id, 4=""></id,></td><td></td></cadena,<>	<id, 4=""></id,>	
	Introduzca n�mero	<eof,></eof,>	
	2:>		
	<input,></input,>		
	<id, 4=""></id,>		
	Fichero errores		

Fichero fuente	Fichero tokens		Fichero simbolos	
public int main (void) {	<id, 0=""></id,>	<int,></int,>	CONTENIDOS DE LA	
	<int,></int,>	<id, 4=""></id,>	TABLA # 0:	
a = 15;	<id, 1=""></id,>	<parentizda,></parentizda,>	* LEXEMA : 'public'	
	<parentizda,></parentizda,>	<id, 6=""></id,>	Atributos:	
output 'Introduzca el número a	<void,></void,>	<id, 2=""></id,>	* LEXEMA : 'main'	
multiplicar'	<llaveizda,></llaveizda,>	<int,></int,>	Atributos:	
input a	<id, 2=""></id,>	<id, 7=""></id,>	* LEXEMA : 'a'	
	<asig,></asig,>	<int,></int,>	Atributos:	
output 'Cuántas veces se repite?'	<entero, 15=""></entero,>	<id, 3=""></id,>	* LEXEMA : 'm'	
input m	<output,></output,>	<llaveizda,></llaveizda,>	Atributos:	
	<cadena,< td=""><td><if,></if,></td><td>* LEXEMA:</td></cadena,<>	<if,></if,>	* LEXEMA:	
<pre>int a = multiplicacion(a, m);</pre>	Introduzca el	<parentizda,></parentizda,>	'multiplicacion'	
	n�mero a	<id, 7=""></id,>	Atributos:	
output a	multiplicar>	<dobleeq,></dobleeq,>	* LEXEMA: 'private'	
	<input,></input,>	<id, 3=""></id,>	Atributos:	
return a;	<id, 2=""></id,>	<llaveizda,></llaveizda,>	* LEXEMA : 'float'	
}	<output,></output,>	<return,></return,>	Atributos:	
	<cadena,< td=""><td><entero, 0=""></entero,></td><td>* LEXEMA: 'pos'</td></cadena,<>	<entero, 0=""></entero,>	* LEXEMA: 'pos'	
private int multiplicacion (float a, int pos,	Cu�ntas veces se	<llavedcha,></llavedcha,>	Atributos:	
int m) {	repite?>	<else,></else,>		
/* No se dispone de la operacion resta */	<input,></input,>	<llaveizda,></llaveizda,>		
if (pos == m) {	<id, 3=""></id,>	<return,></return,>		
return 0;	<int,></int,>	<id, 2=""></id,>		
} else {	<id, 2=""></id,>	<suma,></suma,>		
return a + multiplicacion (a,	<asig,></asig,>	<id, 4=""></id,>		
pos + 1, m);	<id, 4=""></id,>	<parentizda,></parentizda,>		
}	<id, 2=""></id,>	<id, 2=""></id,>		
}	<id, 3=""></id,>	<id, 7=""></id,>		
	<pyc,></pyc,>	<suma,></suma,>		
	<output,></output,>	<entero, 1=""></entero,>		
	<id, 2=""></id,>	<id, 3=""></id,>		
	<return,></return,>	<pyc,></pyc,>		
	<id, 2=""></id,>	<llavedcha,></llavedcha,>		
	<llavedcha,></llavedcha,>	<eof,></eof,>		
	<id, 5=""></id,>			
	Fichero errores		•	

Fichero fuente	Fichero tokens		Fichero simbolos	
function int main (void) {	<function, -=""></function,>	<id, 3=""></id,>	CONTENIDOS DE LA TABLA #	
	<int, -=""></int,>	<int, -=""></int,>	0:	
output 'Introduzca su saldo'	<id, 0=""></id,>	<id, 2=""></id,>	* LEXEMA : 'main'	
;	<parentizda, -=""></parentizda,>	<string, -=""></string,>	Atributos :	
input saldo	<void, -=""></void,>	<id, 4=""></id,>	* LEXEMA : 'saldo'	
	<llaveizda, -=""></llaveizda,>	<int, -=""></int,>	Atributos :	
if (saldo == 0) {	<output, -=""></output,>	<id, 1=""></id,>	* LEXEMA : 'calcular_saldo'	
output 'No se concede	<cadena,< td=""><td><llaveizda, -=""></llaveizda,></td><td>Atributos :</td></cadena,<>	<llaveizda, -=""></llaveizda,>	Atributos :	
prestamo';	Introduzca su	<int, -=""></int,>	* LEXEMA : 'fuction'	
}	saldo>	<id, 5=""></id,>	Atributos :	
	<pyc, -=""></pyc,>	<asig, -=""></asig,>	* LEXEMA : 'usuario'	
calcular_saldo(saldo);	<input, -=""></input,>	<entero, 13=""></entero,>	Atributos :	
}	<id, 1=""></id,>	<boolean, -=""></boolean,>	* LEXEMA : 'tasa'	
	<if, -=""></if,>	<id, 6=""></id,>	Atributos :	
fuction int calcular_saldo(string usuario,	<parentizda, -=""></parentizda,>	<asig, -=""></asig,>	* LEXEMA : 'concede'	
int saldo) {	<id, 1=""></id,>	<id, 7=""></id,>	Atributos :	
int tasa = 13;	<dobleeq, -=""></dobleeq,>	<id, 4=""></id,>	* LEXEMA :	
/* Llamando al servicio concesion	<entero, 0=""></entero,>	<id, 5=""></id,>	'conces1on_Pre5tamo_v18A'	
de prestamos */	<llaveizda, -=""></llaveizda,>	<pyc, -=""></pyc,>	Atributos :	
boolean concede =	<output, -=""></output,>	<if, -=""></if,>		
conces1on_Pre5tamo_v18A(usuario, tasa);	<cadena, no="" se<="" td=""><td><parentizda, -=""></parentizda,></td><td></td></cadena,>	<parentizda, -=""></parentizda,>		
	concede	<id, 6=""></id,>		
if (concede) {	prestamo>	<llaveizda, -=""></llaveizda,>		
return 899099514;	<pyc, -=""></pyc,>	<return, -=""></return,>		
} else {	<llavedcha, -=""></llavedcha,>	<llavedcha, -=""></llavedcha,>		
output 'En estos momentos	<id, 2=""></id,>	<else, -=""></else,>		
la entidad no puede concederte ningun	<id, 1=""></id,>	<llaveizda, -=""></llaveizda,>		
prestamo';	<pyc, -=""></pyc,>	<output,< td=""><td></td></output,<>		
}	<llavedcha, -=""></llavedcha,>			
}				
Fichero errores				

Error: Analizador Lexico: linea 21: El numero '899099514' ha excedido los 16 bits

function int main (void) {  int a, b;	<function, -=""></function,>		
int a. b:		<parentizda, -=""></parentizda,>	CONTENIDOS DE LA
,	<int, -=""></int,>	<int, -=""></int,>	TABLA#0:
	<id, 0=""></id,>	<id, 1=""></id,>	* LEXEMA : 'main'
input a;	<parentizda, -=""></parentizda,>	<coma, -=""></coma,>	Atributos :
input b;	<void, -=""></void,>	<int, -=""></int,>	* LEXEMA : 'a'
input c;	<parentdcha, -=""></parentdcha,>	<id, 2=""></id,>	Atributos :
	<llaveizda, -=""></llaveizda,>	<coma, -=""></coma,>	* LEXEMA : 'b'
return fun(a, b, c);	<int, -=""></int,>	<boolean, -=""></boolean,>	Atributos :
}	<id, 1=""></id,>	<id, 3=""></id,>	* LEXEMA : 'c'
	<coma, -=""></coma,>	<parentdcha, -=""></parentdcha,>	Atributos :
function int fun (int a, int b, boolean c) {	<id, 2=""></id,>	<llaveizda, -=""></llaveizda,>	* LEXEMA : 'fun'
if (c == '+') {	<pyc, -=""></pyc,>	<if, -=""></if,>	Atributos :
output 'La suma de a+b es: ';	<input, -=""></input,>	<parentizda, -=""></parentizda,>	
return a+b;	<id, 1=""></id,>	<id, 3=""></id,>	
} else {	<pyc, -=""></pyc,>	<dobleeq, -=""></dobleeq,>	
output 'Lo sentimos mucho la	<input, -=""></input,>	<cadena, +=""></cadena,>	
operacion que has introducido aun no	<id, 2=""></id,>	<parentdcha, -=""></parentdcha,>	
esta disponible'	<pyc, -=""></pyc,>	<llaveizda, -=""></llaveizda,>	
return -1;	<input, -=""></input,>	<output, -=""></output,>	
}	<id, 3=""></id,>	<cadena, de<="" la="" suma="" td=""><td></td></cadena,>	
}	<pyc, -=""></pyc,>	a+b es: >	
	<return, -=""></return,>	<pyc, -=""></pyc,>	
	<id, 4=""></id,>	<return, -=""></return,>	
	<parentizda, -=""></parentizda,>	<id, 1=""></id,>	
	<id, 1=""></id,>	<suma, -=""></suma,>	
	<coma, -=""></coma,>	<id, 2=""></id,>	
	<id, 2=""></id,>	<pyc, -=""></pyc,>	
	<coma, -=""></coma,>	<llavedcha, -=""></llavedcha,>	
	<id, 3=""></id,>	<else, -=""></else,>	
	<parentdcha, -=""></parentdcha,>	<llaveizda, -=""></llaveizda,>	
	<pyc, -=""></pyc,>	<output, -=""></output,>	
	<llavedcha, -=""></llavedcha,>	<return, -=""></return,>	
	<function, -=""></function,>	<entero, 1=""></entero,>	
	<int, -=""></int,>	<llavedcha, -=""></llavedcha,>	
	<id, 4=""></id,>	<eof, -=""></eof,>	

Fichero errores

Error: Analizador Lexico: linea 16: La cadena 'Lo sentimos mucho la operacion que has introducido aun no esta disponible' ha excedido los 64 caracteres

Error: Analizador Lexico: linea 17: No se reconoce el operador '-'

Fichero fuente	Fichero tokens		Fichero simbolos
public int main (string args) {	<id, 0=""></id,>	<llavedcha,></llavedcha,>	CONTENIDOS DE LA
	<int,></int,>	<int,></int,>	TABLA # 0:
output 'Cuál es la base?'	<id, 1=""></id,>	<id, 6=""></id,>	* LEXEMA : 'public'
input &base	<parentizda,></parentizda,>	<parentizda,></parentizda,>	Atributos :
	<string,></string,>	<int,></int,>	* LEXEMA : 'main'
output 'Cuál es el exponente?'	<id, 2=""></id,>	<id, 7=""></id,>	Atributos :
input &exp	<llaveizda,></llaveizda,>	<int,></int,>	* LEXEMA : 'args'
	<output,></output,>	<id, 8=""></id,>	Atributos :
ret = potencia(base, exp);	<cadena, cu�l="" es<="" td=""><td><llaveizda,></llaveizda,></td><td>* LEXEMA : 'ase'</td></cadena,>	<llaveizda,></llaveizda,>	* LEXEMA : 'ase'
	la base?>	<if,></if,>	Atributos :
output 'Resultado : ' + ret	<input,></input,>	<parentizda,></parentizda,>	* LEXEMA : 'xp'
	<id, 3=""></id,>	<id, 8=""></id,>	Atributos :
return ret;	<output,></output,>	<dobleeq,></dobleeq,>	* LEXEMA : 'ret'
}	<cadena, cu�l="" es<="" td=""><td><entero, 0=""></entero,></td><td>Atributos :</td></cadena,>	<entero, 0=""></entero,>	Atributos :
	el exponente?>	<llaveizda,></llaveizda,>	* LEXEMA : 'potencia'
int potencia (int base, int exp) {	<input,></input,>	<return,></return,>	Atributos :
if (exp == 0) {	<id, 4=""></id,>	<entero, 1=""></entero,>	* LEXEMA : 'base'
return 1;	<id, 5=""></id,>	<llavedcha,></llavedcha,>	Atributos :
} else {	<asig,></asig,>	<else,></else,>	* LEXEMA : 'exp'
return	<id, 6=""></id,>	<llaveizda,></llaveizda,>	Atributos :
multiplicacion(base, base) +	<id, 7=""></id,>	<return,></return,>	* LEXEMA : 'bits'
potencia(base, exp-1);	<id, 8=""></id,>	<id, 10=""></id,>	Atributos :
}	<pyc,></pyc,>	<id, 7=""></id,>	* LEXEMA :
}	<id, 9=""></id,>	<id, 7=""></id,>	'multiplicacion'
	<asig,></asig,>	<suma,></suma,>	Atributos :
	<id, 9=""></id,>	<id, 6=""></id,>	
	<id, 5=""></id,>	<id, 7=""></id,>	
	<pyc,></pyc,>	<id, 8=""></id,>	
	<output,></output,>	<entero, 1=""></entero,>	
	<cadena,< td=""><td><pyc,></pyc,></td><td></td></cadena,<>	<pyc,></pyc,>	
	Resultado:>	<llavedcha,></llavedcha,>	
	<suma,></suma,>	<eof,></eof,>	
	<id, 5=""></id,>		
	<return,></return,>		
	<id, 5=""></id,>		

Fichero errores

Error : Analizador Lexico : linea 4 : No se reconoce el operador '&b' Error : Analizador Lexico : linea 7 : No se reconoce el operador '&e'