GRUPO 63

Procesador de Lenguajes: JS--

Procesadores de Lenguajes

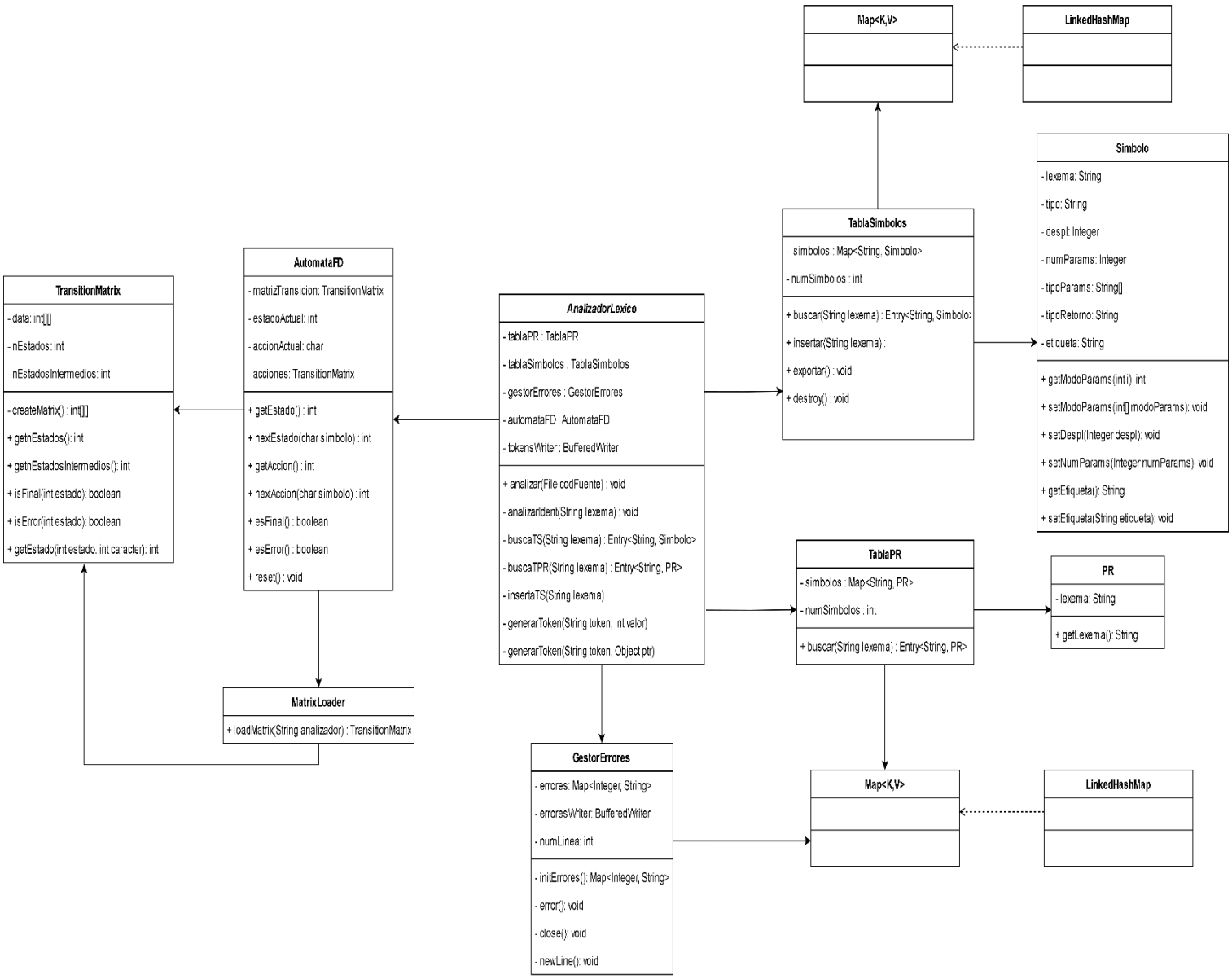
Ignacio García Fernández
José Ruiz Esteban
Eduardo Gil Alba
4-10-2024

Tabla de contenido

1. 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**	<CADENA, lexema> **Cadenas de caracteres**	<ID, posTS> **Identificadores**
<ENTERO, valor> **Constantes enteras**	<SUMA, -> **+** **Operador aritmético**	<ASIG, -> **=** **Operador asignación**
< ASIGAND , -> **&=** **Operador y lógico**	< DOBLEEQ , -> **==** **Operador relacional**	<DOBLEAND, -> **&&** **Operador lógico**
< PARENTIZDA , -> **(** **Apertura paréntesis**	< PARENTDCHA , -> **)** **Cierre paréntesis**	< LLAVEIZDA , -> **{** **Apertura bloque**
< LLAVEDCHA , -> **}** **Cierre bloque**	<PYC, -> **;** **Punto y coma**	<COMA, -> **,** **Coma**
	<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_1:T-{'} c_2:T-{*} c_3: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_1ST| '

4: Y → &|=

5: I → =|λ

6: C → *D

7: D → c_2D|*E

8: E → /A|c_3D|*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.

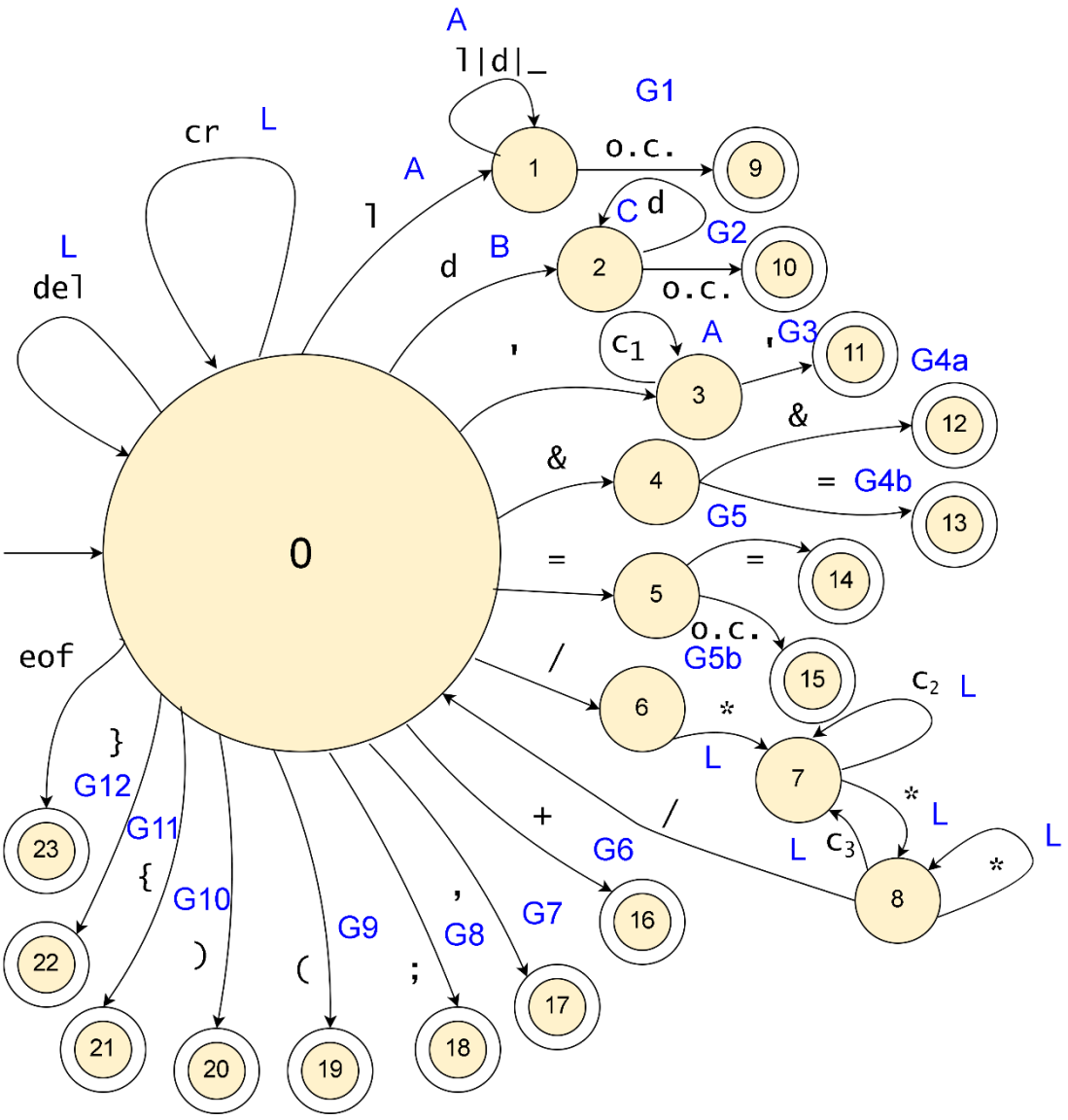
El comportamiento del analizador léxico, va determinado por el autómata generado por esta grámatica (siguiente apartado). La columna de la izquierda indica el estado del autómata. El axioma A corresponde al estado 0 o inicial.

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	Acción en el código	Descripción
A	A	CONCAT LEE
B	B	num := valor(d)
C	C	num := num*10 + valor(d)
L	L	LEE

G1	G	p := BUSCATPR(lex)
		if (p != NULL) GENTOKEN(PR, p)
		p := BUSCATS(lex)
		if (p == NULL) p := INSERTATS(lex)
		GENTOKEN(ID, p)

G2	H	if (num > 32767) { ERROR(36, num) LEE } GENTOKEN(ENTERO, num)
G3	I	if (longitud(lex) > 64) { ERROR(37, lex) LEE } GENTOKEN(CADENA, 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(PARENTDCHA, -) LEE
G11	T	GENTOKEN(LLAVEIZDA, -) LEE
G12	U	GENTOKEN(LLAVEDCHA, -) 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[1]	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.

[1]. 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

<table>
<tr><th>Fichero fuente</th><th colspan="2">Fichero tokens</th><th>Fichero simbolos</th></tr>
<tr>
<td>function void main (void) {

  input a;
  input b;

  ret_suma_ = suma(a, b);
  output ret_suma_;

  return ret;
}

function int suma (int num1, int num2) {
  if (num1 == 0) {
  return num2;
  } else if (num2 == 0) {
  return num1;
  }

  return num1+num2;
}</td>
<td><FUNCTION, ->
<VOID, ->
<ID, 0>
<PARENTIZDA, ->
<VOID, ->
<LLAVEIZDA, ->
<INPUT, ->
<ID, 1>
<INPUT, ->
<ID, 2>
<ID, 3>
<ASIG, ->
<ID, 4>
<ID, 1>
<ID, 2>
<PYC, ->
<OUTPUT, ->
<ID, 3>
<RETURN, ->
<ID, 5>
<LLAVEDCHA, ->
<FUNCTION, ->
<INT, ->
<ID, 4>
<PARENTIZDA, ->
<INT, ->
<ID, 6>
<INT, ->
<ID, 7>
<LLAVEIZDA, -></td>
<td><IF, ->
<PARENTIZDA, ->
<ID, 6>
<DOBLEEQ, ->
<ENTERO, 0>
<LLAVEIZDA, ->
<RETURN, ->
<ID, 7>
<LLAVEDCHA, ->
<ELSE, ->
<IF, ->
<PARENTIZDA, ->
<ID, 7>
<DOBLEEQ, ->
<ENTERO, 0>
<LLAVEIZDA, ->
<RETURN, ->
<ID, 6>
<LLAVEDCHA, ->
<RETURN, ->
<ID, 6>
<ID, 7>
<EOF, -></td>
<td>CONTENIDOS DE LA TABLA # 0 :
* LEXEMA : 'main'
  Atributos :
* LEXEMA : 'a'
  Atributos :
* LEXEMA : 'b'
  Atributos :
* LEXEMA : 'ret_suma_'
  Atributos :
* LEXEMA : 'suma'
  Atributos :
* LEXEMA : 'ret'
  Atributos :
* LEXEMA : 'num1'
  Atributos :
* LEXEMA : 'num2'
  Atributos :</td>
</tr>
<tr><td colspan="4">Fichero errores</td></tr>
<tr><td colspan="4"></td></tr>
</table>

Caso de prueba 2

<table>
<tr><th>Fichero fuente</th><th colspan="2">Fichero tokens</th><th>Fichero simbolos</th></tr>
<tr>
<td>public int void main (string args) {
  int a, b;
  /* TODO:
  Interfaz chula */
  output 'Se va realizar un pequeño test'
  output 'Introduzca número 1: '
  input a;

  output 'Introduzca número 2: '
  input b;

  output 'Se va realizar operación AND'

  c = operac10n_AnD(a,b);

  output c

}

boolean operacion_AND(int a, int b) {
  return a && b;
}</td>
<td><ID, 0>
<INT, >
<VOID, >
<ID, 1>
<PARENTIZDA, >
<STRING, >
<ID, 2>
<LLAVEIZDA, >
<INT, >
<ID, 3>
<ID, 4>
<OUTPUT, >
<CADENA, Se va realizar un peque&#65533;o test>
<OUTPUT, >
<CADENA, Introduzca n&#65533;mero 1: >
<INPUT, >
<ID, 3>
<OUTPUT, >
<CADENA, Introduzca n&#65533;mero 2: >
<INPUT, >
<ID, 4></td>
<td><OUTPUT, >
<CADENA, Se va realizar operaci&#65533;n AND>
<ID, 5>
<ASIG, >
<ID, 6>
<ID, 3>
<ID, 4>
<PYC, >
<OUTPUT, >
<ID, 5>
<LLAVEDCHA, >
<BOOLEAN, >
<ID, 7>
<INT, >
<ID, 3>
<INT, >
<ID, 4>
<LLAVEIZDA, >
<RETURN, >
<ID, 3>
<ID, 4>
<EOF, ></td>
<td>CONTENIDOS DE LA TABLA # 0 :
* LEXEMA : 'public'
  Atributos :
* LEXEMA : 'main'
  Atributos :
* LEXEMA : 'args'
  Atributos :
* LEXEMA : 'a'
  Atributos :
* LEXEMA : 'b'
  Atributos :
* LEXEMA : 'c'
  Atributos :
* LEXEMA : 'operac10n_AnD'
  Atributos :
* LEXEMA : 'operacion_AND'
  Atributos :</td>
</tr>
<tr><td colspan="4">Fichero errores</td></tr>
<tr><td colspan="4"></td></tr>
</table>

Caso de prueba 3

<table>
<tr><th>Fichero fuente</th><th colspan="2">Fichero tokens</th><th>Fichero simbolos</th></tr>
<tr>
<td>public int main (void) {

a = 15;

output 'Introduzca el número a multiplicar'
input a

output 'Cuántas veces se repite?'
input m

int a = multiplicacion(a, m);

output a

return a;
}

private int multiplicacion (float a, int pos, int m) {
/* No se dispone de la operacion resta */
if (pos == m) {
return 0;
} else {
return a + multiplicacion (a, pos + 1, m);
}
}</td>
<td><ID, 0>
<INT, >
<ID, 1>
<PARENTIZDA, >
<VOID, >
<LLAVEIZDA, >
<ID, 2>
<ASIG, >
<ENTERO, 15>
<OUTPUT, >
<CADENA, Introduzca el n&#65533;mero a multiplicar>
<INPUT, >
<ID, 2>
<OUTPUT, >
<CADENA, Cu&#65533;ntas veces se repite?>
<INPUT, >
<ID, 3>
<INT, >
<ID, 2>
<ASIG, >
<ID, 4>
<ID, 2>
<ID, 3>
<PYC, >
<OUTPUT, >
<ID, 2>
<RETURN, >
<ID, 2>
<LLAVEDCHA, >
<ID, 5></td>
<td><INT, >
<ID, 4>
<PARENTIZDA, >
<ID, 6>
<ID, 2>
<INT, >
<ID, 7>
<INT, >
<ID, 3>
<LLAVEIZDA, >
<IF, >
<PARENTIZDA, >
<ID, 7>
<DOBLEEQ, >
<ID, 3>
<LLAVEIZDA, >
<RETURN, >
<ENTERO, 0>
<LLAVEDCHA, >
<ELSE, >
<LLAVEIZDA, >
<RETURN, >
<ID, 2>
<SUMA, >
<ID, 4>
<PARENTIZDA, >
<ID, 2>
<ID, 7>
<SUMA, >
<ENTERO, 1>
<ID, 3>
<PYC, >
<LLAVEDCHA, >
<EOF, ></td>
<td>CONTENIDOS DE LA TABLA # 0 :
* LEXEMA : 'public'
Atributos :
* LEXEMA : 'main'
Atributos :
* LEXEMA : 'a'
Atributos :
* LEXEMA : 'm'
Atributos :
* LEXEMA : 'multiplicacion'
Atributos :
* LEXEMA : 'private'
Atributos :
* LEXEMA : 'float'
Atributos :
* LEXEMA : 'pos'
Atributos :</td>
</tr>
<tr><td colspan="4">Fichero errores</td></tr>
<tr><td colspan="4"></td></tr>
</table>

Caso de prueba 4

<table>
<tr><th>Fichero fuente</th><th colspan="2">Fichero tokens</th><th>Fichero simbolos</th></tr>
<tr>
<td>function int main (void) {

output 'Introduzca su saldo'
;
input saldo

if (saldo == 0) {
output 'No se concede prestamo';
}

calcular_saldo(saldo);
}

fuction int calcular_saldo(string usuario, int saldo) {
int tasa = 13;
/* Llamando al servicio concesion de prestamos */
boolean concede = conces1on_Pre5tamo_v18A(usuario, tasa);

if (concede) {
return 899099514;
} else {
output 'En estos momentos la entidad no puede concederte ningun prestamo';
}
}</td>
<td><FUNCTION, ->
<INT, ->
<ID, 0>
<PARENTIZDA, ->
<VOID, ->
<LLAVEIZDA, ->
<OUTPUT, ->
<CADENA, Introduzca su saldo>
<PYC, ->
<INPUT, ->
<ID, 1>
<IF, ->
<PARENTIZDA, ->
<ID, 1>
<DOBLEEQ, ->
<ENTERO, 0>
<LLAVEIZDA, ->
<OUTPUT, ->
<CADENA, No se concede prestamo>
<PYC, ->
<LLAVEDCHA, ->
<ID, 2>
<ID, 1>
<PYC, ->
<LLAVEDCHA, -></td>
<td><ID, 3>
<INT, ->
<ID, 2>
<STRING, ->
<ID, 4>
<INT, ->
<ID, 1>
<LLAVEIZDA, ->
<INT, ->
<ID, 5>
<ASIG, ->
<ENTERO, 13>
<BOOLEAN, ->
<ID, 6>
<ASIG, ->
<ID, 7>
<ID, 4>
<ID, 5>
<PYC, ->
<IF, ->
<PARENTIZDA, ->
<ID, 6>
<LLAVEIZDA, ->
<RETURN, ->
<LLAVEDCHA, ->
<ELSE, ->
<LLAVEIZDA, ->
<OUTPUT,</td>
<td>CONTENIDOS DE LA TABLA # 0 :
* LEXEMA : 'main'
Atributos :
* LEXEMA : 'saldo'
Atributos :
* LEXEMA : 'calcular_saldo'
Atributos :
* LEXEMA : 'fuction'
Atributos :
* LEXEMA : 'usuario'
Atributos :
* LEXEMA : 'tasa'
Atributos :
* LEXEMA : 'concede'
Atributos :
* LEXEMA : 'conces1on_Pre5tamo_v18A'
Atributos :</td>
</tr>
<tr><td colspan="4">Fichero errores</td></tr>
<tr><td colspan="4">Error : Analizador Lexico : linea 21 : El numero '899099514' ha excedido los 16 bits</td></tr>
</table>

Caso de prueba 5

<table>
<tr><th>Fichero fuente</th><th colspan="2">Fichero tokens</th><th>Fichero simbolos</th></tr>
<tr>
<td>function int main (void) {
  int a, b;

  input a;
  input b;
  input c;

  return fun(a, b, c);
}

function int fun (int a, int b, boolean c) {
  if (c == '+') {
    output 'La suma de a+b es: ' ;
    return a+b;
  } else {
    output 'Lo sentimos mucho la operacion que has introducido aun no esta disponible'
    return -1;
  }
}</td>
<td><FUNCTION, ->
<INT, ->
<ID, 0>
<PARENTIZDA, ->
<VOID, ->
<PARENTDCHA, ->
<LLAVEIZDA, ->
<INT, ->
<ID, 1>
<COMA, ->
<ID, 2>
<PYC, ->
<INPUT, ->
<ID, 1>
<PYC, ->
<INPUT, ->
<ID, 2>
<PYC, ->
<INPUT, ->
<ID, 3>
<PYC, ->
<RETURN, ->
<ID, 4>
<PARENTIZDA, ->
<ID, 1>
<COMA, ->
<ID, 2>
<COMA, ->
<ID, 3>
<PARENTDCHA, ->
<PYC, ->
<LLAVEDCHA, ->
<FUNCTION, ->
<INT, ->
<ID, 4></td>
<td><PARENTIZDA, ->
<INT, ->
<ID, 1>
<COMA, ->
<INT, ->
<ID, 2>
<COMA, ->
<BOOLEAN, ->
<ID, 3>
<PARENTDCHA, ->
<LLAVEIZDA, ->
<IF, ->
<PARENTIZDA, ->
<ID, 3>
<DOBLEEQ, ->
<CADENA, +>
<PARENTDCHA, ->
<LLAVEIZDA, ->
<OUTPUT, ->
<CADENA, La suma de a+b es: >
<PYC, ->
<RETURN, ->
<ID, 1>
<SUMA, ->
<ID, 2>
<PYC, ->
<LLAVEDCHA, ->
<ELSE, ->
<LLAVEIZDA, ->
<OUTPUT, ->
<RETURN, ->
<ENTERO, 1>
<LLAVEDCHA, ->
<EOF, -></td>
<td>CONTENIDOS DE LA TABLA # 0 :
* LEXEMA : 'main'
  Atributos :
* LEXEMA : 'a'
  Atributos :
* LEXEMA : 'b'
  Atributos :
* LEXEMA : 'c'
  Atributos :
* LEXEMA : 'fun'
  Atributos :</td>
</tr>
<tr><td colspan="4">Fichero errores</td></tr>
<tr><td colspan="4">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 '-'</td></tr>
</table>

Caso de prueba 6

<table>
<tr><th>Fichero fuente</th><th colspan="2">Fichero tokens</th><th>Fichero simbolos</th></tr>
<tr>
<td>public int main (string args) {

output 'Cuál es la base?'
input &base;

output 'Cuál es el exponente?'
input &exp;

ret = potencia(base, exp);

output 'Resultado : ' + ret

return ret;
}

int potencia (int base, int exp) {
if (exp == 0) {
return 1;
} else {
return
multiplicacion(base, base) +
potencia(base, exp-1);
}
}</td>
<td><ID, 0>
<INT, >
<ID, 1>
<PARENTIZDA, >
<STRING, >
<ID, 2>
<LLAVEIZDA, >
<OUTPUT, >
<CADENA, Cu&#65533;l es la base?>
<INPUT, >
<ID, 3>
<OUTPUT, >
<CADENA, Cu&#65533;l es el exponente?>
<INPUT, >
<ID, 4>
<ID, 5>
<ASIG, >
<ID, 6>
<ID, 7>
<ID, 8>
<PYC, >
<ID, 9>
<ASIG, >
<ID, 9>
<ID, 5>
<PYC, >
<OUTPUT, >
<CADENA, Resultado : >
<SUMA, >
<ID, 5>
<RETURN, >
<ID, 5></td>
<td><LLAVEDCHA, >
<INT, >
<ID, 6>
<PARENTIZDA, >
<INT, >
<ID, 7>
<INT, >
<ID, 8>
<LLAVEIZDA, >
<IF, >
<PARENTIZDA, >
<ID, 8>
<DOBLEEQ, >
<ENTERO, 0>
<LLAVEIZDA, >
<RETURN, >
<ENTERO, 1>
<LLAVEDCHA, >
<ELSE, >
<LLAVEIZDA, >
<RETURN, >
<ID, 10>
<ID, 7>
<ID, 7>
<SUMA, >
<ID, 6>
<ID, 7>
<ID, 8>
<ENTERO, 1>
<PYC, >
<LLAVEDCHA, >
<EOF, ></td>
<td>CONTENIDOS DE LA TABLA # 0 :
* LEXEMA : 'public'
Atributos :
* LEXEMA : 'main'
Atributos :
* LEXEMA : 'args'
Atributos :
* LEXEMA : 'ase'
Atributos :
* LEXEMA : 'xp'
Atributos :
* LEXEMA : 'ret'
Atributos :
* LEXEMA : 'potencia'
Atributos :
* LEXEMA : 'base'
Atributos :
* LEXEMA : 'exp'
Atributos :
* LEXEMA : 'bits'
Atributos :
* LEXEMA : 'multiplicacion'
Atributos :</td>
</tr>
<tr><td colspan="4">Fichero errores</td></tr>
<tr><td colspan="4">Error : Analizador Lexico : linea 4 : No se reconoce el operador '&b'
Error : Analizador Lexico : linea 7 : No se reconoce el operador '&e'</td></tr>
</table>