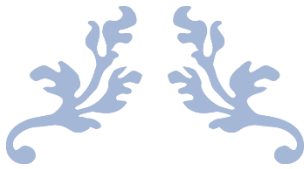


PROCESADORES DE LENGUAJES

Primera Entrega: Analizador Léxico y Tabla de Símbolos

GRUPO 69

Sergio de Ana Ramos
Alberto de la Torre Juárez
Mario del Rincón García

Índice:

1. Objetivo……2

2. Especificación de grupo……2

3. Introducción……2

4. Analizador Léxico……3

4.1. Tokens……3

4.2. Gramática del lenguaje……3

4.3. Autómata finito……4

4.4. Acciones semánticas y errores……4

5. Tabla de símbolos……5

6. Errores……6

7. Pruebas……6

7.1. Pruebas de éxito……6

7.1.1. Prueba 1……6

7.1.2. Prueba 2……7

7.1.3. Prueba 3……8

7.2. Pruebas de errores……9

7.2.1. Prueba 1……9

7.2.2. Prueba 2……9

7.2.3. Prueba 3……10

1. Objetivo:

Esta primera entrega de la práctica consiste en el diseño e implementación de un analizador léxico (tokens, gramática, autómata, acciones semánticas y errores) y de la tabla de símbolos (descripción de su estructura final y organización) de un procesador de lenguajes para un determinado lenguaje de programación denominado JavaScript-PL.

2. Especificación del grupo:

A parte de las características globales para todos los grupos, a nosotros se nos han asignado las siguientes:

- Sentencias: **Sentencia repetitiva (do-while)**
- Operadores especiales: **Post-auto-decremento (-- como sufijo)**
- Técnica de Análisis Sintáctico: **Descendente recursivo**
- Comentarios: **Comentarios de línea (//)**
- Cadenas: **Con comillas simples (‘ ‘)**

Además, del resto de opciones se han elegido:

- Operador aritmético: **Suma (+)**
- Operador relacional: **Igual (==)**
- Operador lógico: **Y lógico (&&)**

3. Introducción:

Se ha decidido usar Java como lenguaje en la práctica.

El diseño del paquete tiene varios scripts:

- **ResultadoToken.java**: Este archivo es el que crea el fichero con los tokens resultado y cómo se generan (su apariencia).
- **Token.java**: Genera los tokens que va encontrando en el fichero fuente o el error si recibe caracteres no programados.
- **ItemTS.java**: Genera las tablas de símbolos que se obtienen a partir del fichero fuente.

En nuestra aplicación se deberá elegir un fichero fuente y un directorio donde crear los ficheros resultado: lista de tokens, tabla de símbolos y errores.

4. Analizador léxico:

La principal tarea del analizador léxico será analizar el fichero fuente carácter a carácter y generar los tokens necesarios o errores encontrados, una vez esto se los trasmitirá al posterior analizador, el analizador sintáctico.

4.1. Tokens:

Se ha agrupado el conjunto de características tanto globales, como especiales a nuestro grupo y se ha diseñado el siguiente número de tokens:

<boolean, _>
<do, _>
<function, _>
<if, _>
<input, _>
<int, _>
<print, _>
<return, _>
<string, _>
<var, _>
<while, _>
<decremento, _> [--]
<entero, valor>
<cad, lexema>
<id, posTS>
<asign, _> [=]
<coma, _> [,]
<pcoma, _> [;]
<pabierto, _> [(]
<pcerrado, _> [)]
<cabierto, _> [{]
<ccerrado, _> [}]
<oparit, _> [+]
<oplog, _> [&&]
<oprel, _> [<]

4.2. Gramática del lenguaje:

Se ha diseñado la siguiente gramática que permita implementar los tokens anteriores:

S → del S | (|) | ; | , | & A | { | } | + | = B | ‘ C | d D | - E | l F | < H

A→ &

B→ λ

C→ l C | d C | _ C | - C | c C | ‘ | del C

D→ d D | λ

E→ -

F→ l F | d G | _ G | λ

G→ d G | l G | _ G | λ

H→ λ

Donde los caracteres utilizados toman los valores:

l = ({A…Z}, {a…z})

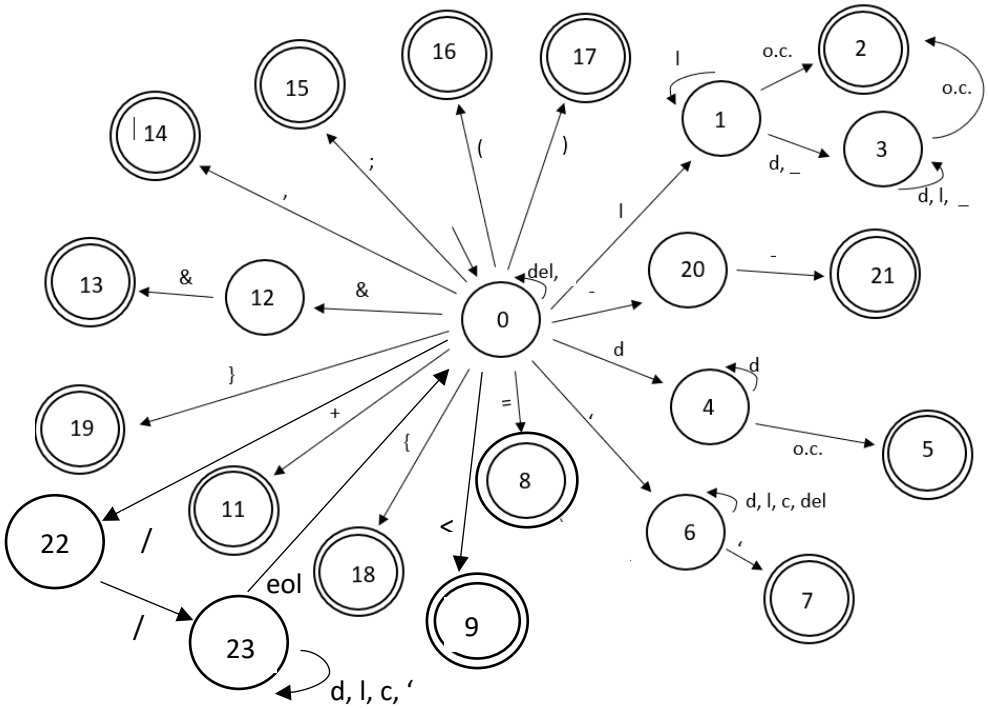
d = {0...9}

c = cualquier carácter - { ‘ }

del = {<blanco>, < tab >}

4. 3. Autómata finito:

El autómata obtenido tras la gramática es:



oc = otra cosa distinta de lo que se espera

4.4. Acciones semánticas y errores:

0:0 Leer();
0:1 lexema=l; Leer();
1:1 lexema=lexema⊕l; Leer();
1:2 p:= buscarTS (lexema);
if (p∈PalabraReservada) then GenToken (PalabraReservada, _);
if (p=NULL) then {
if (ZonaDeclaracion) then p:=intertarTS (lexema); GenToken (id, p);
else Error ("Identificador no declarado por zona");
} else if (ZonaDeclaracion) then Error ("Identificador ya declarado");
Leer();
1:3 lexema=lexema⊕d/_; Leer();
3:3 lexema=lexema⊕d/_/l; Leer();
3:2 p:= buscarTS (lexema);
if (p∈PalabraReservada) then GenToken (PalabraReservada, _);
if (p=NULL) then {
if (ZonaDeclaracion) then p:=intertarTS (lexema); GenToken (lexema, p);
else Error ("Identificador no declarado por zona");
} else if (ZonaDeclaracion) then Error ("Identificador ya declarado");
Leer();
0:4 numero=valorASCII (d); Leer();

4:4 numero=numero*10+valorASCII(d); Leer();
4:5 if (valor<2^15) then GenToken (entero, valor);
else Error (“Número fuera de rango”);
Leer();
0:6 lexema= ‘ ; Leer();
6:6 lexema=lexema⊕d/_/l; Leer();
6:7 lexema=lexema⊕ ‘ ; GenToken (cad, lexema); Leer();
0:8 GenToken (asign, _); Leer();
0:9 GenToken (oprel, _); Leer();
0:11 GenToken (oparit, _); Leer();
0:12 Leer();
12:13 GenToken (oplog, _); Leer();
0:14 GenToken (coma, _); Leer();
0:15 GenToken (pcoma, _); Leer();
0:16 GenToken (pabierto, _); Leer();
0:17 GenToken (pcerrado, _); Leer();
0:18 GenToken (cabierto, _); Leer();
0:19 GenToken(ccerrado, _); Leer();
0:20 Leer();
0:21 GenToken (decremento, _); Leer();
0:22 Leer();
22:23 Leer();
23:23 Leer();

5. Tabla de símbolos:

Las tablas que generaría nuestro programa serian:

- Tabla de símbolos Global: tabla global de información.
- Tabla de símbolos Fn: tabla de símbolos de funciones. Tabla local.

Tabla de Símbolos Global:

Índice	Lexema	Tipo	Desplazamiento
…	…	…	…

Tabla de Símbolos Fn (funciones):

Índice	Lexema	Tipo	NumParam	TipoRetorno	EtiqFuncion
…	…	…	…	…	…

6. Errores:

Los errores se van tratando en cada parte de la ejecución en paralelo con el resto de operaciones. Un error en el analizador léxico sería, por ejemplo, un tipo de token desconocido. Pero hay que tener en cuenta que, las palabras clave no identificadas (por ejemplo, en nuestro caso, un for) serán tokens id (identificadores).

7. Pruebas:

7.1. Pruebas de éxito:

7.1.1. Prueba 1:

Código fuente:

```
var int c;
c = 100;
function int b (){
        var int c;
        var string s;
        s = '$en una funcion$';
        // este comentario no deberia salir
        c = 3;
        c--;
        if (true)
                n = 4 + c;
        return c;
}
```

Tokens generados:

```
<var,>
<int,>
<id,1>
<pcoma,>
<id,1>
<asign,>
<entero,100>
<pcoma,>
<function,>
<int,>
<id,2>
<pabierto,>
<pcerrado,>
<cabierto,>
<var,>
<int,>
<id,1>
<pcoma,>
<var,>
<string,>
<id,3>
<pcoma,>
<id,3>
<asign,>
<cad,"$en una funcion$">
<pcoma,>
<id,1>
<asign,>
<entero,3>
<pcoma,>
<id,1>
<decremento,>
<pcoma,>
<if,>
<pabierto,>
<id,5>
<pcerrado,>
<id,4>
<asign,>
<entero,4>
<oparit,>
<id,1>
<pcoma,>
<return,>
<id,1>
<pcoma,>
<ccerrado,>
```

Tabla de símbolos:

```
TABLA PRINCIPAL #1:
* LEXEMA: 'b'
        + tipo: 'función'
        + numParam: 0
        + TipoRetorno: 'entero'
        + EtiqFuncion: 'Etb1’

* LEXEMA: 'c'
        + tipo: 'entero'
        + despl: 0

TABLA FUNCION b #2:
* LEXEMA: 's'
        + tipo: 'cadena'
        + despl: 2
* LEXEMA: 'c'
        + tipo: 'entero'
        + despl: 0
```

Archivo de errores:

```
No se han encontrado errores en este código.
```

7.1.2. Prueba 2:

Código fuente:

```
var string s;
s = 'esto se va a imprimir';
var int n;
n = 1;
print(n + s);
var string s2;
s = 'esto es el final';
n = 2;
print(n + s2);
```

Tokens generados:

```
<var,>
<string,>
<id,1>
<pcoma,>
<id,1>
<asign,>
<cad,"esto se va a imprimir">
<pcoma,>
<var,>
<int,>
<id,2>
<pcoma,>
<id,2>
<asign,>
<entero,1>
<pcoma,>
<print,>
<pabierto,>
<id,2>
<oparit,>
<id,1>
<pcerrado,>
<pcoma,>
<var,>
<string,>
<id,3>
<pcoma,>
<id,1>
<asign,>
<cad,"esto es el final">
<pcoma,>
<id,2>
<asign,>
<entero,2>
<pcoma,>
<print,>
<pabierto,>
<id,2>
<oparit,>
<id,3>
<pcerrado,>
<pcoma,>
```

Tabla de símbolos:

```
TABLA PRINCIPAL #1:
* LEXEMA: 's2'
        + tipo: 'cadena'
        + despl: 23

* LEXEMA: 'n'
        + tipo: 'entero'
        + despl: 21

* LEXEMA: 's'
        + tipo: 'cadena'
        + despl: 0
```

Archivo de errores:

```
No se han encontrado errores en este código.
```

7.1.3. Prueba 3:

Código fuente:

```
//un while
var bool a;
var bool b;
var int d;
var int c;
c = 1;
a = true;
b = false;
while ( a = b ){
        d = c--;
        if (c = d)
                b = true;
}
```

Tokens generados:

```
<var,>
<id,1>
<id,2>
<pcoma,>
<var,>
<id,1>
<id,3>
<pcoma,>
<var,>
<int,>
<id,4>
<pcoma,>
<var,>
<int,>
<id,5>
<pcoma,>
<id,5>
<asign,>
<entero,1>
<pcoma,>
<id,2>
<asign,>
<id,6>
<pcoma,>
<id,3>
<asign,>
<id,7>
<pcoma,>
<while,>
<pabierto,>
<id,2>
<asign,>
<id,3>
<pcerrado,>
<cabierto,>
<id,4>
<asign,>
<id,5>
<decremento,>
<pcoma,>
<if,>
<pabierto,>
<id,5>
<asign,>
<id,4>
<pcerrado,>
<id,3>
<asign,>
<id,6>
<pcoma,>
<ccerrado,>
```

Tabla de símbolos:

```
TABLA PRINCIPAL #1:
* LEXEMA: 'c'
        + tipo: 'entero'
        + despl: 4

* LEXEMA: 'd'
        + tipo: 'entero'
        + despl: 2

* LEXEMA: 'b'
        + tipo: 'lógico'
        + despl: 1

* LEXEMA: 'a'
        + tipo: 'lógico'
        + despl: 0
```

Archivo de errores:

```
No se han encontrado errores en este código.
```

7.2. Pruebas de errores:

7.2.1. Prueba 1:

Código fuente:

```
/* este comentario no esta */
var s; //variable sin tipo
var int d;
a = 5;
b = 4; //esto funciona pq cualquier variable no definida se inicializa bien
c = a%b;
return a>b;
```

Archivo de errores:

```
Error, no se reconoce el caracter "*" .
Error, no se reconoce el caracter "*" .
Error, no se reconoce el caracter "%" .
Error, no se reconoce el caracter ">" .
```

7.2.2. Prueba 2:

Código fuente:

```
var int c;
c = 40000;
var string s;
s = "$en una funcion$";
// este comentario no deberia salir
```

Archivo de errores:

```
Error, numero 40000 fuera de rango
Error, no se reconoce el caracter """ .
Error, no se reconoce el caracter "$" .
Error, no se reconoce el caracter "$" .
Error, no se reconoce el caracter """ .
```

7.2.3. Prueba 3:

Código fuente:

```
var bool a;
var bool b;
var int d;
var int c;
c = 1;
a = true;
b = false;
while ( a!=b ){
        d = d + c;
        if (c = d)
                b = true.false;
}
```

Archivo de errores:

```
Error, no se reconoce el caracter "!" .
Error, no se reconoce el caracter "." .
```