



INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE

REPORTE DE ACTIVIDAD

CARRERA:

INGENIERÍA EN SISTEMAS COMPUTACIONALES

ASIGNATURA:

LENGUAJES Y AUTOMATAS I

NOMBRE DE LA ACTIVIDAD:

PROGRAMA: ANALIZADOR LÉXICO

NOMBRE DEL ALUMNO:

GONZALO MARTINEZ SILVERIO

DOCENTE:

DR. TANIA TURRUBIATES LÓPEZ

PERIODO ESCOLAR:

FEB 2023 – JUN 2023

SEMESTRE:

6°

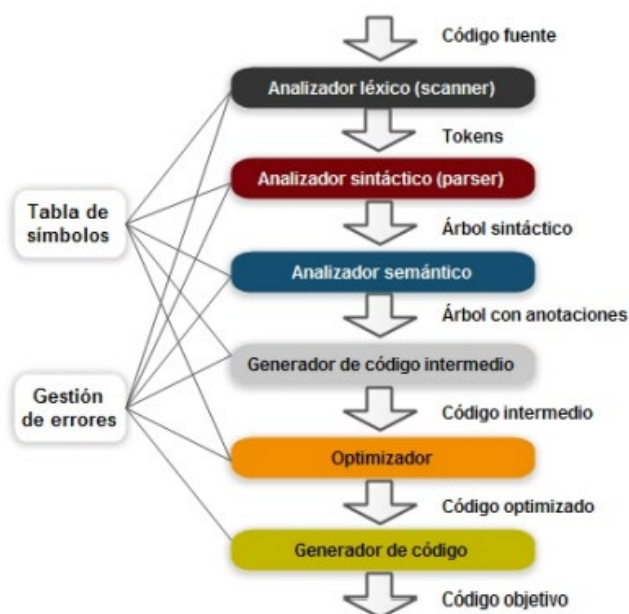
GRUPO:

6S1A



INTRODUCCIÓN:

En este tema vamos a proceder a definir y comprender todas las tareas que realiza el analizador léxico y que son clave para el correcto funcionamiento del compilador. Como vemos en la figura, tiene como entrada el código fuente del lenguaje de programación que acepta el compilador y como salida, proporciona al analizador sintáctico los tokens.



¿Qué es un token?

Es una agrupación de caracteres reconocidos por el analizador léxico que constituyen los símbolos con los que se forman las sentencias del lenguaje.

Lo que el analizador léxico devuelve al analizador sintáctico es el nombre de ese símbolo junto con el valor del atributo (si ese token lo necesita, ya que no todos los tokens llevan atributo, por ejemplo: una palabra reservada "if").

MATERIALES:

1. Computadora personal.
2. NetBeans 12.6
3. JDK.
4. Microsoft Office Word.
5. Apuntes.



REPORTE DE ACTIVIDAD

EJERCICIO PROPUESTO POR EL DOCENTE

Construir un analizador léxico que acepte palabras reservadas, identificadores, enteros positivos y operadores aritméticos y relacionales. Además debe eliminar blancos y comentarios y almacenar en una tabla de símbolos los identificadores y las constantes.

Solución:

Palabras reservadas

El analizador reconocerá las siguientes 20 palabras reservadas:

PROGRAM	END	STEP	TO
ARRAY	INTEGER	CASE	DO
VAR	IF	ELSE	WHILE
CONST	THEN	CHAR	REPEAT
BEGIN	REAL	FOR	UNTIL

y se considera que están almacenadas en las primeras 20 posiciones de la tabla de símbolos. Observe que bajo este supuesto no es relevante de que palabras se trate, ya que la única información que se requiere es el número total de palabras reservadas.

Tabla de componentes léxicos

La tabla de componentes léxicos incluye tres columnas: componente, clase y tipo.

El tipo de componente se usa para diferenciar componentes de una misma clase, como en el caso de los operadores aritméticos. Para las palabras reservadas, identificadores y constantes, el tipo especifica la dirección donde se encuentra almacenado el componente en la tabla de símbolos (Dir TS).

Componente léxico	Clase	Tipo
Palabra reservada	1	Dir TS
Identificadores	2	Dir TS
Constante entera	3	Dir TS
+	4	1
-	4	2
*	4	3
/	4	4
<	5	1
<=	5	2
<>	5	3
>	5	4
>=	5	5
=	5	6
:=	6	0
..	7	0



DESARROLLO:

1. Creamos un proyecto llamado **LEXICO_GMS** y dentro de ella creamos 3 paquetes (**principal**, **error** y **tabla_de_símbolos**), dentro del paquete principal crearemos una clase llamada **Analizador_Léxico** y codificaremos como se muestra a continuación:

```
Start Page x AnalizadorLexico.java x
Source History
1 package Principal;
2 //Importamos librerias
3 import error.ErrorLexico;
4 import error.Error;
5 import tablaDeSimbolos.Simbolos;
6 import java.io.BufferedReader;
7 import java.io.FileNotFoundException;
8 import java.io.FileReader;
9 import java.io.IOException;
10 import java.util.ArrayList;
11 import java.util.List;
12 import javax.swing.JOptionPane;
13 //creamos la clase AnalizadorLexico
14 public class AnalizadorLexico {
15     private BufferedReader LeerTexto;
16     //creamos variables
17     private String linea, caracter, constanteEntera, identificador, literales, palabra = "";
18     //creamos un comentario vacio
19     private boolean comentario = false;
20     //Creamos
21     //creamos un arraylist para los diferentes tipos de texto
22     private final List delimitadores = new ArrayList();
23     private final List OperadorRelacional = new ArrayList();
24     private final List PalabraReservada = new ArrayList();
25     private final List OperadorAsignado = new ArrayList();
26     private final List<Elemento> tokens = new ArrayList();
27     //creamos una variable para llamar a la ubicacion del archivo a leer
28     private String nombreDelArchivo;
29     //creamos una subclase para la lectura de texto del archivo a leer
30     public AnalizadorLexico(String pathFile){
31         try {
32             int i = 10;
33             //
34             this.nombreDelArchivo = pathFile;
35             constanteEntera = ("^\\d+|^\\d+\\.?\\d+");
36             identificador = ("^\\d\\w+|^\\d\\w?$");
37             literales = ("^\\\".+\\\"$");
38             OperadorRelacional.add("<="); OperadorRelacional.add("<>"); OperadorRelacional.add("<");
39             OperadorRelacional.add(">="); OperadorRelacional.add(">"); OperadorRelacional.add("=");
40             OperadorRelacional.add("!="); OperadorRelacional.add(";");
41             PalabraReservada.add("PROGRAM"); PalabraReservada.add("ARRAY"); PalabraReservada.add("VAR");
42             PalabraReservada.add("CONST"); PalabraReservada.add("BEGIN"); PalabraReservada.add("END");
43             PalabraReservada.add("INTEGER"); PalabraReservada.add("IF"); PalabraReservada.add("THEN");
44             PalabraReservada.add("REAL"); PalabraReservada.add("STEP"); PalabraReservada.add("CASE");
45             PalabraReservada.add("ELSE"); PalabraReservada.add("CHAR"); PalabraReservada.add("FOR");
46             PalabraReservada.add("TO"); PalabraReservada.add("DO"); PalabraReservada.add("WHILE");
47             PalabraReservada.add("REPEAT"); PalabraReservada.add("UNTIL");
48             delimitadores.add(" "); delimitadores.add(","); delimitadores.add("==");
49             delimitadores.add("!="); delimitadores.add("("); delimitadores.add(")");
50             delimitadores.add("\\n"); delimitadores.add("("); delimitadores.add(")");
```



REPORTE DE ACTIVIDAD

```
51     OperadorAsignado.add("?");   OperadorAsignado.add("+=");   OperadorAsignado.add("-=");
52     OperadorAsignado.add("*=");   OperadorAsignado.add("/=");   OperadorAsignado.add("%=");
53     OperadorAsignado.add("++");   OperadorAsignado.add("--");
54     //Vamos a mandar a llamar a nuestro archivo de texto
55     LeerTexto = new BufferedReader(new FileReader(pathFile));
56     //Agregamos una excepcion
57     } catch (FileNotFoundException ex) {
58         //Si no se carga el archivo entonces mostrara el siguiente mensaje
59         JOptionPane.showMessageDialog(null, "Archivo no encontrado");
60     }
61 }
62 // Creamos la clase analizarLexema
63 public void analizarLexema() throws IOException{
64     //comenzamos en linea vacia
65     int nLinea = 0;
66     while (true){
67         //se incrementa el numero de lineas
68         nLinea++;
69         //llamamos a linea y leemos linea por linea
70         linea = LeerTexto.readLine();
71         if (linea == null)
72             break;
73         int size = linea.length();
74         linea = linea.split("\r")[0];
75         caracter = "";
76         //ciclo for para la lectura de textos
77         for (int i = 0; i < size;i++){
78             caracter = linea.substring(i,i+1);
79             if (delimitadores.contains(caracter)){
80                 if ((!comentario) && (palabra.length() >= 2) && (palabra.substring(0,2).equals(
81                     "//"))){
82                     palabra = "";
83                     break;
84                 }
85                 if ((!comentario) && (palabra.length() >= 2) && (palabra.substring(0,2).equals(
86                     "/*"))){
87                     palabra = "";
88                     comentario = true;
89                 }
90                 if ((comentario) && (palabra.length() >= 2) && (palabra.substring(0,2).equals(
91                     "+/"))){
92                     palabra = "";
93                     comentario = false;
94                 }
95                 if (!comentario){
96                     if ((!palabra.equals("")) && (!palabra.contains("/+"))){
97                         this.addToken(palabra,nLinea);
98                         //alineamos el token correspondiente
99                     }
100                    palabra = "";
101                }
102                else
103                    palabra = palabra + caracter;
104            }
105        }
106    }
```



REPORTE DE ACTIVIDAD

```
107 //Imprimimos la constantes enteras
108 private void addToken(String palabra,int nLinea) {
109     if (palabra.matches(constanteEntera)) {
110         Elemento elemento = new Elemento();
111         elemento.setToken("\nCONSTANTE ENTERA: ");
112         elemento.setLexema(palabra);
113         tokens.add(elemento);
114         return;
115     }
116 //Imprimimos la palabras literales
117     if (palabra.matches(literales)) {
118         Elemento elemento = new Elemento();
119         elemento.setToken("\nLITERALES ");
120         elemento.setLexema(palabra);
121         tokens.add(elemento);
122         return;
123     }
124 //Imprimimos los operadores relacionales
125     if (OperadorRelacional.contains(palabra)) {
126         Elemento elemento = new Elemento();
127         elemento.setToken("\nOPERADOR RELACIONAL ");
128         elemento.setLexema(palabra);
129         tokens.add(elemento);
130         return;
131     }
132 //Imprimimos las palabras reservadas
133     if (PalabraReservada.contains(palabra)) {
134         Elemento elemento = new Elemento();
135         elemento.setToken("\nOPERADOR RESERVADA: ");
136         elemento.setLexema(palabra);
137         tokens.add(elemento);
138         return;
139     }
140 //Imprimimos los operadores asignados
141     if (OperadorAsignado.contains(palabra)) {
142         Elemento elemento = new Elemento();
143         elemento.setToken("\nOPERADOR ASIGNADO ");
144         elemento.setLexema(palabra);
145         tokens.add(elemento);
146         return;
147     }
}
```



REPORTE DE ACTIVIDAD

```
148 //Imprimimos los operadores aritmeticos
149     if(!palabra.equals(PalabraReservada)){
150         if (palabra.matches(identificador)){
151             Elemento elemento = new Elemento();
152             elemento.setToken("\nOPERADOR ARITMETICO ");
153             elemento.setLexema(palabra);
154             tokens.add(elemento);
155             Simbolos simbolo = new Simbolos();
156             simbolo.setNome(palabra);
157             Simbolos.addSimbolo(simbolo);
158             return;
159         }
160     }
161 //Imprimimos los identificadores desconocidos
162     Error error = new ErrorLexico();
163     error.setCodigo(101);
164     error.setDescripcion("IDENTIFICADORES DESCONOCIDOS: " + palabra);
165     error.setNombreArchivo(this.nombreDelArchivo);
166     error.setNumLinea(nLinea);
167     Error.addError(error);
168 }
169 //arraylist de los tokens
170 public List<Elemento> getTokens(){
171     return tokens;
172 }
173 //metodo para iniciar nuestro programa
174 public static void main(String[] args) {
175     //llamamos al archivo que vamos a analizar
176     AnalizadorLexico analizador = new AnalizadorLexico("Texto.txt");
177     try {
178         analizador.analizarLexema();
179         System.out.println("INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE");
180         System.out.println("LENGUAJES Y AUTOMATAS 1 || PROGRAMA: ANALIZADOR LEXICO");
181         System.out.println("DOCENTE: DR. TANIA TURRUBIATES LOPEZ");
182         System.out.println("PERIODO ESCOLAR: FEB 2023 - JUN 2023 || GRUPO: 681A");
183         System.out.println("ALUMNO: GONZALO MARTINEZ SILVERIO\n"+ analizador.getTokens());
184         System.out.println("\n\nLista de errores léxicos");
185         //mostramos los errores
186         int i;
187         for (i = 0; i < Error.getErrores().size(); i++){
188             Error error = Error.getErrores().get(i);
189             //mostramos los errores
190             System.out.println(error.showErrores());
191         }
192         //mostramos los simbolos
193         System.out.println("\n\nTABLA DE SIMBOLOS\n" + Simbolos.getTablaDeSimbolos());
194     } //excepcion
195     catch (IOException ex) {
196         //si no se puede leer el archivo de texto mostrara un mensaje
197         JOptionPane.showMessageDialog(null, "ERROR AL LEER EL ARCHIVO DE TEXTO");
198     }
199 }
200 }
201 }
```



REPORTE DE ACTIVIDAD

2. Ahora vamos a crear la clase elemento el cual contendrá los get/set del lexema.

```
Start Page × Elemento.java ×
Source History
1 package Principal;
2 public class Elemento {
3     private String token;
4     private String lexema = "";
5     public String getToken() {
6         return token;
7     }
8     public void setToken(String token) {
9         this.token = token;
10    }
11    public String getLexema() {
12        return lexema;
13    }
14    public void setLexema(String lexema) {
15        this.lexema = lexema;
16    }
17    public String toString() {
18        if (lexema.isEmpty())
19            return "" + token + "";
20        else
21            return "" + token + "" + lexema + "";
22    }
23 }
```




REPORTE DE ACTIVIDAD

3. Después vamos a crear una clase error en el paquete error. Este contendrá los errores en un arraylist.

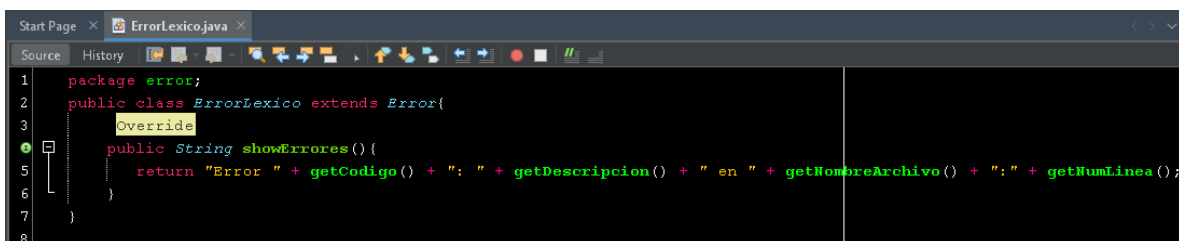
```
Start Page x Error.java x
Source History
1 package error;
2 import java.util.ArrayList;
3 import java.util.List;
4 public abstract class Error {
5     private int codigo;
6     private String descripcion;
7     private int numLinea;
8     private String nombreArchivo;
9     private static final List<Error> errores = new ArrayList();
10
11     public abstract String showErrores();
12
13     public static final void addError(Error error) {
14         Error.errores.add(error);
15     }
16     public static final void limpiarErrores() {
17         Error.errores.clear();
18     }
19     public int getCodigo() {
20         return codigo;
21     }
22     public void setCodigo(int codigo) {
23         this.codigo = codigo;
24     }
25     public String getDescripcion() {
26         return descripcion;
27     }
}
```



REPORTE DE ACTIVIDAD

```
28 public void setDescription(String descripcion) {
29     this.descripcion = descripcion;
30 }
31 public int getNumLinea() {
32     return numLinea;
33 }
34 public void setNumLinea(int numLinea) {
35     this.numLinea = numLinea;
36 }
37 public String getNombreArchivo() {
38     return nombreArchivo;
39 }
40 public void setNombreArchivo(String nombreArchivo) {
41     this.nombreArchivo = nombreArchivo;
42 }
43 public static List<Error> getErrores() {
44     return errores;
45 }
46 }
```

4. Creamos la clase ErrorLexico el cual nos permite obtener los simbolos desconocidos en nuestro lexema.



```
1 package error;
2 public class ErrorLexico extends Error{
3     override
4     public String showErrores(){
5         return "Error " + getCodigo() + ": " + getDescripcion() + " en " + getNombreArchivo() + ":" + getNumLinea();
6     }
7 }
8
```



5. Creamos la clase simbolos en el paquete elementos, este contendrá la tabla de simbolos en un arraylist y contendrá los get/set.

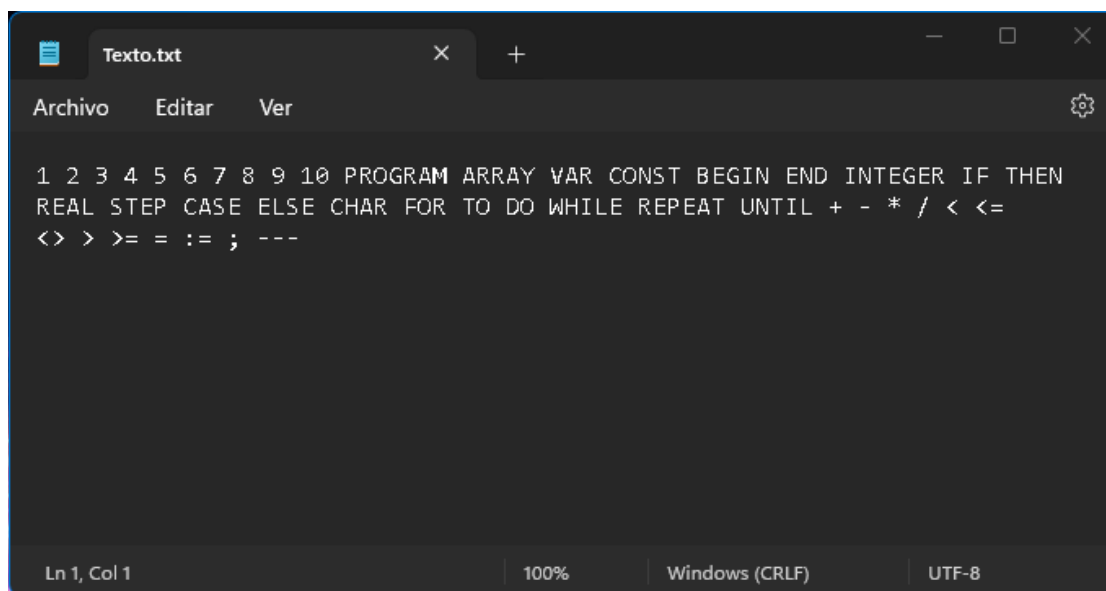
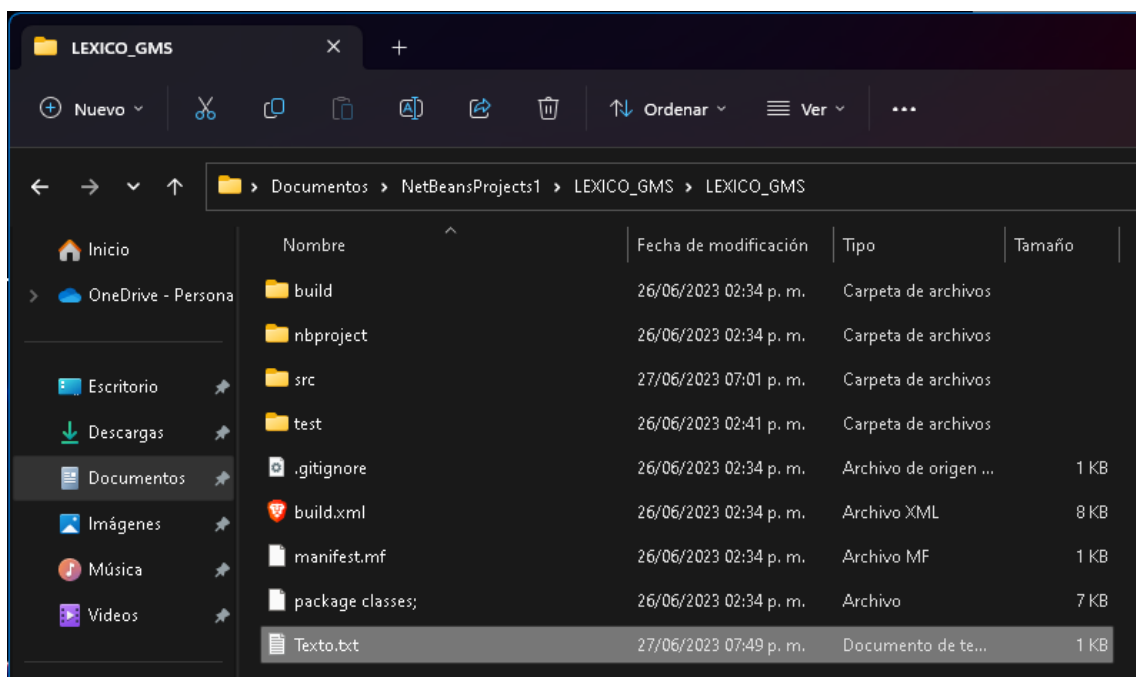
```
Start Page x Simbolos.java x
Source History
1 package tablaDeSimbolos;
2 import java.util.ArrayList;
3 import java.util.List;
4 public class Simbolos {
5     private String nombre;
6     private static final List<Simbolos> tablaDeSimbolos = new ArrayList();
7     public String getNombre() {
8         return nombre;
9     }
10    public void setNome(String nombre) {
11        this.nombre = nombre;
12    }
13    public static void addSimbolo(Simbolos simbolo) {
14        if (!Simbolos.tablaDeSimbolos.contains(simbolo))
15            Simbolos.tablaDeSimbolos.add(simbolo);
16    }
17    public static List<Simbolos> getTabelaDeSimbolos() {
18        return tablaDeSimbolos;
19    }
20    Override
21    public String toString(){
22        return this.getNombre();
23    }
24    Override
25    public boolean equals(Object obj){
26        Simbolos s = (Simbolos) obj;
27        return s.getNombre().equals(this.nombre);
28    }
29 }
30
```



INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE

REPORTE DE ACTIVIDAD

6. Creamos un archivo de texto en el cual insertaremos diversos símbolos y/o palabras las cuales nuestro analizador lexico leera su contenido:





INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE

REPORTE DE ACTIVIDAD

RESULTADOS:

```
run:
INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE
LENGUAJES Y AUTOMATAS 1 || PROGRAMA: ANALIZADOR LEXICO
DOCENTE: DR. TANIA TURRUBIATES LOPEZ
PERIODO ESCOLAR: FEB 2023 - JUN 2023 || GRUPO: 6S1A
ALUMNO: GONZALO MARTINEZ SILVERIO
[
CONSTANTE ENTERA: 1,
CONSTANTE ENTERA: 2,
CONSTANTE ENTERA: 3,
CONSTANTE ENTERA: 4,
CONSTANTE ENTERA: 5,
CONSTANTE ENTERA: 6,
CONSTANTE ENTERA: 7,
CONSTANTE ENTERA: 8,
CONSTANTE ENTERA: 9,
CONSTANTE ENTERA: 10,
OPERADOR RESERVADA: PROGRAM,
OPERADOR RESERVADA: ARRAY,
OPERADOR RESERVADA: VAR,
OPERADOR RESERVADA: CONST,
OPERADOR RESERVADA: BEGIN,
OPERADOR RESERVADA: END,
OPERADOR RESERVADA: INTEGER,
OPERADOR RESERVADA: IF,
OPERADOR RESERVADA: THEN,
OPERADOR RESERVADA: REAL,
OPERADOR RESERVADA: STEP,
OPERADOR RESERVADA: CASE,
OPERADOR RESERVADA: ELSE,
OPERADOR RESERVADA: CHAR,
OPERADOR RESERVADA: FOR,
OPERADOR RESERVADA: TO,
OPERADOR RESERVADA: DO,
OPERADOR RESERVADA: WHILE,
OPERADOR RESERVADA: REPEAT,
OPERADOR RESERVADA: UNTIL,
OPERADOR ARITMETICO +,
OPERADOR ARITMETICO -,
OPERADOR ARITMETICO *,
OPERADOR ARITMETICO /,
OPERADOR RELACIONAL <,
OPERADOR RELACIONAL <=,
OPERADOR RELACIONAL <>,
OPERADOR RELACIONAL >,
OPERADOR RELACIONAL >=,
OPERADOR RELACIONAL =,
OPERADOR RELACIONAL :=,
OPERADOR RELACIONAL ;]

LISTA DE ERRORES LEXICOS
Error 101: IDENTIFICADORES DESCONOCIDOS: --- En el archivo Texto.txt || Linea: 1

TABLA DE SIMBOLOS
[+, -, *, /]
BUILD SUCCESSFUL (total time: 0 seconds)
```



CONCLUSIÓN:

En este tema hemos entendido como funciona un analizador léxico y cómo se relaciona con el analizador sintáctico y con otras estructuras necesarias, como la tabla de símbolos. Para hacernos una idea más completa hemos visto todas las funciones que debe llevar a cabo el analizador léxico con el fichero de entrada que contiene el código fuente. Además, hemos conocido que ventajas aporta a un compilador. También he conocido cómo se diseña un analizador léxico por medio de una tabla o un diagrama de transiciones, representando de esta forma los estados por los que pasa el analizador para reconocer un token.

Posteriormente, hemos aprendido mediante un ejemplo a reconocer un identificador, y esto puede extenderse a los números enteros, operadores, comentarios, palabras reservadas, etc.

Si observamos con más detalle lo que pasa en el analizador léxico, en su relación con el analizador sintáctico y con la tabla de símbolos, vemos que una vez empieza a leer el código fuente y reconoce el primer token, se lo envía al analizador sintáctico y este, en cuanto lo recibe, le pide el siguiente token para que siga reconociendo la entrada. Por tanto, los tokens son enviados al analizador sintáctico bajo demanda.

Por otro lado, si reconoce un identificador lo almacena en la tabla de símbolos, y posteriormente, si el analizador sintáctico reconoce que ese identificador lleva asociada información de tipo (entero, real, etc.) o de valor, también añade esta información a la mencionada tabla.

En cuanto al sistema de gestión de errores, se encarga de detectar símbolos que no pertenezcan a la gramática porque no encajen con ningún patrón. Bien porque haya caracteres inválidos, ejemplo @, o bien porque se escriban mal las palabras reservadas del lenguaje, los identificadores o los números, como 5.25 en lugar de 5,25, pudiendo simplemente no hacer nada o bien informar del tipo de error que se ha cometido. Se puede minimizar el número de errores borrando caracteres inválidos, insertando el carácter que falta o reemplazando un carácter por otro según sea el caso.



BIBLIOGRAFÍA:

1. Aho, Sethi, Ullman, Compiladores Principios, técnicas y herramientas, Ed. Addison Wesley.
2. Hopcroft John E., Introducción a la Teoría de Autómatas, Lenguajes y Computación, 2da ed, Ed. Addison Wesley, 2004.
3. Lemote Karen A. , Fundamentos de compiladores Cómo traducir al lenguaje de computadora, Ed. Compañía Editorial Continental.
4. Martin John, Lenguajes formales y teoría de la computación, Ed. Mc Graw Hill.
5. Kelley, Dean, Teoría de Automatas y Lenguajes Formales, Prentice Hall.
6. Brookshear. Teoría de la Computación, Lenguajes Formales, Autómatas y Complejidad. Addison Wesley.
7. Isasi, Martínez y Borrajo. Lenguajes, Gramáticas y Autómatas. Addison Wesley.
8. Dr. Sergio Gálvez Rojas y Miguel Ángel Mora Mata ,Compiladores “Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC”, , <http://www.lcc.uma.es/~galvez/Compiladores.html>, 3/nov/2009