



**Facultad: Ingeniería**  
**Escuela: Ingeniería en Computación**  
**Asignatura: Autómatas y Compiladores**

## Análisis Léxico

### Contenido

En la presente guía se dará a conocer la manera en que un compilador realiza el análisis léxico de un programa fuente.

### Objetivos Específicos

- Simular un compilador, tomando en cuenta únicamente el análisis léxico.
- Identificar la forma en que un compilador realiza el análisis léxico de un programa.

### Material y Equipo

- Guía de laboratorio N° 5.
- Computadora con Netbeans 7 o superior.

### Introducción Teórica

#### Léxico

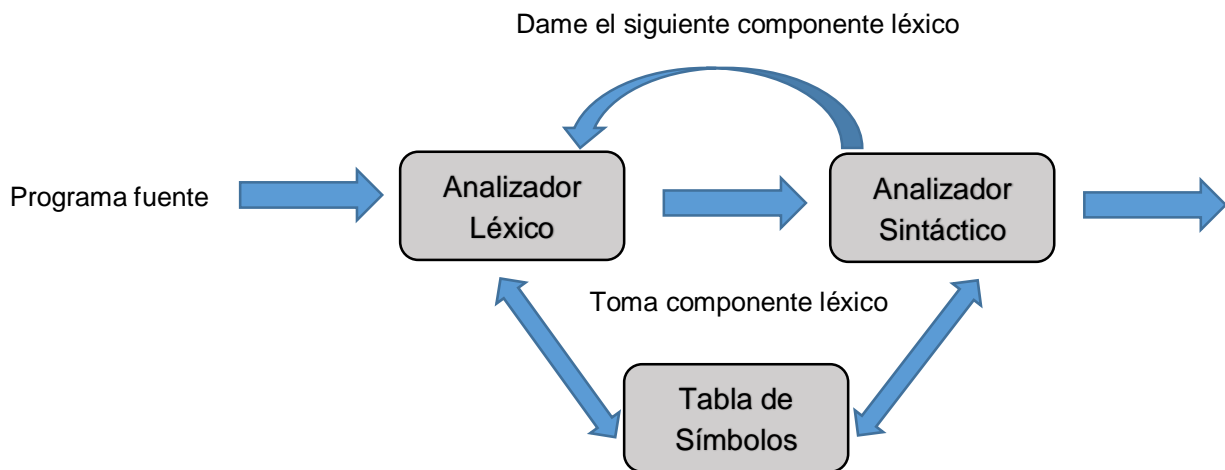
El léxico de un lenguaje de programación u otro lenguaje usado en informática está constituido por todas las palabras y símbolos que lo componen. En los lenguajes de programación, el léxico lo constituyen los elementos individuales del lenguaje denominados **tokens**. Los **tokens** son todas las palabras reservadas del lenguaje, los símbolos que denotan los distintos tipos de operadores, identificadores (de variables, de funciones, de procedimientos, de tipos, etc.), separadores de sentencias, y otros símbolos empleados en las distintas instrucciones del lenguaje.

#### Analizador Léxico

Al analizador léxico (scanner): lee la secuencia de caracteres del programa fuente, carácter a carácter, y los agrupa para formar unidades con significado propio, los componentes léxicos (tokens en inglés). Estos componentes léxicos representan:

- ♣ Palabras reservadas: if, while, do, ...
- ♣ Identificadores asociados a variables, nombres de funciones, tipos definidos por el usuario, etiquetas, etc.
- ♣ Operadores: =, +, -, /, ==, <, >, &, !=, ...
- ♣ Símbolos especiales: ;, (, ), [ ], { }, ...
- ♣ Constantes numéricas: literales que representan valores enteros, en coma flotante, etc.
- ♣ Constantes de caracteres: literales que representan cadenas de caracteres, por ejemplo: "estoy en labo de compi",...

El analizador léxico opera bajo petición del analizador sintáctico devolviendo un componente léxico conforme el analizador sintáctico lo va necesitando para avanzar en la gramática. Los componentes léxicos son los símbolos terminales de la gramática. Suele implementarse como una subrutina del analizador sintáctico. Cuando recibe la orden "obtén el siguiente componente léxico" el analizador léxico lee los caracteres de entrada hasta identificar el siguiente componente léxico.



### Procedimiento

Siguiendo con el estudio del analizador léxico, se puede representar de la siguiente manera:

(Expresión regular 1)	(Acción 1)
(Expresión regular 2)	(Acción 2)
(Expresión regular 3)	(Acción 3)
.	.
.	.
.	.
(Expresión regular n)	(Acción n)

Donde cada acción es un fragmento de programa que describe cual ha de ser la acción del analizador léxico cuando la secuencia de entrada coincida con la expresión regular.

### ¿Qué es una expresión regular?

Son una serie de caracteres que forman un patrón, normalmente representativo de otro grupo de caracteres mayor, de tal forma que podemos comparar el patrón con otro conjunto de caracteres para encontrar las coincidencias.

Las expresiones regulares están disponibles en casi cualquier lenguaje de programación, pero, aunque su sintaxis es relativamente uniforme, cada lenguaje usa su propio dialecto.

- ♣ **Patrón:** es una expresión regular.
- ♣ **Token:** es el terminal asociado a un patrón. Utilizamos la palabra terminal desde el punto de vista de la gramática utilizada por el analizador sintáctico.
- ♣ **Lexema:** es cada secuencia de caracteres que encaja con un patrón, es decir, es como instancia de un patrón.

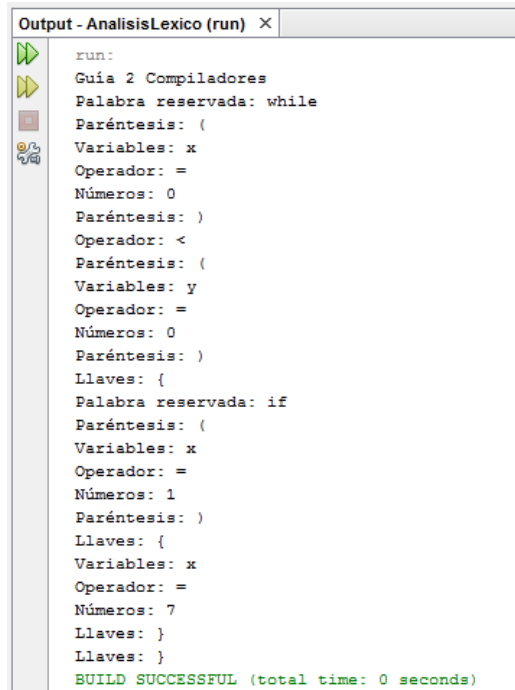
A continuación, se presenta una aplicación capaz de evaluar el léxico utilizado en una expresión que corresponde a un lenguaje de programación. Dicho programa hace las respectivas distinciones entre cada elemento que forma parte de las expresiones (variables, palabras reservadas, símbolos, números, etc.)

```

1  package analisislexico;
2  //Importamos paquete para poder trabajar con expresiones regulares
3  import java.util.regex.Matcher;
4  //Importamos paquete para trabajar con un patrón
5  import java.util.regex.Pattern;
6  /**
7   * @author HSerrano
8   */
9  public class AnalisisLexico {
10     /**
11      * @param args the command line arguments
12      */
13     public static void main(String[] args) {
14         // TODO code application logic here
15         String patron = ("(while|if)|([a-zA-Z]+)|(>|<|=|+)|([0-9]+)|([(){}]|(;))");
16         //La expresión que vamos a evaluar es la siguiente:
17         String Texto = "while (x=0) < (y=0) {if (x=1) {x=7}}";
18
19         Pattern p = Pattern.compile(patron);
20         Matcher matcher = p.matcher(Texto);
21
22         System.out.println("Guía 2 Compiladores");
23         while(matcher.find())
24         {
25             String tokenTipo1 = matcher.group(1);
26             if(tokenTipo1 != null)
27             {
28                 System.out.println("Palabra reservada: "+tokenTipo1);
29             }
30
31             String tokenTipo2 = matcher.group(2);
32             if(tokenTipo2 != null)
33             {
34                 System.out.println("Variables: "+tokenTipo2);
35             }
36
37             String tokenTipo3 = matcher.group(3);
38             if(tokenTipo3 != null)
39             {
40                 System.out.println("Operador: "+tokenTipo3);
41             }
42
43             String tokenTipo4 = matcher.group(4);
44             if(tokenTipo4 != null)
45             {
46                 System.out.println("Números: "+tokenTipo4);
47             }
48
49             String tokenTipo5 = matcher.group(5);
50             if(tokenTipo5 != null)
51             {
52                 System.out.println("Paréntesis: "+tokenTipo5);
53             }
54
55             String tokenTipo6 = matcher.group(6);
56             if(tokenTipo6 != null)
57             {
58                 System.out.println("Llaves: "+tokenTipo6);
59             }
60
61             String tokenTipo7 = matcher.group(7);
62             if(tokenTipo7 != null)
63             {
64                 System.out.println("Punto y coma: "+tokenTipo7);
65             }
66
67         }
68     }
69 }

```

La salida que se obtiene del programa debería ser la siguiente:



```

run:
Guía 2 Compiladores
Palabra reservada: while
Paréntesis: (
Variables: x
Operador: =
Números: 0
Paréntesis: )
Operador: <
Paréntesis: (
Variables: y
Operador: =
Números: 0
Paréntesis: )
Llaves: {
Palabra reservada: if
Paréntesis: (
Variables: x
Operador: =
Números: 1
Paréntesis: )
Llaves: {
Variables: x
Operador: =
Números: 7
Llaves: }
Llaves: }
BUILD SUCCESSFUL (total time: 0 seconds)

```

La idea es que el analizador léxico reconozca cada una de las representaciones que se encuentran en las expresiones del lenguaje de programación.

Cada vez que se desee hacer más robusto nuestro analizador debemos agregar más patrones en nuestro código.

### Ejercicios:

1. Modificar el ejemplo de manera que se pueda evaluar la expresión:

```

x=1;
do{
  if (w==1)
  {
    while(x<5)
    {
      z++;
    }
  }
  else{
    Array[i] = 7;
  }
}while(j==0);

```

2. Modificar el ejemplo de manera que acepte más palabras reservadas, por ejemplo: for, foreach, bool, int, double, char, string.

### Investigación Complementaria

1. Investigar qué es un analizador sintáctico y los elementos que lo componen.

### Bibliografía

- Manuel Alfonseca Moreno, Marina de la Cruz Echeandía, Alfonso Ortega de la Puente, Estrella Pulido Cañabate, Compiladores: Teoría y Práctica, Pearson Educación, S.A., Madrid, 2006