



# INSTITUTO TECNOLÓGICO DE MORELIA

## División De Estudios Profesionales

Departamento De Sistemas Y Computación

## LENGUAJES Y AUTÓMATAS II

Unidad I: Análisis semántico.

### PRÁCTICA 1: ANALIZADOR LÉXICO

PROFESOR

BALLESTEROS LEÓN JAVIER

ALUMNOS

Ascencio Flores Ricardo - 15120269

Gil Jacome Arturo - 14121426

Martínez Aguilar Diego Ulises - 15121185

**MORELIA, MICHOACÁN, MÉXICO**

Lunes, 28 de enero del 2019

## Tabla de contenido

---

INTRODUCCIÓN .....	3
<b>PRÁCTICA NO.1</b> .....	4
CREAR UN ANALIZADOR LÉXICO QUE RECONOZCA LOS SIGUIENTES LEXEMAS: .....	4
DESARROLLO DE LA PRÁCTICA .....	4
CONCLUSIONES .....	8
REFERENCIAS .....	8

## INTRODUCCIÓN

El **analizador léxico** es la primera fase de un compilador. Su principal función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis.

El analizador léxico (*scanner*) se encarga de suministrar al analizador sintáctico una serie de unidades lógicas llamadas elementos léxicos que resultan de agrupar los caracteres del fichero de entrada. Cada uno de estos elementos léxicos se denomina token. Consistente en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos. Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico (*parser*).

La especificación de un lenguaje de programación a menudo incluye un conjunto de reglas que definen el léxico. Estas reglas consisten comúnmente en expresiones regulares que indican el conjunto de posibles secuencias de caracteres que definen un token o lexema (unidad mínima con significado).

El analizador léxico es una función o método que es llamado por el analizador sintáctico cada vez que necesita conocer un nuevo token para continuar el proceso de traducción. Es el único módulo del compilador que maneja el fichero de entrada.

La herramienta que vamos a usar para definir nuestro lenguaje, y por consiguiente su intérprete, va a ser el metacompilador JavaCC.

**JavaCC** es una herramienta que generalmente se utiliza para generar analizadores léxicos y sintácticos (parsers) de una forma muy cómoda, actualmente se encuentra mantenida por Sun Microsystems y es muy robusta, eficiente y fiable.

Un metacompilador o generador de parsers es una herramienta que, a partir de la especificación de un lenguaje, construye un programa o analizador que es capaz de reconocer secuencias o elementos de dicho lenguaje. En general, la especificación del lenguaje abarca tanto el aspecto léxico como el sintáctico, y son los que permiten la construcción del parser, mientras que el aspecto semántico del lenguaje se deja en manos del usuario, para que lo ensamble una vez obtenido el parser.

JavaCC integra las funciones de análisis léxico y análisis sintáctico en una sola herramienta, obteniendo a la salida código java.

## PRÁCTICA NO.1

### CREAR UN ANALIZADOR LÉXICO QUE RECONOZCA LOS SIGUIENTES LEXEMAS:

- Letra minúscula ["a"-"z"].
- Letra mayúscula ["A"-"Z"].
- Dígitos del **0** al **9** de cualquier longitud.
- Paréntesis que abre y cierra ( ).
- Llave que abre y cierra { }.
- Operadores: +, -, /, \*, ^.
- Dígito binario: comenzando con **B** y siguiendo con **0** y/o **1**.
- Dígito octal: comenzando por letra **O** seguidos con dígitos del **0** al **7**.
- Dígito hexadecimal: **0x** seguido de números del **0** al **9** o letras de **A** hasta **F**, permite minúsculas.
- Identificador variable: **L(N)+\_** con un máximo de 64 caracteres. Todos los identificadores son una secuencia de letras (a-zA-Z) y números que obligatoriamente deben comenzar con una letra, además, pueden incluir un guion bajo.
- Tipo de datos: "**byte**", "**short**", "**int**", "**long**", "**float**", "**double**", "**char**", "**boolean**".

### DESARROLLO DE LA PRÁCTICA

```
Practica1.jj  x  Prueba.txt  x  Comandos.txt  x
1  options { Ignore_Case = false; }
2
3  PARSER_BEGIN (Practical)
4  public class Practical {
5      //Ejecución del analizador
6      public static void main (String[] argum) throws ParseException {
7          //Inicialización del analizador
8          Practical anLexSint = new Practical (System.in);
9          anLexSint.expression();
10         System.out.println("Análisis terminado");
11     }
12 }
13
14 PARSER_END (Practical)
15
```

**Figura 1:** Se crea un archivo nuevo con extensión **.jj** y se establece la estructura necesaria para la ejecución e inicialización del analizador.

```
16 //tokens estáticos
17 TOKEN:
18 {
19     <NUM: ([ "0"- "9" ])+> {System.out.println("Numero");} |
20     <TIPODEDATO: "byte"|"short"|"int"|"long"|"float"|"double"|"char"|"boolean"> {System.out.println("Tipo de Dato");} |
21     <BINARIO: "B"([ "0"- "1" ])+> {System.out.println("Binario");} |
22     <OCTAL: "O"([ "0"- "7" ])+> {System.out.println("Octal");} |
23     <HEX: "0x"(<NUM>| [ "A"- "F" ]| [ "a"- "f" ])+> {System.out.println("Hexadecimal");} |
24     <VAR: (<LETRAMIN>|<LETRAMAY>)(<LETRAMIN>|<LETRAMAY>| [ "0"- "9" ]| " "){0,63}> {System.out.println("Identificador Variable");} |
25     <LETRAMIN: [ "a"- "z" ]> {System.out.println("Letra Minuscula");} |
26     <LETRAMAY: [ "A"- "Z" ]> {System.out.println("Letra Mayuscula");} |
27     <GUION: " _" > {System.out.println("Guion bajo");} |
28     <ABRIRPARENTESIS: "(" > {System.out.println("Abrir Parentesis");} |
29     <CERRARPARENTESIS: ")" > {System.out.println("Cerrar Parentesis");} |
30     <ABRIRLLAVE: "{" > {System.out.println("Abrir Llave");} |
31     <CERRARLLAVE: "}" > {System.out.println("Cerrar Llave");} |
32     <OPSUMA: "+" > {System.out.println("Operador Suma");} |
33     <OPRESTA: "-" > {System.out.println("Operador Resta");} |
34     <OPPOTENCIA: "^" > {System.out.println("Operador Potencia");} |
35     <OPMULTIPLICACION: "*" > {System.out.println("Operador multiplicacion");} |
36     <OPDIVISION: "/" > {System.out.println("Operador Division");}
37 }
38 }
```

**Figura 2:** Se establecen los tokens estáticos que nos permitirán reconocer los los lexemas establecidos.

```
38 //Estructuras y caracteres de escape
39 SKIP:
40 { " " | "\t" | "\n" | "\r" }
41
42 //Unidad principal
43 void expression() : {}
44 {
45     (<NUM>|<LETRAMIN>|<LETRAMAY>|<ABRIRPARENTESIS>|<CERRARPARENTESIS>|<ABRIRLLAVE>|<CERRARLLAVE>|<OPSUMA>|<OPRESTA>|<OPPOTENCIA>|
46     <OPMULTIPLICACION>|<OPDIVISION>|<BINARIO>|<OCTAL>|<HEX>|<VAR>|<TIPODEDATO>)+ <EOF>
47 }
48 }
```

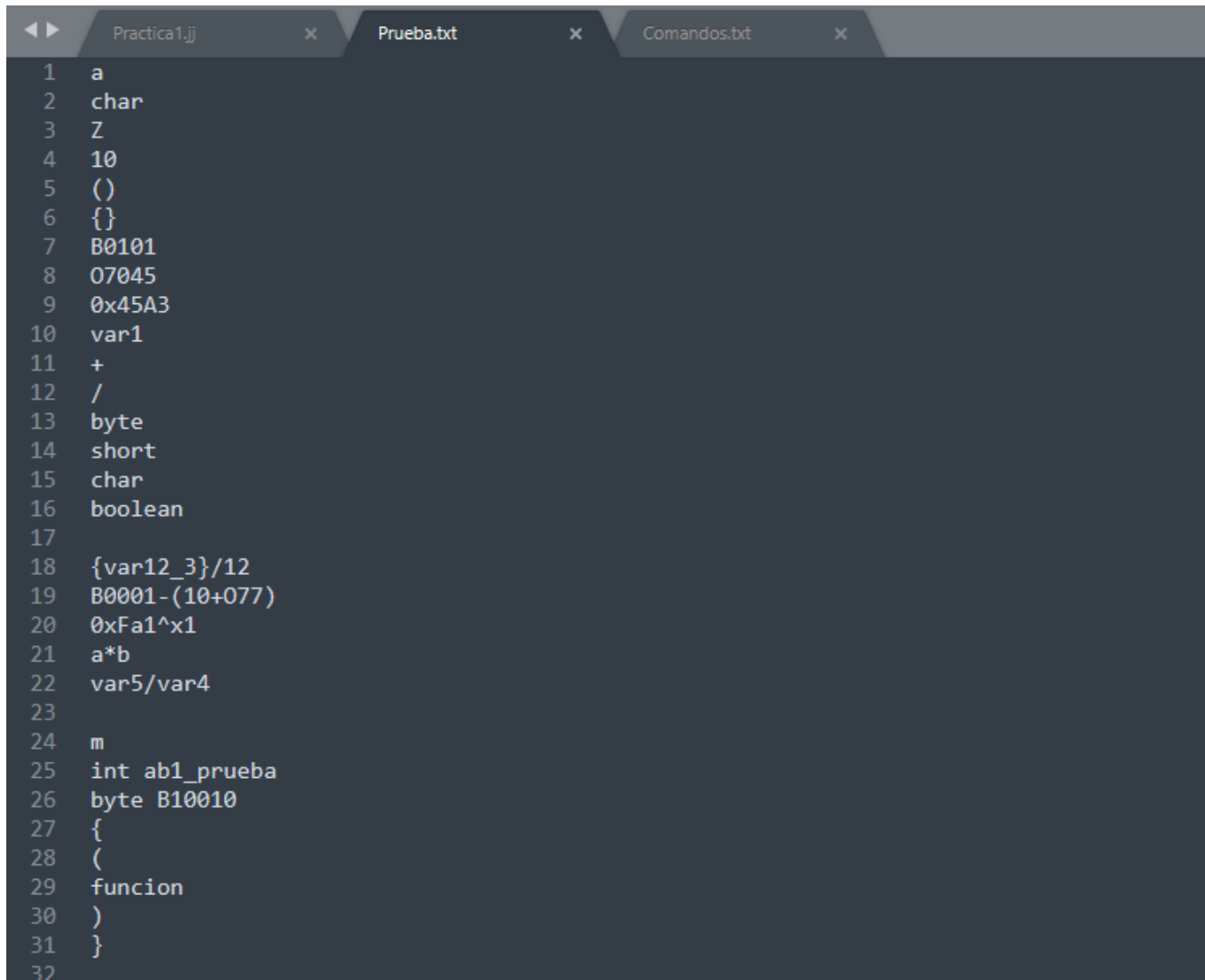
**Figura 3:** Se establecen las estructuras de escape, así como también se indica la unidad principal como aquella que va a aceptar la gramática establecida.

```
C:\Users\DiegoUlises\Documents\8vo Semestre\Automatas II\Unidad 1\Practica 1>javacc Practica1.jj
Java Compiler Version 5.0 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file Practica1.jj . . .
File "TokenMgrError.java" is being rebuilt.
File "ParseException.java" is being rebuilt.
File "Token.java" is being rebuilt.
File "SimpleCharStream.java" is being rebuilt.
Parser generated successfully.

C:\Users\DiegoUlises\Documents\8vo Semestre\Automatas II\Unidad 1\Practica 1>javac *.java

C:\Users\DiegoUlises\Documents\8vo Semestre\Automatas II\Unidad 1\Practica 1>
```

**Figura 4:** Para compilar este fichero, se debe hacer con "javacc" y posteriormente con "javac".



```
1 a
2 char
3 Z
4 10
5 ()
6 {}
7 B0101
8 07045
9 0x45A3
10 var1
11 +
12 /
13 byte
14 short
15 char
16 boolean
17
18 {var12_3}/12
19 B0001-(10+077)
20 0xFa1^x1
21 a*b
22 var5/var4
23
24 m
25 int ab1_prueba
26 byte B10010
27 {
28 (
29 funcion
30 )
31 }
32
```

**Figura 5:** En un archivo de prueba se crea con extensión **.txt** el cual será evaluado por el analizador léxico antes creado, tal archivo contiene ejemplos en desorden de cada uno de los lexemas.

```

C:\Users\DiegoUlises\Documents\8vo Semestre\Automatas II\Unidad 1\Practica 1>java Practica1 < prueba.txt
Letra Minuscula
Tipo de Dato
Letra Mayuscula
Numero
Abrir Parentesis
Cerrar Parentesis
Abrir Llave
Cerrar Llave
Binario
Octal
Hexadecimal
Identificador Variable
Operador Suma
Operador Division
Tipo de Dato
Tipo de Dato
Tipo de Dato
Tipo de Dato
Abrir Llave
Identificador Variable
Cerrar Llave
Operador Division
Numero
Binario
Operador Resta
Abrir Parentesis
Numero
Operador Suma
Octal
Cerrar Parentesis
Hexadecimal
Operador Potencia
Identificador Variable
Letra Minuscula
Operador multiplicacion
Letra Minuscula
Identificador Variable
Operador Division
Identificador Variable
Letra Minuscula
Tipo de Dato
Identificador Variable
Tipo de Dato

```

**Figura 6:** Para ejecutar el programa y dentro del símbolo del sistema con los comandos de **java Practica1 < prueba.txt** con el cual comprobamos el funcionamiento del analizador léxico, no trata de comprobar las estructuras del lenguaje ni compilará el fichero en código binario.

## CONCLUSIONES

Un analizador léxico es un módulo destinado a leer caracteres del archivo de entrada, donde se encuentra la cadena a analizar, reconocer subcadenas que correspondan a símbolos del lenguaje y retornar los tokens correspondientes y sus atributos.

Los Tokens en los archivos de gramática siguen las mismas convenciones que para el lenguaje de programación Java. Por tanto, identificadores, cadenas, caracteres, etc.

Utilizada en las gramáticas son iguales que los identificadores de Java, cadenas de Java, caracteres de Java, etc.

Los espacios en blanco en los archivos de gramática también siguen las mismas convenciones para el lenguaje de programación Java. Este incluye la sintaxis para comentarios. La mayor parte de los comentarios presentes en los archivos de gramáticas son generados dentro del analizador léxico/sintáctico generado.

Los archivos de gramática son preprocesados por códigos de escape Unicode tal como aparecen en los archivos Java.

## REFERENCIAS

[1] <http://www.lcc.uma.es/~galvez/theme/IntroduccionAJavaCC.pdf>

[2] <https://www.dlsi.ua.es/asignaturas/pl/downloads/1617/tema2-imprimir.pdf>

[3] <http://www.galeon.com/shock/tareas.html>

[4] <http://kiwwito.com/construir-un-analizador-lexico-con-javacc/>

[5] <http://genaromendez.com/cursos/progsist/JavaCC/Documentation/ArchivosJavaCC.pdf>