**A picture containing text, clipart

Description automatically generated**

**UNIVERSIDAD DON BOSCO**

**FACULTAD DE INGENIERÍA**

**ESCUELA DE COMPUTACIÓN**

**CICLO 03-2022**

**“GUIA 5-ANÁLISIS LÉXICO”**

**GRUPO 01L**

**PRESENTADO POR:**

|  |  |  |
| --- | --- | --- |
| **Carnet** | **Nombre** | **Apellido** |
| CG172415 | Josué Alexander | Chávez García |
| HG190072 | Carlos David | Herrera Guardado |
| VC190438 | Gerardo José | Velásquez Cruz |
| VC190544 | Francisco José | Valle Cornejo |

**DOCENTE:**

Herson Miguel Serrano Chacón

Desarrollo de la guía:

//Importamos paquete para poder trabajar con expresiones regulares

import java.util.regex.Matcher;

//Importamos paquete para trabajar con un patrón

import java.util.regex.Pattern;

/\*\*

 \* @author cesco\_valle

 \*/

public class Ejemplo1 {

    /\*\*

     \* @param args the command line arguments

     \*/

    public static void main(String[] args) {

        // TODO code application logic here

        String patron = ("(while|if)|([a-zA-Z]+)|([>|<|=]+)|([0-9]+)|([(|)]+)|([{|}])|(;)");

        // La expresión que vamos a evaluar es la siguiente:

        String texto = "while (x=0) < (y=0) {if (x=1) {x=7}}";

        Pattern p = Pattern.compile(patron);

        Matcher matcher = p.matcher(texto);

        System.out.println("Guia 5 Análisis Léxico");

        while (matcher.find()) {

            String tokenTipo1 = matcher.group(1);

            if (tokenTipo1 != null) {

                System.out.println("Palabras reservadas: " + tokenTipo1);

            }

            String tokenTipo2 = matcher.group(2);

            if (tokenTipo2 != null) {

                System.out.println("Variables: " + tokenTipo2);

            }

            String tokenTipo3 = matcher.group(3);

            if (tokenTipo3 != null) {

                System.out.println("Operador: " + tokenTipo3);

            }

            String tokenTipo4 = matcher.group(4);

            if (tokenTipo4 != null) {

                System.out.println("Números: " + tokenTipo4);

            }

            String tokenTipo5 = matcher.group(5);

            if (tokenTipo5 != null) {

                System.out.println("Paréntesis: " + tokenTipo5);

            }

            String tokenTipo6 = matcher.group(6);

            if (tokenTipo6 != null) {

                System.out.println("Llaves: " + tokenTipo6);

            }

            String tokenTipo7 = matcher.group(7);

            if (tokenTipo7 != null) {

                System.out.println("Punto y coma: " + tokenTipo7);

            }

        }

    }

}

Text

Description automatically generated

Ejercicio 1:

//Importamos paquete para poder trabajar con expresiones regulares

import java.util.regex.Matcher;

//Importamos paquete para trabajar con un patrón

import java.util.regex.Pattern;

/\*\*

 \* @author cesco\_valle

 \*/

public class Ejercicio1 {

    /\*\*

     \* @param args the command line arguments

     \*/

    public static void main(String[] args) {

        // TODO code application logic here

        String patron = ("(while|if|else|do)|([a-zA-Z]+)|([>|<|=|+]+)|([0-9]+)|([(|)]+)|([{|}])|(;)|([\\[|\\]])");

        // La expresión que vamos a evaluar es la siguiente:

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

        Pattern p = Pattern.compile(patron);

        Matcher matcher = p.matcher(texto);

        System.out.println("Guia 5 Análisis Léxico");

        while (matcher.find()) {

            String tokenTipo1 = matcher.group(1);

            if (tokenTipo1 != null) {

                System.out.println("Palabras reservadas: " + tokenTipo1);

            }

            String tokenTipo2 = matcher.group(2);

            if (tokenTipo2 != null) {

                System.out.println("Variables: " + tokenTipo2);

            }

            String tokenTipo3 = matcher.group(3);

            if (tokenTipo3 != null) {

                System.out.println("Operador: " + tokenTipo3);

            }

            String tokenTipo4 = matcher.group(4);

            if (tokenTipo4 != null) {

                System.out.println("Números: " + tokenTipo4);

            }

            String tokenTipo5 = matcher.group(5);

            if (tokenTipo5 != null) {

                System.out.println("Paréntesis: " + tokenTipo5);

            }

            String tokenTipo6 = matcher.group(6);

            if (tokenTipo6 != null) {

                System.out.println("Llaves: " + tokenTipo6);

            }

            String tokenTipo7 = matcher.group(7);

            if (tokenTipo7 != null) {

                System.out.println("Punto y coma: " + tokenTipo7);

            }

            String tokenTipo8 = matcher.group(8);

            if (tokenTipo8 != null) {

                System.out.println("Corchetes: " + tokenTipo8);

            }

        }

    }

}

Text

Description automatically generated

Ejercicio 2:

//Importamos paquete para poder trabajar con expresiones regulares

import java.util.regex.Matcher;

//Importamos paquete para trabajar con un patrón

import java.util.regex.Pattern;

/\*\*

 \* @author cesco\_valle

 \*/

public class Ejercicio2 {

    /\*\*

     \* @param args the command line arguments

     \*/

    public static void main(String[] args) {

        // TODO code application logic here

        String patron = ("(while|if|else|do|for|foreach|bool|int|double|char|string)|([a-zA-Z]+)|([>|<|=|+]+)|([0-9]+)|([(|)]+)|([{|}])|(;)|([\\[|\\]])");

        // La expresión que vamos a evaluar es la siguiente:

        String texto = "x=1; do{ if (w==1) { While(x<5) { z++; } } else{ Array[i] = 7; } }while(j==0); int no=5; double des=3.5;bool flag=true; char knife='a'; string='space\_jam';for (int i = 0; i < 5; i++) {i;};String[] cars = {'Volvo', 'BMW', 'Ford', 'Mazda'};for (String i : cars) {i;}";

        Pattern p = Pattern.compile(patron);

        Matcher matcher = p.matcher(texto);

        System.out.println("Guia 5 Análisis Léxico");

        while (matcher.find()) {

            String tokenTipo1 = matcher.group(1);

            if (tokenTipo1 != null) {

                System.out.println("Palabras reservadas: " + tokenTipo1);

            }

            String tokenTipo2 = matcher.group(2);

            if (tokenTipo2 != null) {

                System.out.println("Variables: " + tokenTipo2);

            }

            String tokenTipo3 = matcher.group(3);

            if (tokenTipo3 != null) {

                System.out.println("Operador: " + tokenTipo3);

            }

            String tokenTipo4 = matcher.group(4);

            if (tokenTipo4 != null) {

                System.out.println("Números: " + tokenTipo4);

            }

            String tokenTipo5 = matcher.group(5);

            if (tokenTipo5 != null) {

                System.out.println("Paréntesis: " + tokenTipo5);

            }

            String tokenTipo6 = matcher.group(6);

            if (tokenTipo6 != null) {

                System.out.println("Llaves: " + tokenTipo6);

            }

            String tokenTipo7 = matcher.group(7);

            if (tokenTipo7 != null) {

                System.out.println("Punto y coma: " + tokenTipo7);

            }

            String tokenTipo8 = matcher.group(8);

            if (tokenTipo8 != null) {

                System.out.println("Corchetes: " + tokenTipo8);

            }

        }

    }

}

Text

Description automatically generated

Investigación Complementaria:

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

ANALIZADOR SINTACTICO

Todo lenguaje de programación tiene reglas que describen la estructura sintáctica de programas bien formados.

¿Qué es el analizador sintáctico?

Es la fase del analizador que se encarga de chequear el texto de entrada en base a una

gramática dada. Y en caso de que el programa de entrada sea válido, suministra el árbol

sintáctico que lo reconoce.

En teoría, se supone que la salida del analizador sintáctico es alguna representación del

árbol sintáctico que reconoce la secuencia de tokens suministrada por el analizador léxico.En la práctica, el analizador sintáctico también hace:

* Acceder a la tabla de símbolos (para hacer parte del trabajo del analizador
* semántico).
* Chequeo de tipos (del analizador semántico).
* Generar código intermedio.
* Generar errores cuando se producen.

En definitiva, realiza casi todas las operaciones de la compilación. Este método de trabajo

da lugar a los métodos de compilación dirigidos por sintaxis.

Si un compilador tuviera que procesar sólo programas correctos, su diseño e implantación

se simplificarían mucho. Pero los programadores a menudo escriben programas

incorrectos, y un buen compilador debería ayudar al programador a identificar y localizar

errores. Es más, considerar desde el principio el manejo de errores puede simplificar la

estructura de un compilador y mejorar su respuesta a los errores.

Los errores en la programación pueden ser de los siguientes tipos:

* Léxicos, producidos al escribir mal un identificador, una palabra clave o un
* operador.
* Sintácticos, por una expresión aritmética o paréntesis no equilibrados.
* Semánticos, como un operador aplicado a un operando incompatible.
* Lógicos, puede ser una llamada infinitamente recursiva.

