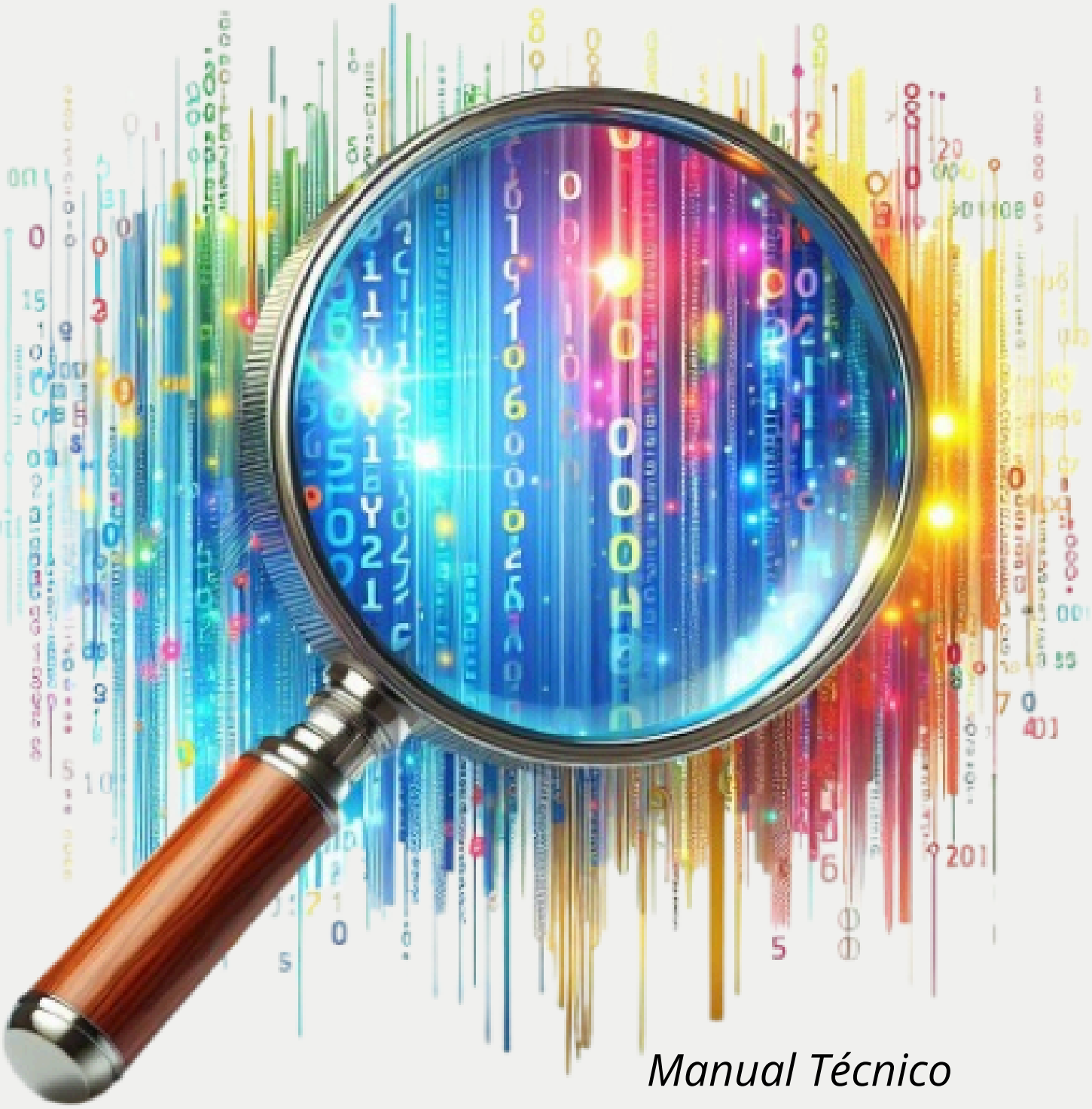
# Analizador Léxico y Sintáctico



*Manual Técnico*

# *Introducción*

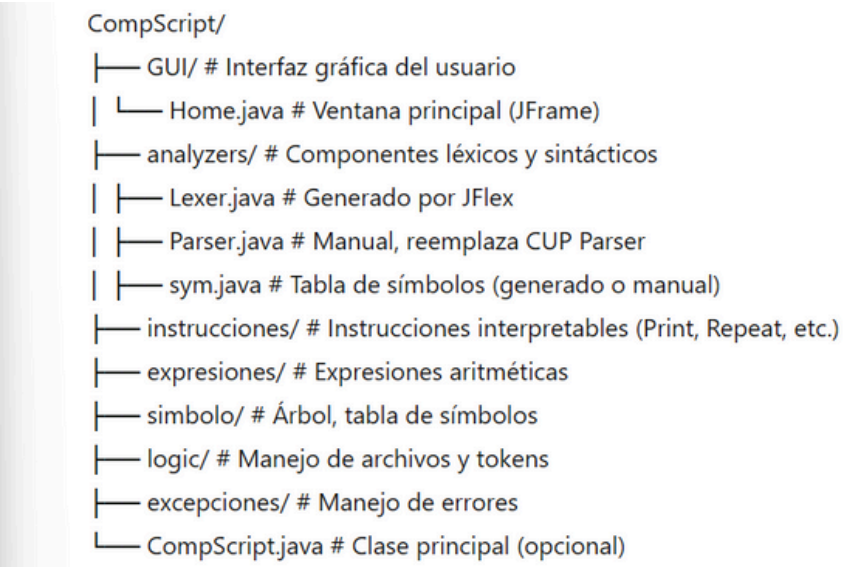
*ProyectoLFP es una aplicación en WPF diseñada para analizar texto y proporcionar un análisis léxico. El programa permite a los usuarios buscar y resaltar palabras específicas dentro de un texto, además de generar reportes sobre las coincidencias encontradas.*

# *Requisitos del sistema:*

- ***Sistema Operativo:*** *Windows 10 o superior*
- ***JDK:*** *Java 17 o superior*
- ***IDE recomendado:*** *Visual Studio Code*
- ***Herramientas necesarias:***
  - ***JFlex 1.9.1***
  - ***Java CUP 11b (20160615)***
- ***Hardware:***
  - ***Procesador:*** *1 GHz o superior*
  - ***Memoria RAM:*** *4 GB mínimo*
  - ***Espacio en disco:*** *500 MB libres para la instalación*

# *Estructura del proyecto*

- *El proyecto está desarrollado bajo una arquitectura básica de carpestas y archivos.*




- ***Codigo fuente: <u>Analizadores</u>***

# Descripción de código

## *Funcionalidad del Sistema*

- *Entrada del usuario desde JTextPane*
- *Análisis Léxico mediante Lexer.java*
- *Análisis Sintáctico mediante Parser.java*
- *Interpretación de instrucciones válidas (e.g., PRINT, REPEAT, IF)*
- *Generación de archivo de salida si no hay errores*

*Visualización de:*

1. *Tabla de Tokens*
2. *Tabla de Errores*
3. *Consola de Salida*

## *Componentes relevantes*

- ***Home.java***
  - *JFrame principal*
  - *Contiene menú Archivo, Reportes, Editar y Ayuda*
  - *Ejecuta análisis léxico y sintáctico desde el botón "Ejecutar"*
- ***Lexer.flex → Lexer.java***
  - *Archivo de reglas léxicas usado por JFlex*
  - *Devuelve tokens reconocidos por CUP (o Parser manual)*
- ***parser.cup (referencia)***
  - *Definía las reglas sintácticas (actualmente parser.java es manual)*
- ***Parser.java***
  - *Simula comportamiento de CUP*
  - *Incluye interpretación e impresión de instrucciones*
  - *Permite construcción dinámica desde la interfaz gráfica.*

# *Métodos importantes:*

## 1. *Buscar_Click:*

***Descripción:*** *Este método se ejecuta cuando el usuario hace clic en el botón de búsqueda. Obtiene la cadena que el usuario desea buscar y la pasa al método BuscarYResaltar para que realice la búsqueda y resalte las coincidencias.*

```
private void buscarYResaltar(String textoBuscar) {
    // Limpiar resaltados anteriores
    limpiarResaltados();

    if (textoBuscar == null || textoBuscar.isEmpty()) {
        return;
    }

    String contenido = codeTextPane.getText();
    Highlighter highlighter = codeTextPane.getHighlighter();
    int pos = 0;
    int contador = 0;

    try {
        while ((pos = contenido.indexOf(textoBuscar, pos)) >= 0) {
            int endPos = pos + textoBuscar.length();
            highlights.add(highlighter.addHighlight(pos, endPos, yellowHighlighter));
            pos = endPos;
            contador++;
        }

        if (contador > 0) {
            terminalTextPane.setText("Se encontraron " + contador + " coincidencias");
        } else {
            terminalTextPane.setText(t:"No se encontraron coincidencias");
        }
    } catch (BadLocationException e) {
        terminalTextPane.setText("Error en la búsqueda: " + e.getMessage());
    }
}
```

## *2. initComponents()*

***Descripción:***

*Método autogenerado por el diseñador de interfaces de Java Swing. Crea y configura los elementos gráficos como JTable, JScrollPane, etc.*

***Importancia:***

*Es fundamental para que la ventana funcione correctamente, pero su contenido no debe modificarse directamente. Se encarga de armar la vista de la tabla de tokens.*

```
57      private void initComponents() {
58
59          jScrollPane1 = new javax.swing.JScrollPane();
60          tokensTable = new javax.swing.JTable();
61
62          setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
63          setTitle(title:"Tabla de Tokens");
64          setResizable(resizable:false);
65
66          tokensTable.setModel(new javax.swing.table.DefaultTableModel(
67              new Object [][] {
68
69              },
70              new String [] {
71                  "#", "Lexema", "Tipo", "Línea", "Columna"
72              }
73          ) {
74              Class[] types = new Class [] {
75                  java.lang.Object.class, java.lang.String.class, java.lang.String.class, java
76              };
77              boolean[] canEdit = new boolean [] {
78                  false, false, false, false, false
79              };
80
81              public Class getColumnClass(int columnIndex) {
82                  return types [columnIndex];
83              }
84
85              public boolean isCellEditable(int rowIndex, int columnIndex) {
86                  return canEdit [columnIndex];
87              }
88          });
```

## *3. TokensTable()*

***Descripción:***

*Inicializa la ventana, configura su ubicación en pantalla al centro y carga los tokens en la tabla llamando al método printTable().*

***Importancia:***

*Es el punto de entrada de la vista gráfica que presenta la tabla de tokens al usuario tras el análisis léxico.*

```
public TokensTable() {
    initComponents();
    this.setLocationRelativeTo(c:null);
    printTable();
}
```

## *4. printTable()*

***Descripción:***

*Recorre la lista tokensList y agrega cada token como una nueva fila a la tabla de la interfaz gráfica.*

*Pasos que realiza:*

1. *Obtiene el modelo de la tabla.*
2. *Recorre cada token en tokensList.*
3. *Agrega una fila con número, lexema, tipo, línea y columna.*
4. *Centra el contenido con centerTable().*

***Importancia:***

*Este método muestra visualmente todos los tokens reconocidos del archivo fuente, incluyendo su posición y tipo.*

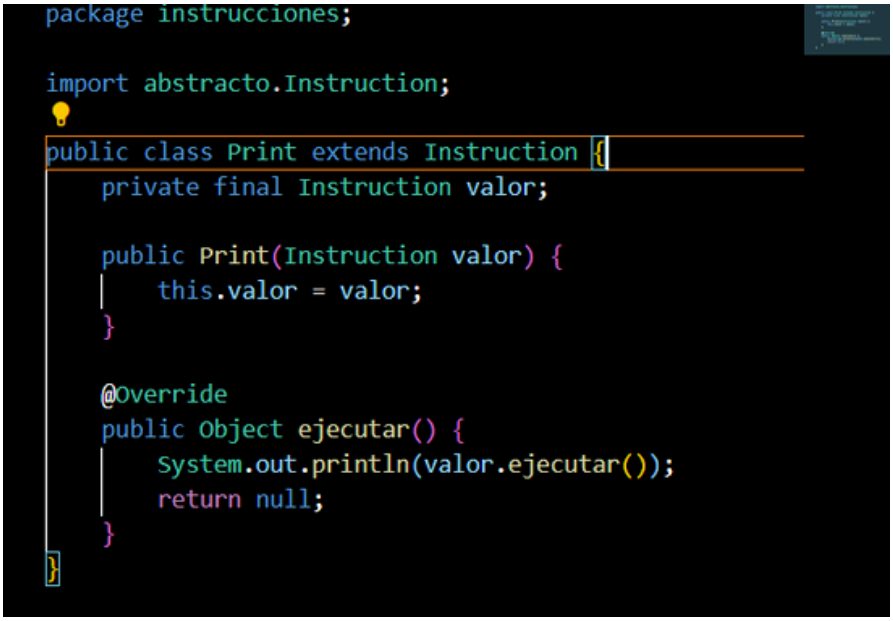
```
19
20      private void printTable() {
21          DefaultTableModel tableModel = (DefaultTableModel) tokensTable.getModel();
22
23          int number = 1;
24          Object[] row;
25
26          for (int i = 0; i < tokensList.size(); i++) {
27              row = new Object[5];
28              row[0] = number;
29              row[1] = tokensList.get(i).getLexeme();
30              row[2] = tokensList.get(i).getType();
31              row[3] = tokensList.get(i).getLine();
32              row[4] = tokensList.get(i).getColumn();
33
34              tableModel.addRow(row);
35              number++;
36          }
37
38          centerTable();
39      }
```

## *5. Método ejecutar() en la clase Print*

***Descripción:***

*Este método ejecuta una asignación de valor a una variable identificada por un nombre. Almacena el valor en una tabla de símbolos.*

```
function ejecutar()
    si condicion es verdadera entonces
        si printInstruccion ≠ null entonces
            printInstruccion.ejecutar()
    retornar null
```



```java
package instrucciones;

import abstracto.Instruction;

public class Print extends Instruction {
    private final Instruction valor;

    public Print(Instruction valor) {
        this.valor = valor;
    }

    @Override
    public Object ejecutar() {
        System.out.println(valor.ejecutar());
        return null;
    }
}
```

## *clase Errors*

***Descripción:***

*La clase Errors encapsula la información relacionada con errores detectados durante las fases de análisis léxico o sintáctico del compilador. Permite almacenar y acceder a datos clave sobre el tipo de error, su descripción, y su localización (línea y columna).*

## *Manejo de errores:*

*El programa maneja errores comunes de la siguiente manera:*

- *Si el usuario intenta buscar un texto vacío, se muestra un mensaje de advertencia para que ingrese una cadena.*
- *Si se intenta abrir un archivo de texto dañado o incompatible, se muestra un mensaje de error.*
- *Los errores en la ejecución del análisis léxico también son capturados y se notifican al usuario mediante la consola.*

## *Flujo de la aplicación*

- *El usuario ingresa un texto en el Editor de texto.*
- *Al hacer clic en el botón de Buscar, el texto se pasa al método BuscarYResaltar, que buscará todas las coincidencias de la cadena y las resaltará.*
- *Al hacer clic en el botón de Ejecutar, el programa procesa el texto con el algoritmo de análisis léxico, generando una lista de tokens que se muestra en una ventana secundaria.*
- *Si no hay errores en el análisis léxico se genera el análisis sintáctico.*

## *Instalación y Configuración*

1. *Asegúrate de tener Java y  Visual Studio Code configurado correctamente.*
2. *Ejecuta JFlex para generar Lexer.java:*
3. *java -jar jflex-full-1.9.1.jar lexer.flex*
4. *Asegúrate de que parser.java y sym.java estén en analyzers/*
5. *Ejecuta Home.java desde Visual Studio Code (construye proyecto si es necesario)*
6. *Abre o escribe código y presiona "Ejecutar"*

## *Mantenimiento y soporte*

*Para garantizar el correcto funcionamiento del sistema, se recomienda:*

- *Verificar rutas de JARs (JFlex y CUP)*
- *Asegurar que lexer.flex esté actualizado*
- *Limpiar listas (tokensList, errorsList) antes de ejecutar*

### *Información del Desarrollador*

*Desarrollador Principal:*
*Dayana Sofía Orozco Mendóza*
*Fecha de Creación: Mayo 2025*
*Tecnologías Utilizadas:*

- *Lenguaje de programación: Java*
  - *Tecnologías: Java 17, JFlex, CUP (manual), Swing (JFrame),*
- *Visual Studio Code*