Nombre de la práctica	ANALIZADOR LEXICO (UNIDAD 4)			No.	4
Asignatura:	LENGUAJES Y AUTÓMATAS I	Carrera:	INGENIERÍA EN SISTEMAS COMPUTACIONALES-3501	Duración de la práctica (Hrs)	10 horas

NOMBRE DEL ALUMNO: Jesús Navarrete Martínez
GRUPO: 3501

I. Competencia(s) específica(s):

Construye un analizador léxico a partir de un lenguaje de programación.

Encuadre con CACEI: Registra el (los) atributo(s) de egreso y los criterios de desempeño que se evaluarán en la materia.

No. atributo	Atributos de egreso del PE que impactan en la asignatura	No. Criterio	Criterios de desempeño	No. Indicador	Indicadores
2	El estudiante diseñará esquemas de trabajo y procesos, usando metodologías congruentes en la resolución de problemas de Ingeniería en Sistemas Computacionales	CD1	Identifica metodologías y procesos empleados en la resolución de problemas	I1	Identificación y reconocimiento de distintas metodologías para la resolución de problemas
		CD2	Diseña soluciones a problemas, empleando metodologías apropiadas al área	I1	Uso de metodologías para el modelado de la solución de sistemas y aplicaciones
				I2	Diseño algorítmico (Representación de diagramas de transiciones)
3	El estudiante plantea soluciones basadas en tecnologías empleando su juicio ingenieril para valorar necesidades, recursos y resultados esperados.	CD1	Emplea los conocimientos adquiridos para el desarrollar soluciones	I1	Elección de metodologías, técnicas y/o herramientas para el desarrollo de soluciones
				I2	Uso de metodologías adecuadas para el desarrollo de proyectos
				I3	Generación de productos y/o proyectos
		CD2	Analiza y comprueba resultados	I1	Realizar pruebas a los productos obtenidos
				I2	Documentar información de las pruebas realizadas y los resultados

II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):

Laboratorio de cómputo y equipo de cómputo personal.

III. Material empleado:

- Equipo de cómputo
- Software para desarrollo NetBeans

IV. Desarrollo de la práctica:

ANALIZADOR LÉXICO

DESCRIPCIÓN DEL PROBLEMA: Construcción de un Analizador Léxico

Un analizador léxico, también conocido como lexer o escáner, es una de las primeras y más fundamentales etapas en el proceso de compilación de un lenguaje de programación. Este componente tiene como principal objetivo transformar el código fuente, que generalmente consiste en una secuencia de caracteres, en una secuencia estructurada de tokens. Los tokens representan las unidades básicas y esenciales del lenguaje, tales como palabras reservadas, identificadores, operadores matemáticos y lógicos, delimitadores como paréntesis o llaves, y literales como números o cadenas de texto.
El proceso de análisis léxico no solo consiste en identificar y categorizar estas unidades, sino también en ignorar elementos que no contribuyen directamente a la semántica del lenguaje, como espacios en blanco y comentarios. Esto permite que las etapas posteriores del compilador trabajen con una representación más abstracta y manejable del código fuente.

La construcción de un analizador léxico enfrenta varios desafíos clave. Uno de ellos es la definición clara y precisa de las reglas léxicas que describen los tokens válidos del lenguaje. Estas reglas suelen expresarse mediante expresiones regulares, una herramienta poderosa para capturar patrones en cadenas de texto. Además, es crucial implementar un mecanismo eficiente para procesar el código fuente, ya que un lexer mal diseñado puede convertirse en un cuello de botella en el proceso de compilación.
Para simplificar y automatizar la implementación de analizadores léxicos, se utilizan herramientas como JFlex. JFlex es una librería que permite a los desarrolladores especificar las reglas léxicas de su lenguaje de programación utilizando una sintaxis basada en expresiones regulares. A partir de estas reglas, JFlex genera código Java que implementa el analizador léxico. Esto no solo reduce el tiempo y el esfuerzo de desarrollo, sino que también garantiza que el lexer sea eficiente y robusto.

En el contexto de este proyecto, se busca diseñar un analizador léxico para un lenguaje de programación definido específicamente, con una sintaxis y semántica particulares. Este analizador tendrá la responsabilidad de leer el código fuente proporcionado por el usuario y producir una secuencia de tokens que puedan ser utilizados por las etapas posteriores del compilador, como el analizador sintáctico y el analizador semántico. Estas etapas adicionales utilizarán los tokens para verificar la estructura y el significado del programa, asegurando que sea válido según las reglas del lenguaje y que cumpla con las expectativas del desarrollador, la implementación de un analizador léxico es un paso crucial en el diseño de un compilador. Su correcta ejecución no solo facilita el análisis posterior del código, sino que también permite detectar errores tempranos en el proceso de compilación, mejorando la calidad del software final y la experiencia del usuario.

EXPLICACIÓN DEL CONTENIDO DE LA TABLA DE TOKENS

¿Qué es?

La tabla de tokens es una estructura fundamental en el desarrollo de un analizador léxico. Consiste en un listado de elementos léxicos (tokens), que representan los componentes básicos de un lenguaje de programación, junto con sus respectivos lexemas, que son las representaciones textuales de estos elementos en el código fuente.

¿Para qué sirve?

La tabla de tokens tiene múltiples propósitos dentro del proyecto de un compilador o intérprete, entre ellos:

- **Definir los componentes léxicos del lenguaje:** Agrupa operadores, palabras reservadas, símbolos, tipos de datos, y otros elementos que el analizador léxico debe reconocer.
- **Facilitar el reconocimiento del código fuente:** Permite transformar las cadenas de caracteres del programa en una secuencia de tokens, simplificando su procesamiento.
- **Estandarizar el análisis:** Cada token está asociado a una categoría semántica, ayudando a las etapas posteriores del compilador (sintáctica y semántica) a trabajar con datos consistentes y uniformes.

Importancia dentro del proyecto:

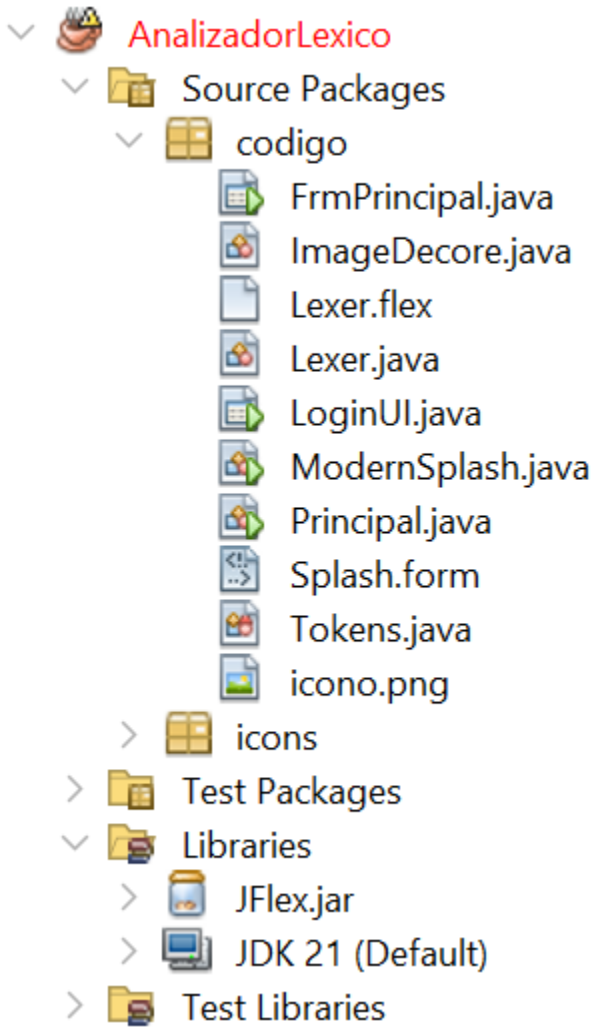
En el contexto del desarrollo de un analizador léxico, la tabla de tokens es esencial por las siguientes razones:

1. **Estructuración del lenguaje:** Define de manera precisa los elementos permitidos en el lenguaje, estableciendo sus reglas y delimitaciones.
2. **Optimización del análisis:** Al categorizar cada elemento del código fuente, se reduce la complejidad del análisis sintáctico y semántico, permitiendo que el compilador trabaje de manera más eficiente.
3. **Prevención de errores:** Facilita la detección de errores léxicos, como el uso de elementos no válidos, y contribuye a generar mensajes de error más claros para el usuario.

TABLA DE TOKENS			
	# TOKEN	TOKEN	LEXEMA
Operadores Binarios	1	operadorSuma	'+'
	2	operadorResta	'-'
	3	operadorMultiplicacion	'*'
	4	operadorDivision	'/'
	5	operadorModulo	'%'
	6	operadorPotencia	'^'
	7	operadorRaiz	'Rcuad'
	8	asignacion	'=>'
Operador Unario	9	incremento	'++'
	10	decremento	'--'
	11	operadorPositivo	'Pos'
	12	operadorNegativo	'Neg'
Operadores Comparacion	13	comparacionIgualdad	'=='
	14	mayorQue	'>'
	15	menorQue	'<'
	16	mayorIgual	'>=<'
	17	menorIgual	'<=>'
Operador Logico	18	operadorY	'&'
	19	operadorO	'|'
	20	operadorNo	'!'
Alfabeto	21	letrasMin	[a-z]'
	22	letrasMay	'[A-Z]'
	23	numero	[0-9]'
Comentarios	24	inicioComentarioMult	'(:'
	25	finalComentarioMult	':)'
	26	comentarioLinea	'}=)'
Tipos de Datos	27	valorEntero	'Ent'
	28	valorFlotante	'Flot'
	29	valorBooleano	'Boo'
	30	cadena	'Cad'
	31	carácter	'Cars'
Simbolos Especiales	32	finLinea	';-;'
	33	saltoLinea	'\sl'
	34	inicioTexto	'<<'
	35	finalTexto	'>>'
	36	parentIzq	'('
	37	parentDer	')'
	38	llaveIzq	'{'
	39	llaveDer	'}'
	40	corchetIzq	'['
	41	corchetDer	']'
	42	concatenar	':-:'
	43	arreglo	'[]-[]'
	44	puntoAcceso	'.'
	45	puntoDecmial	'.'
	46	dosPuntos	':'
	47	espacio	' '
	48	tabulacion	' '
	49	comillaDoble	'"'
	50	comillaSimple	'"
	51	coma	','
	52	guionBajo	' _'
	53	guionMedio	'-'
	54	diagonalInvertido	'\'
	55	diagonal	'/'
	56	signoAdmiracionAbre	'!'
	57	signoAdmiracionCierra	'ı'
	58	gato	'#'
	59	pesos	'$'

Palabras rervadas	60	condicionalIf	'si'
	61	condicionalElse	'siNo'
	62	cicloFor	'ciclo'
	63	cicloWhile	'whilo'
	64	cicloDo	'hacer'
	65	funcionSwitch	'segun'
	66	funcionCase	'caso'
	67	funcionSalirCase	'salir'
	68	predeterminado	'predeterminado'
	69	detener	'Detener'
	70	constante	'Constante'
	71	variable	'Variable'
	72	funcion	'funcion'
	73	clase	'Clase'
	74	imprimir	'Imprimir'
	75	importar	'Importar'
	76	funcionLeer	'Leer
	77	funcionIntroducirDatos	'IntroducirD'
Constantes	78	pi	'PI'
	79	euler	'E'

Estructura Del Proyecto



DESCRIPCIÓN DE CADA UNO DE LOS ARCHIVOS GENERADOS EN EL PROYECTO

Archivo Lexer.flex

El archivo Lexer.flex es un componente clave en el desarrollo de un compilador o intérprete. Este archivo define el analizador léxico, una herramienta que descompone el código fuente en unidades fundamentales llamadas *tokens*. Un *token* es una representación simbólica de elementos como operadores, palabras reservadas, identificadores, números, o símbolos especiales.

El propósito principal de este archivo es procesar y clasificar el código fuente de manera sistemática, eliminando elementos irrelevantes como espacios y comentarios, para facilitar el análisis sintáctico en las etapas posteriores del procesamiento del lenguaje. Además, al manejar errores mediante el token ERROR, el analizador contribuye a detectar entradas no válidas en el código, mejorando la eficiencia del sistema.
Este archivo actúa como el primer filtro del procesamiento del lenguaje, estableciendo una base sólida para que el resto del sistema interprete o compile el código fuente de manera eficiente y precisa. Es esencial para automatizar el reconocimiento de los elementos del lenguaje y garantizar que el código fuente sea correctamente comprendido, minimizando errores y facilitando el desarrollo general del proyecto.

Código Realizado:

```
package codigo;
import static codigo.Tokens.*;
%%
%class Lexer
%type Tokens
L=[a-zA-Z_]+
D=[0-9]+
espacio=[ ,\t,\r]+
%{
    public String lexeme;
%}
%%

/* Espacios en blanco */
{espacio} {/*Ignore*/}

/* Comentarios */
"//".* {/*Ignore*/}

/* Salto de línea */
"\n" {return Linea;}

/* Comillas */
"\"" {lexeme=yytext(); return comillaDoble;}
"'" {lexeme=yytext(); return comillaSimple;}

/* Tipos de datos */
"Ent" {lexeme=yytext(); return valorEntero;}
"Flot" {lexeme=yytext(); return valorFlotante;}
"Boo" {lexeme=yytext(); return valorBooleano;}
"Cad" {lexeme=yytext(); return cadena;}
"Cars" {lexeme=yytext(); return carácter;}

```

```
   /* Palabras reservadas */
   "si" {lexeme=yytext(); return condicionalIf;}
   "siNo" {lexeme=yytext(); return condicionalElse;}
   "ciclo" {lexeme=yytext(); return cicloFor;}
   "mientras" {lexeme=yytext(); return cicloWhile;}
   "hacer" {lexeme=yytext(); return cicloDo;}
   "segun" {lexeme=yytext(); return funcionSwitch;}
   "caso" {lexeme=yytext(); return funcionCase;}
   "salir" {lexeme=yytext(); return funcionSalirCase;}
   "predeterminado" {lexeme=yytext(); return predeterminado;}
   "Detener" {lexeme=yytext(); return detener;}
   "Constante" {lexeme=yytext(); return constante;}
   "Variable" {lexeme=yytext(); return variable;}
   "funcion" {lexeme=yytext(); return funcion;}
   "Clase" {lexeme=yytext(); return clase;}
   "Imprimir" {lexeme=yytext(); return imprimir;}
   "Leer" {lexeme=yytext(); return funcionLeer;}
   "IntroducirD" {lexeme=yytext(); return funcionIntroducirDatos;}

   /* Operadores matemáticos */
   "+" {lexeme=yytext(); return operadorSuma;}
   "-" {lexeme=yytext(); return operadorResta;}
   "*" {lexeme=yytext(); return operadorMultiplicacion;}
   "/" {lexeme=yytext(); return operadorDivision;}
   "%" {lexeme=yytext(); return operadorModulo;}
   "^" {lexeme=yytext(); return operadorPotencia;}
   "Rcuad" {lexeme=yytext(); return operadorRaiz;}

   /* Operadores de asignación y comparación */
   "=>" {lexeme=yytext(); return asignacion;}
   "==" {lexeme=yytext(); return comparacionIgualdad;}
   ">" {lexeme=yytext(); return mayorQue;}
   "<" {lexeme=yytext(); return menorQue;}
   "<" {lexeme=yytext(); return menorQue;}
   ">=<" {lexeme=yytext(); return mayorIgual;}
   "<=>" {lexeme=yytext(); return menorIgual;}

   /* Operadores lógicos */
   "&" {lexeme=yytext(); return operadorY;}
   "|" {lexeme=yytext(); return operadorO;}
   "!" {lexeme=yytext(); return operadorNo;}

   /* Operadores incrementales */
   "++" {lexeme=yytext(); return incremento;}
   "--" {lexeme=yytext(); return decremento;}
   /* Caracteres especiales */
   "(" {lexeme=yytext(); return parentIzq;}
   ")" {lexeme=yytext(); return parentDer;}
   "{" {lexeme=yytext(); return llaveIzq;}
   "}" {lexeme=yytext(); return llaveDer;}
   "[" {lexeme=yytext(); return corchetIzq;}
   "]" {lexeme=yytext(); return corchetDer;}
   ";-;" {lexeme=yytext(); return finLinea;}
   "<<" {lexeme=yytext(); return inicioTexto;}
   ">>" {lexeme=yytext(); return finalTexto;}
   "." {lexeme=yytext(); return puntoDecimal;}
   ":" {lexeme=yytext(); return dosPuntos;}
   " " {lexeme=yytext(); return espacio;}
   /* Identificadores y números */
   {L}({L}|{D})* {lexeme=yytext(); return Identificador;}
   ("(-"{D}+")"|{D}+ {lexeme=yytext(); return Numero;}

   /* Errores */
   . {return ERROR;}
```

Archivo Lexer.java (Generado automaticamente)

El archivo Lexer.java es una clase generada automáticamente por la herramienta JFlex, diseñada para construir un analizador léxico o escáner. Este archivo se encarga de identificar los distintos componentes léxicos o tokens de un lenguaje definido en una especificación, en este caso el archivo Lexer.flex. Los tokens pueden representar palabras clave, identificadores, operadores, literales, y otros elementos sintácticos básicos del lenguaje.

El analizador léxico es una parte esencial de un compilador o intérprete, ya que actúa como el primer paso en el análisis del código fuente. Este archivo tiene la responsabilidad de leer el flujo de entrada de caracteres y convertirlo en una secuencia organizada de tokens que el analizador sintáctico puede entender. Su importancia radica en garantizar que el código fuente se procese de manera eficiente y que cualquier error léxico se detecte desde etapas tempranas del análisis. Este proceso es fundamental para garantizar la correcta interpretación y ejecución del código fuente.

Código Generado:

```
/* The following code was generated by JFlex 1.4.3 on 01/01/25, 12:03 */

package codigo;
import static codigo.Tokens.*;

/**
 * This class is a scanner generated by
 * <a href="http://www.jflex.de/">JFlex</a> 1.4.3
 * on 01/01/25, 12:03 from the specification file
 * <tt>C:/Users/jesus/Downloads/AnalizadorLexico/src/codigo/Lexer.flex</tt>
 */
class Lexer {

  /** This character denotes the end of file */
  public static final int YYEOF = -1;

  /** initial size of the lookahead buffer */
  private static final int ZZ_BUFFERSIZE = 16384;

  /** lexical states */
  public static final int YYINITIAL = 0;

  /**
   * ZZ_LEXSTATE[l] is the state in the DFA for the lexical state l
   * ZZ_LEXSTATE[l+1] is the state in the DFA for the lexical state l
   *                  at the beginning of a line
   * l is of the form l = 2*k, k a non negative integer
   */
  private static final int ZZ_LEXSTATE[] = {
     0, 0
  };

  /**
```

```
   * Translates characters to character classes
   */
  private static final String ZZ_CMAP_PACKED =
    "\11\0\1\3\1\16\2\0\1\3\22\0\1\47\1\42\1\65\1\71"+
    "\1\60\1\22\1\40\1\66\1\45\1\50\1\21\1\17\1\3\1\20"+
    "\1\64\1\15\12\2\1\46\1\57\1\37\1\31\1\32\2\0\1\44"+
    "\1\54\1\55\1\74\1\52\1\53\2\44\1\77\2\44\1\100\1\44"+
    "\1\35\1\44\1\33\1\44\1\24\3\44\1\75\4\44\1\62\1\67"+
    "\1\63\1\23\1\1\1\0\1\27\1\76\1\25\1\30\1\10\1\7"+
    "\1\36\1\14\1\4\2\43\1\11\1\72\1\5\1\34\1\73\1\43"+
    "\1\56\1\12\1\6\1\26\1\43\1\13\3\43\1\61\1\41\1\51"+
    "\43\0\1\70\uff5e\0";

  /**
   * Translates characters to character classes
   */
  private static final char [] ZZ_CMAP = zzUnpackCMap(ZZ_CMAP_PACKED);

  /**
   * Translates DFA states to action switch labels.
   */
  private static final int [] ZZ_ACTION = zzUnpackAction();

  private static final String ZZ_ACTION_PACKED_0 =
    "\1\0\1\1\1\2\1\3\1\4\7\5\1\6\1\7"+
    "\1\10\1\11\1\12\1\13\1\14\1\5\1\1\1\15"+
    "\2\14\1\16\1\17\1\20\1\21\1\14\1\22\1\23"+
    "\1\24\1\25\4\14\1\1\1\26\1\27\1\30\1\31"+
    "\1\32\1\33\1\34\1\35\1\36\1\37\2\5\4\14"+
    "\1\40\1\1\41\1\40\1\42\2\40\1\43\4\40\1\4"+
    "\1\44\1\45\3\40\1\46\1\47\1\0\1\50\1\40"+
    "\1\51\1\40\1\0\1\52\3\0\1\53\1\0\6\40"+
    "\1\0\1\54\1\0\21\40\1\55\1\56\1\57\1\60"+
    "\1\0\1\61\1\62\1\63\1\64\1\40\1\65\1\40"+
    "\1\66\2\40\1\67\1\0\10\40\1\70\5\40\1\71"+
    "\1\41\1\72\1\40\1\73\1\40\1\0\7\40\1\74"+
    "\1\40\1\75\1\76\1\77\1\100\1\101\1\102\1\40"+
    "\1\103\20\40\1\104\3\40\1\105\5\40\1\106\1\40"+
    "\1\107\1\40\1\110\1\111\1\112\5\40\1\113\2\40"+
    "\1\114";

  private static int [] zzUnpackAction() {
    int [] result = new int[208];
    int offset = 0;
    offset = zzUnpackAction(ZZ_ACTION_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackAction(String packed, int offset, int [] result) {
    int i = 0;       /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int count = packed.charAt(i++);
      int value = packed.charAt(i++);
      do result[j++] = value; while (--count > 0);
    }
    return j;
  }


  /**
   * Translates a state to a row index in the transition table
   */
  private static final int [] ZZ_ROWMAP = zzUnpackRowMap();

```

```
  private static final String ZZ_ROWMAP_PACKED_0 =
    "\0\0\0\101\0\202\0\303\0\u0104\0\u0145\0\202\0\u0186"+
    "\0\u01c7\0\u0208\0\u0249\0\u028a\0\u02cb\0\u030c\0\u034d\0\101"+
    "\0\101\0\101\0\u038e\0\u03cf\0\u0410\0\u0451\0\u0492\0\u04d3"+
    "\0\u0514\0\101\0\101\0\101\0\202\0\u0555\0\u0596\0\101"+
    "\0\u05d7\0\u0618\0\u0659\0\u069a\0\u06db\0\u071c\0\u075d\0\101"+
    "\0\u079e\0\101\0\101\0\101\0\101\0\101\0\101\0\101"+
    "\0\u07df\0\u0820\0\u0861\0\u08a2\0\u08e3\0\u0924\0\202\0\303"+
    "\0\u0965\0\202\0\u09a6\0\u09e7\0\u0a28\0\u0a69\0\u0aaa\0\u0aeb"+
    "\0\u0b2c\0\u0b6d\0\101\0\101\0\u0bae\0\u0bef\0\u0c30\0\101"+
    "\0\101\0\u0c71\0\101\0\u0cb2\0\202\0\u0cf3\0\u0d34\0\101"+
    "\0\u0d75\0\u0db6\0\u0df7\0\101\0\u0e38\0\u0e79\0\u0eba\0\u0efb"+
    "\0\u0f3c\0\u0f7d\0\u0fbe\0\u0fff\0\101\0\u1040\0\u1081\0\u10c2"+
    "\0\u1103\0\u1144\0\u1185\0\u11c6\0\u1207\0\u1248\0\u1289\0\u12ca"+
    "\0\u130b\0\u134c\0\u138d\0\u13ce\0\u140f\0\u1450\0\u1491\0\101"+
    "\0\202\0\202\0\101\0\u14d2\0\101\0\101\0\101\0\202"+
    "\0\u1513\0\202\0\u1554\0\202\0\u1595\0\u15d6\0\101\0\u1617"+
    "\0\u1658\0\u1699\0\u16da\0\u171b\0\u175c\0\u179d\0\u17de\0\u181f"+
    "\0\202\0\u1860\0\u18a1\0\u18e2\0\u1923\0\u1964\0\202\0\101"+
    "\0\202\0\u19a5\0\202\0\u19e6\0\u1a27\0\u1a68\0\u1aa9\0\u1aea"+
    "\0\u1b2b\0\u1b6c\0\u1bad\0\u1bee\0\202\0\u1c2f\0\202\0\202"+
    "\0\202\0\202\0\202\0\202\0\u1c70\0\101\0\u1cb1\0\u1cf2"+
    "\0\u1d33\0\u1d74\0\u1db5\0\u1df6\0\u1e37\0\u1e78\0\u1eb9\0\u1efa"+
    "\0\u1f3b\0\u1f7c\0\u1fbd\0\u1ffe\0\u203f\0\u2080\0\202\0\u20c1"+
    "\0\u2102\0\u2143\0\202\0\u2184\0\u21c5\0\u2206\0\u2247\0\u2288"+
    "\0\202\0\u22c9\0\202\0\u230a\0\202\0\202\0\202\0\u234b"+
    "\0\u238c\0\u23cd\0\u240e\0\u244f\0\202\0\u2490\0\u24d1\0\202";

  private static int [] zzUnpackRowMap() {
    int [] result = new int[208];
    int offset = 0;
    offset = zzUnpackRowMap(ZZ_ROWMAP_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackRowMap(String packed, int offset, int [] result) {
    int i = 0;  /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int high = packed.charAt(i++) << 16;
      result[j++] = high | packed.charAt(i++);
    }
    return j;
  }

  /**
   * The transition table of the DFA
   */
  private static final int [] ZZ_TRANS = zzUnpackTrans();

  private static final String ZZ_TRANS_PACKED_0 =
    "\1\2\1\3\1\4\1\5\1\6\2\7\1\10\1\11"+
    "\1\7\1\12\1\13\1\14\1\15\1\5\1\16\1\17"+
    "\1\20\1\21\1\22\1\23\1\24\3\7\1\25\1\26"+
    "\1\27\1\7\1\30\1\7\1\31\1\32\1\33\1\34"+
    "\1\7\1\35\1\36\1\37\1\5\1\40\1\41\1\42"+
    "\1\43\1\44\1\45\1\7\1\46\1\47\1\50\1\51"+
    "\1\52\1\53\1\54\1\55\1\56\1\57\1\60\1\61"+
    "\1\62\1\63\1\64\1\7\1\65\1\66\102\0\2\67"+
    "\1\0\11\67\7\0\5\67\2\0\4\67\4\0\2\67"+
    "\5\0\5\67\13\0\7\67\2\0\1\70\101\0\1\5"+
    "\12\0\1\5\30\0\1\5\32\0\2\67\1\0\1\67"+
    "\1\71\1\67\1\72\5\67\7\0\5\67\2\0\4\67"+
    "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
    "\1\0\11\67\7\0\2\67\1\73\2\67\2\0\4\67"+
```

```
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\5\67\1\74\3\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\1\75\3\67\1\76\4\67\7\0\3\67\1\77"+
        "\1\67\2\0\4\67\4\0\2\67\5\0\5\67\13\0"+
        "\7\67\1\0\2\67\1\0\10\67\1\100\7\0\5\67"+
        "\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
        "\1\0\2\67\1\0\11\67\7\0\3\67\1\101\1\67"+
        "\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
        "\15\0\1\102\102\0\1\103\101\0\1\104\61\0\2\67"+
        "\1\0\11\67\7\0\1\67\1\105\3\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\1\106\10\67\7\0\3\67\1\107\1\67\2\0"+
        "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\31\0"+
        "\1\110\1\111\77\0\1\112\1\113\47\0\2\67\1\0"+
        "\11\67\7\0\5\67\2\0\1\67\1\114\2\67\4\0"+
        "\2\67\5\0\5\67\13\0\5\67\1\115\1\67\1\0"+
        "\2\67\1\0\4\67\1\116\4\67\7\0\5\67\2\0"+
        "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\31\0"+
        "\1\117\5\0\1\120\61\0\1\121\25\0\1\122\52\0"+
        "\1\123\27\0\1\124\61\0\1\125\50\0\2\67\1\0"+
        "\1\67\1\126\7\67\7\0\5\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\5\67\1\127\3\67\7\0\5\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\11\67\7\0\5\67\2\0\1\67\1\130\2\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\5\67\1\131\3\67\7\0\3\67\1\132\1\67\2\0"+
        "\1\67\1\133\2\67\4\0\2\67\5\0\5\67\13\0"+
        "\7\67\20\0\1\134\71\0\1\135\152\0\1\136\16\0"+
        "\2\67\1\0\1\137\10\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\11\67\7\0\5\67\2\0\4\67\4\0\2\67"+
```

```
        "\5\0\4\67\1\140\13\0\7\67\1\0\2\67\1\0"+
        "\4\67\1\141\4\67\7\0\5\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\11\67\7\0\3\67\1\142\1\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\1\67\1\143\7\67\7\0\5\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\1\144\6\67\1\0\2\67"+
        "\1\0\4\67\1\145\4\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\2\67\1\72\6\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\1\67\1\146\7\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\6\67\1\147\2\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\11\67\7\0\5\67\2\0\2\67\1\150\1\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\11\67\7\0\5\67\2\0\3\67\1\151\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\5\67\1\152\3\67\7\0\5\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\1\153\10\67\7\0\5\67\2\0\4\67\4\0\2\67"+
        "\5\0\5\67\13\0\7\67\1\0\2\67\1\0\11\67"+
        "\7\0\1\67\1\154\3\67\2\0\4\67\4\0\2\67"+
        "\5\0\5\67\13\0\7\67\16\102\1\0\62\102\1\0"+
        "\2\67\1\0\11\67\7\0\2\67\1\155\2\67\2\0"+
        "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\1\0"+
        "\2\67\1\0\11\67\7\0\1\67\1\156\3\67\2\0"+
        "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\1\0"+
        "\2\67\1\0\6\67\1\157\2\67\7\0\5\67\2\0"+
        "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\37\0"+
        "\1\160\42\0\2\67\1\0\6\67\1\161\2\67\7\0"+
        "\5\67\2\0\4\67\4\0\2\67\5\0\5\67\13\0"+
```

```
"\7\67\11\0\2\67\11\0\11\67\7\0\5\67\2\0"+
"\13\67\11\162\4\0\2\67\5\0\5\67\13\0\7\67"+
"\32\0\1\163\50\0\1\164\145\0\1\165\77\0\1\166"+
"\102\0\1\167\31\0\2\67\11\0\2\67\11\170\6\67"+
"\7\0\5\67\2\0\4\67\4\0\2\67\5\0\5\67"+
"\13\0\7\67\11\0\2\67\11\0\11\67\7\0\5\67"+
"\2\0\1\67\11\171\2\67\4\0\2\67\5\0\5\67"+
"\13\0\7\67\11\0\2\67\11\0\11\67\7\0\5\67"+
"\2\0\1\67\11\172\2\67\4\0\2\67\5\0\5\67"+
"\13\0\7\67\11\0\2\67\11\0\11\67\7\0\3\67"+
"\11\173\11\67\2\0\4\67\4\0\2\67\5\0\5\67"+
"\13\0\7\67\11\0\2\67\11\0\11\67\7\0\4\67"+
"\11\174\2\0\4\67\4\0\2\67\5\0\4\67\11\175"+
"\13\0\7\67\11\0\2\67\11\0\1\67\11\176\7\67"+
"\7\0\5\67\2\0\4\67\4\0\2\67\5\0\5\67"+
"\13\0\7\67\57\0\1\177\41\0\1\200\61\0\2\67"+
"\11\0\4\67\11\201\4\67\7\0\5\67\2\0\4\67"+
"\4\0\2\67\5\0\5\67\13\0\7\67\11\0\2\67"+
"\11\0\4\67\11\202\4\67\7\0\5\67\2\0\4\67"+
"\4\0\2\67\5\0\5\67\13\0\7\67\11\0\2\67"+
"\11\0\2\67\11\203\6\67\7\0\5\67\2\0\4\67"+
"\4\0\2\67\5\0\5\67\13\0\7\67\11\0\2\67"+
"\11\0\11\67\7\0\5\67\2\0\4\67\4\0\2\67"+
"\5\0\4\67\11\204\13\0\7\67\11\0\2\67\11\0"+
"\2\67\11\205\6\67\7\0\5\67\2\0\4\67\4\0"+
"\2\67\5\0\5\67\13\0\7\67\11\0\2\67\11\0"+
"\11\67\7\0\5\67\2\0\4\67\4\0\2\67\5\0"+
"\5\67\13\0\11\67\11\206\5\67\11\0\2\67\11\0"+
"\4\67\11\207\4\67\7\0\5\67\2\0\4\67\4\0"+
"\2\67\5\0\5\67\13\0\7\67\11\0\2\67\11\0"+
"\11\67\7\0\11\67\11\210\3\67\2\0\4\67\4\0"+
"\2\67\5\0\5\67\13\0\7\67\11\0\2\67\11\0"+
"\4\67\11\72\4\67\7\0\5\67\2\0\4\67\4\0"+
```

```
"\2\67\5\0\5\67\13\0\7\67\11\0\2\67\11\0"+
"\11\67\7\0\5\67\2\0\11\67\11\211\2\67\4\0"+
"\2\67\5\0\5\67\13\0\7\67\11\0\2\67\11\0"+
"\11\67\7\0\2\67\11\212\2\67\2\0\4\67\4\0"+
"\2\67\5\0\5\67\13\0\7\67\11\0\2\67\11\0"+
"\11\213\10\67\7\0\5\67\2\0\4\67\4\0\2\67"+
"\5\0\5\67\13\0\7\67\11\0\2\67\11\0\5\67"+
"\11\147\3\67\7\0\5\67\2\0\4\67\4\0\2\67"+
"\5\0\5\67\13\0\7\67\11\0\2\67\11\0\4\67"+
"\11\214\4\67\7\0\5\67\2\0\4\67\4\0\2\67"+
"\5\0\5\67\13\0\7\67\11\0\2\67\11\0\11\67"+
"\7\0\3\67\11\215\11\67\2\0\4\67\4\0\2\67"+
"\5\0\5\67\13\0\7\67\11\0\2\67\11\0\5\67"+
"\11\216\3\67\7\0\5\67\2\0\4\67\4\0\2\67"+
"\5\0\5\67\13\0\7\67\11\0\2\67\11\0\11\67"+
"\7\0\5\67\2\0\11\67\11\217\2\67\4\0\2\67"+
"\5\0\5\67\13\0\7\67\2\0\1\164\45\0\1\220"+
"\31\0\2\67\11\0\2\67\11\221\6\67\7\0\5\67"+
"\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
"\11\0\2\67\11\0\6\67\11\222\2\67\7\0\5\67"+
"\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
"\11\0\2\67\11\0\6\67\11\223\2\67\7\0\5\67"+
"\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
"\11\0\2\67\11\0\6\67\11\224\2\67\7\0\5\67"+
"\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
"\62\0\1\225\17\0\2\67\11\0\11\67\11\226\7\67"+
"\7\0\5\67\2\0\4\67\4\0\2\67\5\0\5\67"+
"\13\0\7\67\11\0\2\67\11\0\11\67\7\0\4\67"+
"\11\227\2\0\4\67\4\0\2\67\5\0\5\67\13\0"+
"\7\67\11\0\2\67\11\0\4\67\11\230\4\67\7\0"+
"\5\67\2\0\4\67\4\0\2\67\5\0\5\67\13\0"+
"\7\67\11\0\2\67\11\0\11\231\10\67\7\0\5\67"+
"\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
```

```
        "\1\0\2\67\1\0\11\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\4\67\1\232\13\0\7\67\1\0"+
        "\2\67\1\0\11\67\7\0\5\67\2\0\1\67\1\233"+
        "\2\67\4\0\2\67\5\0\4\67\1\234\13\0\7\67"+
        "\1\0\2\67\1\0\11\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\4\67\1\235\13\0\7\67\1\0"+
        "\2\67\1\0\1\236\10\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\1\67\1\237\7\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
        "\1\0\11\67\7\0\5\67\2\0\4\67\4\0\2\67"+
        "\5\0\4\67\1\240\13\0\7\67\1\0\2\67\1\0"+
        "\11\67\7\0\5\67\2\0\4\67\4\0\2\67\5\0"+
        "\4\67\1\241\13\0\7\67\1\0\2\67\1\0\11\67"+
        "\7\0\4\67\1\242\2\0\4\67\4\0\2\67\5\0"+
        "\5\67\13\0\7\67\1\0\2\67\1\0\11\67\7\0"+
        "\5\67\2\0\1\67\1\243\2\67\4\0\2\67\5\0"+
        "\5\67\13\0\7\67\1\0\2\67\1\0\4\67\1\244"+
        "\4\67\7\0\5\67\2\0\4\67\4\0\2\67\5\0"+
        "\5\67\13\0\7\67\1\0\2\67\1\0\2\67\1\245"+
        "\6\67\7\0\5\67\2\0\4\67\4\0\2\67\5\0"+
        "\5\67\13\0\7\67\63\0\1\246\16\0\2\67\1\0"+
        "\2\67\1\247\6\67\7\0\5\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\4\67\1\250\4\67\7\0\5\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\1\67\1\251\7\67\7\0\5\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\11\67\7\0\3\67\1\252\1\67\2\0\4\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\11\67\7\0\5\67\2\0\1\67\1\253\2\67\4\0"+
        "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
        "\11\67\7\0\5\67\2\0\4\67\4\0\2\67\5\0"+
```

```
        "\4\67\1\254\13\0\7\67\1\0\2\67\1\0\1\255"+
        "\10\67\7\0\5\67\2\0\4\67\4\0\2\67\5\0"+
        "\5\67\13\0\7\67\1\0\2\67\1\0\11\67\7\0"+
        "\5\67\2\0\1\67\1\256\2\67\4\0\2\67\5\0"+
        "\5\67\13\0\7\67\1\0\2\67\1\0\11\67\7\0"+
        "\3\67\1\257\1\67\2\0\4\67\4\0\2\67\5\0"+
        "\5\67\13\0\7\67\1\0\2\67\1\0\11\67\7\0"+
        "\5\67\2\0\4\67\4\0\2\67\5\0\4\67\1\260"+
        "\13\0\7\67\1\0\2\67\1\0\2\67\1\261\6\67"+
        "\7\0\5\67\2\0\4\67\4\0\2\67\5\0\5\67"+
        "\13\0\7\67\1\0\2\67\1\0\4\67\1\262\4\67"+
        "\7\0\5\67\2\0\4\67\4\0\2\67\5\0\5\67"+
        "\13\0\7\67\1\0\2\67\1\0\11\67\7\0\5\67"+
        "\2\0\4\67\4\0\2\67\5\0\5\67\13\0\4\67"+
        "\1\263\2\67\1\0\2\67\1\0\11\67\7\0\4\67"+
        "\1\264\2\0\4\67\4\0\2\67\5\0\5\67\13\0"+
        "\7\67\1\0\2\67\1\0\2\67\1\265\6\67\7\0"+
        "\5\67\2\0\4\67\4\0\2\67\5\0\5\67\13\0"+
        "\7\67\1\0\2\67\1\0\11\67\7\0\5\67\2\0"+
        "\4\67\4\0\2\67\5\0\5\67\13\0\1\266\6\67"+
        "\1\0\2\67\1\0\1\67\1\267\7\67\7\0\5\67"+
        "\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
        "\1\0\2\67\1\0\1\67\1\270\7\67\7\0\5\67"+
        "\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
        "\1\0\2\67\1\0\11\67\7\0\3\67\1\271\1\67"+
        "\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
        "\1\0\2\67\1\0\4\67\1\272\4\67\7\0\5\67"+
        "\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
        "\1\0\2\67\1\0\11\67\7\0\5\67\2\0\4\67"+
        "\4\0\2\67\5\0\4\67\1\273\13\0\7\67\1\0"+
        "\2\67\1\0\5\67\1\274\3\67\7\0\5\67\2\0"+
        "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\1\0"+
        "\2\67\1\0\11\67\7\0\2\67\1\275\2\67\2\0"+
```

```
    "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\1\0"+
    "\2\67\1\0\11\67\7\0\3\67\1\276\1\67\2\0"+
    "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\1\0"+
    "\2\67\1\0\1\277\10\67\7\0\5\67\2\0\4\67"+
    "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
    "\1\0\2\67\1\300\6\67\7\0\5\67\2\0\4\67"+
    "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
    "\1\0\6\67\1\301\2\67\7\0\5\67\2\0\4\67"+
    "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
    "\1\0\11\67\7\0\5\67\2\0\4\67\4\0\2\67"+
    "\5\0\4\67\1\302\13\0\7\67\1\0\2\67\1\0"+
    "\4\67\1\303\4\67\7\0\5\67\2\0\4\67\4\0"+
    "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
    "\11\67\7\0\1\67\1\304\3\67\2\0\4\67\4\0"+
    "\2\67\5\0\5\67\13\0\7\67\1\0\2\67\1\0"+
    "\11\67\7\0\5\67\2\0\4\67\4\0\2\67\5\0"+
    "\4\67\1\305\13\0\7\67\1\0\2\67\1\0\11\67"+
    "\7\0\5\67\2\0\4\67\4\0\2\67\5\0\4\67"+
    "\1\306\13\0\7\67\1\0\2\67\1\0\4\67\1\307"+
    "\4\67\7\0\5\67\2\0\4\67\4\0\2\67\5\0"+
    "\5\67\13\0\7\67\1\0\2\67\1\0\11\67\7\0"+
    "\5\67\2\0\4\67\4\0\2\67\5\0\5\67\13\0"+
    "\1\310\6\67\1\0\2\67\1\0\1\311\10\67\7\0"+
    "\5\67\2\0\4\67\4\0\2\67\5\0\5\67\13\0"+
    "\7\67\1\0\2\67\1\0\1\312\10\67\7\0\5\67"+
    "\2\0\4\67\4\0\2\67\5\0\5\67\13\0\7\67"+
    "\1\0\2\67\1\0\11\67\7\0\5\67\2\0\4\67"+
    "\4\0\2\67\5\0\4\67\1\313\13\0\7\67\1\0"+
    "\2\67\1\0\1\67\1\314\7\67\7\0\5\67\2\0"+
    "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\1\0"+
    "\2\67\1\0\11\67\7\0\5\67\2\0\4\67\4\0"+
    "\2\67\5\0\5\67\13\0\2\67\1\315\4\67\1\0"+
    "\2\67\1\0\11\67\7\0\3\67\1\316\1\67\2\0"+
    "\4\67\4\0\2\67\5\0\5\67\13\0\7\67\1\0"+
    "\4\67\1\0\11\67\7\0\4\67\1\317\2\0\4\67"+
    "\4\0\2\67\5\0\5\67\13\0\7\67\1\0\2\67"+
    "\1\0\11\67\7\0\5\67\2\0\1\67\1\320\2\67"+
    "\4\0\2\67\5\0\5\67\13\0\7\67";

  private static int [] zzUnpackTrans() {
    int [] result = new int[9490];
    int offset = 0;
    offset = zzUnpackTrans(ZZ_TRANS_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackTrans(String packed, int offset, int [] result) {
    int i = 0;       /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int count = packed.charAt(i++);
      int value = packed.charAt(i++);
      value--;
      do result[j++] = value; while (--count > 0);
    }
    return j;
  }


  /* error codes */
  private static final int ZZ_UNKNOWN_ERROR = 0;
  private static final int ZZ_NO_MATCH = 1;
  private static final int ZZ_PUSHBACK_2BIG = 2;

  /* error messages for the codes above */
```

```
  private static final String ZZ_ERROR_MSG[] = {
    "Unkown internal scanner error",
    "Error: could not match input",
    "Error: pushback value was too large"
  };

  /**
   * ZZ_ATTRIBUTE[aState] contains the attributes of state <code>aState</code>
   */
  private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();

  private static final String ZZ_ATTRIBUTE_PACKED_0 =
    "\1\0\1\11\15\1\3\11\7\1\3\11\3\1\1\11"+
    "\7\1\1\11\1\1\7\11\22\1\2\11\3\1\2\11"+
    "\1\0\1\11\3\1\1\0\1\11\3\0\1\11\1\0"+
    "\6\1\1\0\1\11\1\0\21\1\1\11\2\1\1\11"+
    "\1\0\3\11\7\1\1\11\1\0\17\1\1\11\4\1"+
    "\1\0\20\1\1\11\52\1";

  private static int [] zzUnpackAttribute() {
    int [] result = new int[208];
    int offset = 0;
    offset = zzUnpackAttribute(ZZ_ATTRIBUTE_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackAttribute(String packed, int offset, int [] result) {
    int i = 0;       /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int count = packed.charAt(i++);
      int value = packed.charAt(i++);
      do result[j++] = value; while (--count > 0);
    }
    return j;
  }

  /** the input device */
  private java.io.Reader zzReader;

  /** the current state of the DFA */
  private int zzState;

  /** the current lexical state */
  private int zzLexicalState = YYINITIAL;

  /** this buffer contains the current text to be matched and is
      the source of the yytext() string */
  private char zzBuffer[] = new char[ZZ_BUFFERSIZE];

  /** the textposition at the last accepting state */
  private int zzMarkedPos;

  /** the current text position in the buffer */
  private int zzCurrentPos;

  /** startRead marks the beginning of the yytext() string in the buffer */
  private int zzStartRead;

  /** endRead marks the last character in the buffer, that has been read
      from input */
  private int zzEndRead;

  /** number of newlines encountered up to the start of the matched text */
  private int yyline;
```

```
496
497     /** the number of characters up to the start of the matched text */
        private int yychar;
499
500     /**
501      * the number of characters from the last newline up to the start of the
502      * matched text
503      */
        private int yycolumn;
505
506     /**
507      * zzAtBOL == true <=> the scanner is currently at the beginning of a line
508      */
        private boolean zzAtBOL = true;
510
511     /** zzAtEOF == true <=> the scanner is at the EOF */
512     private boolean zzAtEOF;
513
514     /** denotes if the user-EOF-code has already been executed */
        private boolean zzEOFDone;
516
517     /* user code: */
518       public String lexeme;
519
520
521     /**
522      * Creates a new scanner
523      * There is also a java.io.InputStream version of this constructor.
524      *
525      * @param   in  the java.io.Reader to read input from.
526      */
527     Lexer(java.io.Reader in) {
528       this.zzReader = in;
529     }
530
531     /**
532      * Creates a new scanner.
533      * There is also java.io.Reader version of this constructor.
534      *
535      * @param   in  the java.io.Inputstream to read input from.
536      */
        Lexer(java.io.InputStream in) {
538       this(new java.io.InputStreamReader(in));
539     }
540
541     /**
542      * Unpacks the compressed character translation table.
543      *
544      * @param packed   the packed character translation table
545      * @return         the unpacked character translation table
546      */
547     private static char [] zzUnpackCMap(String packed) {
548       char [] map = new char[0x10000];
549       int i = 0;  /* index in packed string  */
550       int j = 0;  /* index in unpacked array */
551       while (i < 166) {
552         int  count = packed.charAt(i++);
553         char value = packed.charAt(i++);
554         do map[j++] = value; while (--count > 0);
555       }
556       return map;
557     }
558
559
560     /**
561      * Refills the input buffer.
```

```
     *
     * @return      <code>false</code>, iff there was new input.
     *
     * @exception   java.io.IOException  if any I/O-Error occurs
     */
    private boolean zzRefill() throws java.io.IOException {

      /* first: make room (if you can) */
      if (zzStartRead > 0) {
        System.arraycopy(zzBuffer, zzStartRead,
                         zzBuffer, 0,
                         zzEndRead-zzStartRead);

        /* translate stored positions */
        zzEndRead-= zzStartRead;
        zzCurrentPos-= zzStartRead;
        zzMarkedPos-= zzStartRead;
        zzStartRead = 0;
      }

      /* is the buffer big enough? */
      if (zzCurrentPos >= zzBuffer.length) {
        /* if not: blow it up */
        char newBuffer[] = new char[zzCurrentPos*2];
        System.arraycopy(zzBuffer, 0, newBuffer, 0, zzBuffer.length);
        zzBuffer = newBuffer;
      }

      /* finally: fill the buffer with new input */
      int numRead = zzReader.read(zzBuffer, zzEndRead,
                                  zzBuffer.length-zzEndRead);

      if (numRead > 0) {
        zzEndRead+= numRead;
        return false;
      }
      // unlikely but not impossible: read 0 characters, but not at end of stream
      if (numRead == 0) {
        int c = zzReader.read();
        if (c == -1) {
          return true;
        } else {
          zzBuffer[zzEndRead++] = (char) c;
          return false;
        }
      }

        // numRead < 0
      return true;
    }


    /**
     * Closes the input stream.
     */
    public final void yyclose() throws java.io.IOException {
      zzAtEOF = true;            /* indicate end of file */
      zzEndRead = zzStartRead;  /* invalidate buffer    */

      if (zzReader != null)
        zzReader.close();
    }


    /**
     * Resets the scanner to read from a new input stream.
```

```
628       * Does not close the old reader.
629       *
630       * All internal variables are reset, the old input stream
631       * <b>cannot</b> be reused (internal buffer is discarded and lost).
632       * Lexical state is set to <tt>ZZ_INITIAL</tt>.
633       *
634       * @param reader   the new input stream
635       */
636      public final void yyreset(java.io.Reader reader) {
637        zzReader = reader;
638        zzAtBOL  = true;
639        zzAtEOF  = false;
640        zzEOFDone = false;
641        zzEndRead = zzStartRead = 0;
642        zzCurrentPos = zzMarkedPos = 0;
643        yyline = yychar = yycolumn = 0;
644        zzLexicalState = YYINITIAL;
645      }
646
647
648      /**
649       * Returns the current lexical state.
650       */
651      public final int yystate() {
652        return zzLexicalState;
653      }
654
655
656      /**
657       * Enters a new lexical state
658       *
659       * @param newState the new lexical state
660       */
661      public final void yybegin(int newState) {
662        zzLexicalState = newState;
663      }
664
665
666      /**
667       * Returns the text matched by the current regular expression.
668       */
669      public final String yytext() {
670        return new String( zzBuffer, zzStartRead, zzMarkedPos-zzStartRead );
671      }
672
673
674      /**
675       * Returns the character at position <tt>pos</tt> from the
676       * matched text.
677       *
678       * It is equivalent to yytext().charAt(pos), but faster
679       *
680       * @param pos the position of the character to fetch.
681       *            A value from 0 to yylength()-1.
682       *
683       * @return the character at position pos
684       */
685      public final char yycharat(int pos) {
686        return zzBuffer[zzStartRead+pos];
687      }
688
689
690      /**
691       * Returns the length of the matched text region.
692       */
693      public final int yylength() {
```

```
    return zzMarkedPos-zzStartRead;
  }


  /**
   * Reports an error that occured while scanning.
   *
   * In a wellformed scanner (no or only correct usage of
   * yypushback(int) and a match-all fallback rule) this method
   * will only be called with things that "Can't Possibly Happen".
   * If this method is called, something is seriously wrong
   * (e.g. a JFlex bug producing a faulty scanner etc.).
   *
   * Usual syntax/scanner level error handling should be done
   * in error fallback rules.
   *
   * @param   errorCode  the code of the errormessage to display
   */
  private void zzScanError(int errorCode) {
    String message;
    try {
      message = ZZ_ERROR_MSG[errorCode];
    }
    catch (ArrayIndexOutOfBoundsException e) {
      message = ZZ_ERROR_MSG[ZZ_UNKNOWN_ERROR];
    }

    throw new Error(message);
  }


  /**
   * Pushes the specified amount of characters back into the input stream.
   *
   * They will be read again by then next call of the scanning method
   *
   * @param number  the number of characters to be read again.
   *                This number must not be greater than yylength()!
   */
  public void yypushback(int number)  {
    if ( number > yylength() )
      zzScanError(ZZ_PUSHBACK_2BIG);

    zzMarkedPos -= number;
  }


  /**
   * Resumes scanning until the next regular expression is matched,
   * the end of input is encountered or an I/O-Error occurs.
   *
   * @return      the next token
   * @exception   java.io.IOException  if any I/O-Error occurs
   */
  public Tokens yylex() throws java.io.IOException {
    int zzInput;
    int zzAction;

    // cached fields:
    int zzCurrentPosL;
    int zzMarkedPosL;
    int zzEndReadL = zzEndRead;
    char [] zzBufferL = zzBuffer;
    char [] zzCMapL = ZZ_CMAP;

    int [] zzTransL = ZZ_TRANS;
```

```
760       int [] zzRowMapL = ZZ_ROWMAP;
761       int [] zzAttrL = ZZ_ATTRIBUTE;
762
763       while (true) {
764         zzMarkedPosL = zzMarkedPos;
765
766         zzAction = -1;
767
768         zzCurrentPosL = zzCurrentPos = zzStartRead = zzMarkedPosL;
769
770         zzState = ZZ_LEXSTATE[zzLexicalState];
771
772
773         zzForAction: {
774           while (true) {
775
776             if (zzCurrentPosL < zzEndReadL)
777               zzInput = zzBufferL[zzCurrentPosL++];
778             else if (zzAtEOF) {
779               zzInput = YYEOF;
780               break zzForAction;
781             }
782             else {
783               // store back cached positions
784               zzCurrentPos  = zzCurrentPosL;
785               zzMarkedPos   = zzMarkedPosL;
786               boolean eof = zzRefill();
787               // get translated positions and possibly new buffer
788               zzCurrentPosL  = zzCurrentPos;
789               zzMarkedPosL   = zzMarkedPos;
790               zzBufferL      = zzBuffer;
791               zzEndReadL     = zzEndRead;
792               if (eof) {
793                 zzInput = YYEOF;
794                 break zzForAction;
795               }
796               else {
797                 zzInput = zzBufferL[zzCurrentPosL++];
798               }
799             }
800             int zzNext = zzTransL[ zzRowMapL[zzState] + zzCMapL[zzInput] ];
801             if (zzNext == -1) break zzForAction;
802             zzState = zzNext;
803
804             int zzAttributes = zzAttrL[zzState];
805             if ( (zzAttributes & 1) == 1 ) {
806               zzAction = zzState;
807               zzMarkedPosL = zzCurrentPosL;
808               if ( (zzAttributes & 8) == 8 ) break zzForAction;
809             }
810
811           }
812         }
813
814         // store back cached position
815         zzMarkedPos = zzMarkedPosL;
816
817         switch (zzAction < 0 ? zzAction : ZZ_ACTION[zzAction]) {
818           case 20:
819             { return parentDer;
820             }
821           case 77: break;
822           case 21:
823             { return llaveDer;
824             }
825           case 78: break;
```

```
826             case 10:
827               { return operadorModulo;
828               }
829             case 79: break;
830             case 32:
831               { lexeme=yytext(); return Identificador;
832               }
833             case 80: break;
834             case 22:
835               { return pesos;
836               }
837             case 81: break;
838             case 54:
839               { return cadena;
840               }
841             case 82: break;
842             case 69:
843               { return detener;
844               }
845             case 83: break;
846             case 31:
847               { return gato;
848               }
849             case 84: break;
850             case 3:
851               { return numero;
852               }
853             case 85: break;
854             case 2:
855               { return guionBajo;
856               }
857             case 86: break;
858             case 67:
859               { return arreglo;
860               }
861             case 87: break;
862             case 11:
863               { return operadorPotencia;
864               }
865             case 88: break;
866             case 72:
867               { return importar;
868               }
869             case 89: break;
870             case 34:
871               { lexeme=yytext(); return Reservadas;
872               }
873             case 90: break;
874             case 75:
875               { return funcionIntroducirDatos;
876               }
877             case 91: break;
878             case 36:
879               { return incremento;
880               }
881             case 92: break;
882             case 24:
883               { return corchetIzq;
884               }
885             case 93: break;
886             case 17:
887               { return operadorNo;
888               }
889             case 94: break;
890             case 4:
891               { /*Ignore*/
```

```
892             }
893             case 95: break;
894             case 8:
895               { return operadorResta;
896               }
897             case 96: break;
898             case 60:
899               { return funcionLeer;
900               }
901             case 97: break;
902             case 46:
903               { return operadorPositivo;
904               }
905             case 98: break;
906             case 14:
907               { return menorQue;
908               }
909             case 99: break;
910             case 5:
911               { return letrasMin;
912               }
913             case 100: break;
914             case 70:
915               { return cicloWhile;
916               }
917             case 101: break;
918             case 53:
919               { return valorBooleano;
920               }
921             case 102: break;
922             case 65:
923               { return cicloFor;
924               }
```

```
925             case 103: break;
926             case 42:
927               { return inicioTexto;
928               }
929             case 104: break;
930             case 66:
931               { return clase;
932               }
933             case 105: break;
934             case 1:
935               { return ERROR;
936               }
937             case 106: break;
938             case 19:
939               { return dosPuntos;
940               }
941             case 107: break;
942             case 28:
943               { return comillaSimple;
944               }
945             case 108: break;
946             case 76:
947               { return predeterminado;
948               }
949             case 109: break;
950             case 50:
951               { return concatenar;
952               }
953             case 110: break;
954             case 56:
955               { return condicionalElse;
956               }
957             case 111: break;
```

```
958             case 39:
959               { return asignacion;
960               }
961             case 112: break;
962             case 44:
963               { return saltoLinea;
964               }
965             case 113: break;
966             case 63:
967               { return cicloDo;
968               }
969             case 114: break;
970             case 49:
971               { return inicioComentarioMult;
972               }
973             case 115: break;
974             case 37:
975               { return decremento;
976               }
977             case 116: break;
978             case 13:
979               { return mayorQue;
980               }
981             case 117: break;
982             case 64:
983               { return operadorRaiz;
984               }
985             case 118: break;
986             case 35:
987               { return condicionalIf;
988               }
989             case 119: break;
990             case 57:
991               { return funcionCase;
992               }
993             case 120: break;
994             case 16:
995               { return operadorO;
996               }
997             case 121: break;
998             case 43:
999               { return finalComentarioMult;
1000              }
1001            case 122: break;
1002            case 74:
1003              { return constante;
1004              }
1005            case 123: break;
1006            case 25:
1007              { return corchetDer;
1008              }
1009            case 124: break;
1010            case 58:
1011              { return valorFlotante;
1012              }
1013            case 125: break;
1014            case 51:
1015              { return comentarioLinea;
1016              }
1017            case 126: break;
1018            case 47:
1019              { return operadorNegativo;
1020              }
1021            case 127: break;
1022            case 26:
1023              { return puntoAcceso;
```

```
1024            }
1025          case 128: break;
1026          case 18:
1027            { return parentIzq;
1028            }
1029          case 129: break;
1030          case 23:
1031            { return llaveIzq;
1032            }
1033          case 130: break;
1034          case 40:
1035            { return finalTexto;
1036            }
1037          case 131: break;
1038          case 33:
1039            { lexeme=yytext(); return Numero;
1040            }
1041          case 132: break;
1042          case 48:
1043            { return menorIgual;
1044            }
1045          case 133: break;
1046          case 27:
1047            { return comillaDoble;
1048            }
1049          case 134: break;
1050          case 7:
1051            { return operadorSuma;
1052            }
1053          case 135: break;
1054          case 12:
1055            { return letrasMay;
1056            }
```

```
1057          case 136: break;
1058          case 45:
1059            { return mayorIgual;
1060            }
1061          case 137: break;
1062          case 62:
1063            { return funcionSalirCase;
1064            }
1065          case 138: break;
1066          case 68:
1067            { return funcion;
1068            }
1069          case 139: break;
1070          case 29:
1071            { return diagonalInvertido;
1072            }
1073          case 140: break;
1074          case 30:
1075            { return signoAdmiracionAbre;
1076            }
1077          case 141: break;
1078          case 71:
1079            { return variable;
1080            }
1081          case 142: break;
1082          case 73:
1083            { return imprimir;
1084            }
1085          case 143: break;
1086          case 15:
1087            { return operadorY;
1088            }
1089          case 144: break;
```

```
            case 59:
              { return carácter;
              }
            case 145: break;
            case 38:
              { return comparacionIgualdad;
              }
            case 146: break;
            case 41:
              { return pi;
              }
            case 147: break;
            case 6:
              { return operadorDivision;
              }
            case 148: break;
            case 9:
              { return operadorMultiplicacion;
              }
            case 149: break;
            case 61:
              { return funcionSwitch;
              }
            case 150: break;
            case 52:
              { return valorEntero;
              }
            case 151: break;
            case 55:
              { return finLinea;
              }
            case 152: break;
            default:
```

```
              if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {
                zzAtEOF = true;
                return null;
              }
              else {
                zzScanError(ZZ_NO_MATCH);
              }
          }
        }
      }


  }

```

Archivo Tokens.java

El archivo Tokens.java es una pieza fundamental en la implementación de este analizador léxico, ya que define una enumeración que clasifica los diferentes tipos de tokens que el analizador puede identificar en el código fuente. Los tokens representan las unidades léxicas básicas del lenguaje, como palabras reservadas, operadores, identificadores, números y símbolos especiales, así como elementos estructurales como llaves, paréntesis y comentarios. Además, incluye un token genérico para errores, lo que permite gestionar y reportar entradas no reconocidas.

Este archivo actúa como una referencia central para el análisis y clasificación de los elementos léxicos, estableciendo una relación clara entre el código fuente y los componentes del lenguaje que se está interpretando o compilando. Esta enumeración facilita la implementación de reglas de análisis, mejora la legibilidad del código y contribuye a mantener una estructura ordenada y extensible para futuras actualizaciones o ampliaciones del lenguaje soportado por el analizador.

Código Realizado:

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package codigo;


public enum Tokens {
    Linea,
    Reservadas,
operadorSuma,
operadorResta,
operadorMultiplicacion,
operadorDivision,
operadorModulo,
operadorPotencia,
operadorRaiz,
asignacion,
incremento,
decremento,
operadorPositivo,
operadorNegativo,
comparacionIgualdad,
mayorQue,
menorQue,
mayorIgual,
menorIgual,
operadorY,
operadorO,
operadorNo,
letrasMin,
letrasMay,
numero,
inicioComentarioMult,
finalComentarioMult,
comentarioLinea,
valorEntero,
valorFlotante,
valorBooleano,
cadena,
carácter,
finLinea,

inicioTexto,
finalTexto,
parentIzq,
parentDer,
llaveIzq,
llaveDer,
corchetIzq,
corchetDer,
concatenar,
arreglo,

puntoDecimal,
dosPuntos,
espacio,
tabulacion,
comillaDoble,
comillaSimple,
coma,
guionBajo,
guionMedio,
diagonalInvertido,
diagonal ,
```

```
    signoAdmiracionAbre,
    signoAdmiracionCierra,
    gato,
    pesos,
    condicionalIf,
    condicionalElse,
    cicloFor,
    cicloWhile,
    cicloDo,
    funcionSwitch,
    funcionCase,
    funcionSalirCase,
    predeterminado,
    detener,
    constante,
    variable,
    funcion,
    clase,
    imprimir,
    importar,
    funcionLeer,
    funcionIntroducirDatos,
    pi,
    euler,
    Numero,
    Identificador,
        ERROR
    )
```

Archivo Principal.java

El archivo Principal.java actúa como el punto de entrada para la ejecución del analizador léxico en Java. Este archivo combina la inicialización de la interfaz gráfica con la ejecución del proceso que genera el analizador léxico a partir de una especificación escrita en JFlex. La clase implementa un splash screen mediante un hilo separado, que sirve como una introducción visual antes de cargar la interfaz principal del sistema, representada por el componente LoginUI.

Este archivo es un coordinador del flujo de ejecución del programa. Pues gestiona la experiencia del usuario al proporcionar un inicio visualmente atractivo y funcional. Además también automatiza la generación del archivo del analizador léxico mediante la herramienta JFlex, asegurando que el componente encargado del análisis de tokens esté disponible y actualizado. Esto lo convierte en una pieza esencial para la interacción fluida entre el usuario y el sistema, además de garantizar la eficiencia técnica del proceso de análisis léxico.

Código Realizado:

```java
package codigo;

import java.io.File;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Principal {

    public static void main(String[] args) {

        // Implementación del Runnable para el Splash y la pantalla principal
        Runnable mRun = () -> {
            // JFrame que funge como splash
            ModernSplash mSplash = new ModernSplash();
            mSplash.setVisible(true);

            try {
                Thread.sleep(10000); // 5000 milisegundos equivale a 5 segundos.
            } catch (InterruptedException ex) {
                Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null, ex);
            }

            mSplash.dispose();

            // JFrame que funge como pantalla principal
            LoginUI mFrmPrincipal = new LoginUI(); // Renombrado de FrmBienvenida
            mFrmPrincipal.setVisible(true);
        };

        // Creación e inicio del hilo para el Splash
        Thread mHiloSplash = new Thread(mRun);
        mHiloSplash.start();

        // Ejecución del método para generar el Lexer
        String ruta = "C:/Users/jesus/Downloads/AnalizadorLexico/src/codigo/Lexer.flex";
        generarLexer(ruta);
    }

    public static void generarLexer(String ruta) {
        File archivo = new File(ruta);
        JFlex.Main.generate(archivo);
    }
}
```

Archivo ModernSplash.java (Splash del proyecto)

El splash de este proyecto, llamado ModernSplash, es una pantalla inicial diseñada para ofrecer una experiencia visual atractiva mientras se cargan los recursos necesarios del sistema prácticamente es una ventana inicial moderna e interactiva que combina varias animaciones y efectos visuales para ofrecer una experiencia de bienvenida atractiva al usuario.. Este componente destaca por su diseño moderno, que incluye un fondo degradado en tonos azules, un título animado que realiza un efecto de caída y rebote simulado con física, y un icono que flota suavemente gracias a una animación sinusoidal. Además, incorpora una barra de progreso estilizada con colores dinámicos y mensajes de estado que informan al usuario sobre el avance de la carga, como "Preparando recursos..." y "Cargando módulos...".

La finalidad principal de este splash es mejorar la interacción inicial con el usuario, proporcionando un entorno visual que no solo entretiene, sino que también comunica claramente el progreso mientras se inicializan los módulos del proyecto. Este tipo de elemento no solo ayuda a gestionar la percepción del tiempo de carga, sino que también refuerza la identidad visual del proyecto, haciendo que la primera impresión sea profesional y agradable.

Código Realizado:

```java
package codigo;

import java.awt.*;
import javax.swing.*;
import java.awt.event.ActionEvent;

public class ModernSplash extends JFrame {
    private JProgressBar progressBar;
    private JLabel titleLabel, loadingLabel, iconLabel;
    private Timer progressTimer, fadeTimer, bounceTimer, colorTimer, floatTimer;
    private float opacity = 0.0f;
    private Color[] gradientColors = {new Color(76, 175, 80)};
    private int gradientIndex = 0;

    // Variables para animación del título
    private int titleYPosition = 50; // Posición inicial
    private int targetY = 170; // Posición final
    private double velocity = 0; // Velocidad inicial
    private double gravity = 1.2; // Simula la gravedad
    private int bouncesRemaining = 3; // Número de rebotes
    private double dampingFactor = 0.6; // Factor de amortiguación

    // Variables para animación flotante del ícono
    private int iconBaseY = 30; // Posición base del ícono
    private int floatAmplitude = 10; // Amplitud de la flotación
    private double floatAngle = 0; // Ángulo para la animación sinusoidal

    public ModernSplash() {
        initComponents();
    }
```

```java
    private void initComponents() {
        // Configuración principal del JFrame
        setUndecorated(true);
        setSize(400, 300);
        setOpacity(0.0f);
        setLocationRelativeTo(null);
        setLayout(null);

        // Fondo personalizado
        JPanel backgroundPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                Graphics2D g2d = (Graphics2D) g;
                g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

                // Degradado del fondo
                GradientPaint gradient = new GradientPaint(0, 0, new Color(30, 136, 229), getWidth(), getHeight(), new Color(63, 81, 181));
                g2d.setPaint(gradient);
                g2d.fillRoundRect(0, 0, getWidth(), getHeight(), 30, 30);
            }
        };
        backgroundPanel.setBounds(0, 0, 400, 300);
        backgroundPanel.setLayout(null);
        add(backgroundPanel);

        // Icono
        iconLabel = new JLabel(new ImageIcon("C:\\Users\\jesus\\Downloads\\AnalizadorLexico\\src\\codigo\\icono.png"));
        iconLabel.setBounds(125, iconBaseY, 150, 150);
        iconLabel.setHorizontalAlignment(SwingConstants.CENTER);
        backgroundPanel.add(iconLabel);

```

```
        // Título con animación
        titleLabel = new JLabel("Token-Visor");
        titleLabel.setBounds(50, titleYPosition, 300, 40);
        titleLabel.setFont(new Font("Roboto", Font.BOLD, 24));
        titleLabel.setForeground(Color.WHITE);
        titleLabel.setHorizontalAlignment(SwingConstants.CENTER);
        backgroundPanel.add(titleLabel);

        // Barra de progreso
        progressBar = new JProgressBar() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                Graphics2D g2d = (Graphics2D) g;
                g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

                int progressWidth = (int) (getWidth() * (getValue() / 100.0));
                g2d.setColor(gradientColors[gradientIndex]); // Color dinámico
                g2d.fillRoundRect(0, 0, progressWidth, getHeight(), 10, 10);
            }
        };
        progressBar.setBounds(50, 230, 300, 10);
        progressBar.setValue(0);
        backgroundPanel.add(progressBar);

        // Texto de carga
        loadingLabel = new JLabel("Iniciando...");
        loadingLabel.setBounds(50, 250, 300, 20);
        loadingLabel.setFont(new Font("Roboto", Font.PLAIN, 14));
        loadingLabel.setForeground(Color.WHITE);
        loadingLabel.setHorizontalAlignment(SwingConstants.CENTER);
        backgroundPanel.add(loadingLabel);

```

```
        // Iniciar animaciones
        startAnimations();
    }

    private void startAnimations() {
        // Fade-in de la ventana
        fadeTimer = new Timer(30, (ActionEvent e) -> {
            opacity += 0.02f;
            if (opacity >= 1.0f) {
                setOpacity(1.0f);
                fadeTimer.stop();
                startFloatAnimation(); // Inicia la animación flotante del ícono
                startBounceAnimation(); // Inicia la animación del título
            } else {
                setOpacity(opacity);
            }
        });
        fadeTimer.start();

        // Cambio de color dinámico para la barra de progreso
        colorTimer = new Timer(2000, (ActionEvent e) -> {
            gradientIndex = (gradientIndex + 1) % gradientColors.length;
            progressBar.repaint();
        });
        colorTimer.start();
    }

    private void startFloatAnimation() {
     floatTimer = new Timer(30, (ActionEvent e) -> { // Intervalo más largo para mayor fluidez
        // Animación sinusoidal para "flotar"
        floatAngle += 0.05; // Incremento más pequeño para ralentizar el movimiento
        int offsetY = (int) (Math.sin(floatAngle) * floatAmplitude);
        iconLabel.setBounds(125, iconBaseY + offsetY, 150, 150);
```

```
        });
        floatTimer.start();
    }


    private void startBounceAnimation() {
        bounceTimer = new Timer(20, (ActionEvent e) -> {
            velocity += gravity; // Incrementa velocidad por efecto de "gravedad"
            titleYPosition += velocity; // Actualiza posición del título

            // Detecta colisión con el suelo virtual
            if (titleYPosition >= targetY) {
                titleYPosition = targetY; // Ajusta la posición al suelo
                velocity = -velocity * dampingFactor; // Rebote con amortiguación
                bouncesRemaining--;

                // Si no quedan más rebotes, detiene la animación
                if (bouncesRemaining == 0) {
                    bounceTimer.stop();
                    titleYPosition = targetY; // Asegura posición final
                    startProgressBar(); // Inicia la barra de progreso
                }
            }

            // Actualiza la posición del título en el componente
            titleLabel.setBounds(50, titleYPosition, 300, 40);
        });

        bounceTimer.start();
    }

    private void startProgressBar() {
        progressTimer = new Timer(50, (ActionEvent e) -> {
            int value = progressBar.getValue();
            if (value < 100) {
                progressBar.setValue(value + 1);

                // Cambiar texto dinámicamente
                if (value < 30) {
                    loadingLabel.setText("Preparando recursos...");
                } else if (value < 60) {
                    loadingLabel.setText("Cargando módulos...");
                } else if (value < 90) {
                    loadingLabel.setText("Finalizando configuración...");
                } else {
                    loadingLabel.setText("¡Listo para comenzar!");
                }
            } else {
                progressTimer.stop();
                startFadeOut();
            }
        });
        progressTimer.start();
    }

    private void startFadeOut() {
        fadeTimer = new Timer(30, (ActionEvent e) -> {
            opacity -= 0.02f;
            if (opacity <= 0.0f) {
                setOpacity(0.0f);
                fadeTimer.stop();
                dispose(); // Cierra la ventana
            } else {
                setOpacity(opacity);
            }
        });
        fadeTimer.start();
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            ModernSplash splash = new ModernSplash();
            splash.setVisible(true);
        });
    }
}
```

Archivo LoginUI.java

El archivo LoginUI.java es una implementación de una interfaz gráfica de usuario (GUI) en Java diseñada para ofrecer una experiencia interactiva de inicio de sesión dentro del analizador léxico. Utilizando la biblioteca Swing, esta clase presenta una ventana dividida en dos paneles: un lado izquierdo visualmente atractivo con un ícono flotante animado y un mensaje de bienvenida, y un lado derecho funcional que contiene el formulario de inicio de sesión con campos para el usuario y la contraseña.
La importancia de este archivo radica en su combinación de funcionalidad y estética. Además de validar credenciales de acceso con una lógica sencilla, proporciona un diseño profesional que mejora la interacción del usuario mediante detalles como animaciones suaves y botones personalizados. Su estructura modular y enfoque en la usabilidad lo convierten en un componente clave del sistema, creando una primera impresión positiva y facilitando la transición hacia las funcionalidades principales del programa.

Código Realizado:

```java
package codigo;

import java.awt.*;
import javax.swing.*;
import java.awt.event.ActionEvent;

public class LoginUI extends JFrame {

    private JTextField registerNameField;
    private JPasswordField registerPasswordField;
    private JPanel leftPanel;
    private JLabel iconLabel;
    private Timer floatTimer;
    private int iconBaseY = 100; // Posición base del ícono en el eje Y
    private int floatAmplitude = 10; // Amplitud de la flotación
    private double floatAngle = 0; // Ángulo para la animación sinusoidal

    public LoginUI() {
        initComponents();
        startFloatAnimation(); // Inicia la animación flotante del ícono
    }

    private void initComponents() {
        setTitle("Login UI");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(800, 500);
        setLocationRelativeTo(null);
        setLayout(new GridLayout(1, 2));

        // Panel izquierdo (Bienvenida)
        leftPanel = new JPanel();
        leftPanel.setBackground(new Color(0, 38, 95));
        leftPanel.setLayout(null); // Posición absoluta para incluir el icono
```

```
        JLabel welcomeLabel = new JLabel("<html><center>¡Nos Alegra Verte De Nuevo!<br>Inicia sesión para continuar</center></html>");
        welcomeLabel.setForeground(Color.WHITE);
        welcomeLabel.setFont(new Font("Arial", Font.BOLD, 18));
        welcomeLabel.setHorizontalAlignment(SwingConstants.CENTER);
        welcomeLabel.setBounds(50, 300, 300, 100);
        leftPanel.add(welcomeLabel);

        // Ícono flotante
        iconLabel = new JLabel(new ImageIcon("C:\\Users\\jesus\\Downloads\\AnalizadorLexico\\src\\codigo\\icono.png"));
        iconLabel.setBounds(0, iconBaseY, 150, 150); // La posición X se ajustará dinámicamente
        leftPanel.add(iconLabel);

        add(leftPanel);

        // Panel derecho (Formulario de registro)
        JPanel rightPanel = new JPanel();
        rightPanel.setBackground(Color.WHITE);
        rightPanel.setLayout(new GridBagLayout());

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(10, 10, 10, 10);
        gbc.gridx = 0;
        gbc.anchor = GridBagConstraints.CENTER;

        JLabel createAccountLabel = new JLabel("Inicio De Sesión");
        createAccountLabel.setFont(new Font("Arial", Font.BOLD, 24));
        createAccountLabel.setForeground(Color.BLACK);
        gbc.gridy = 0;
        rightPanel.add(createAccountLabel, gbc);

        registerNameField = new JTextField(20);
        registerNameField.setFont(new Font("Arial", Font.PLAIN, 14));
        registerNameField.setBorder(BorderFactory.createTitledBorder("Usuario"));
        gbc.gridy = 1;
        rightPanel.add(registerNameField, gbc);

        registerPasswordField = new JPasswordField(20);
        registerPasswordField.setFont(new Font("Arial", Font.PLAIN, 14));
        registerPasswordField.setBorder(BorderFactory.createTitledBorder("Contraseña"));
        gbc.gridy = 2;
        rightPanel.add(registerPasswordField, gbc);

        CustomButton signUpButton = new CustomButton("Ingresar");
        signUpButton.setPreferredSize(new Dimension(200, 50));
        signUpButton.addActionListener(e -> {
            String name = registerNameField.getText();
            String password = new String(registerPasswordField.getPassword());

            // Validación simple
            if (name.equals("jesus") && password.equals("jesus123")) {
                JOptionPane.showMessageDialog(this, "Inicio de sesión exitoso");
                this.dispose(); // Cierra la ventana actual
                FrmPrincipal mainFrame = new FrmPrincipal();
                mainFrame.setVisible(true); // Abre la ventana principal
            } else {
                JOptionPane.showMessageDialog(this, "Credenciales inválidas. Intenta de nuevo.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        });
        gbc.gridy = 3;
        rightPanel.add(signUpButton, gbc);

        add(rightPanel);
    }

    private void startFloatAnimation() {
        floatTimer = new Timer(30, (ActionEvent e) -> { // Intervalo más largo para mayor fluidez
```

```
            // Animación sinusoidal para "flotar"
            floatAngle += 0.05; // Incremento más pequeño para ralentizar el movimiento
            int offsetY = (int) (Math.sin(floatAngle) * floatAmplitude);

            // Recalcula la posición horizontal centrada
            int iconX = (leftPanel.getWidth() - iconLabel.getWidth()) / 2;

            // Aplica la posición centrada y el desplazamiento vertical
            iconLabel.setBounds(iconX, iconBaseY + offsetY, 150, 150);
        });
        floatTimer.start();
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new LoginUI().setVisible(true);
        });
    }
}

// Clase personalizada para el botón
class CustomButton extends JButton {
    private boolean isHovered = false;

    public CustomButton(String text) {
        super(text);
        setContentAreaFilled(false);
        setFocusPainted(false);
        setBorderPainted(false);
        setFont(new Font("Arial", Font.BOLD, 16));
        setForeground(Color.WHITE);
        setCursor(new Cursor(Cursor.HAND_CURSOR));

        addMouseListener(new java.awt.event.MouseAdapter() {
            @Override
            public void mouseEntered(java.awt.event.MouseEvent e) {
                isHovered = true;
                repaint();
            }

            @Override
            public void mouseExited(java.awt.event.MouseEvent e) {
                isHovered = false;
                repaint();
            }
        });
    }

    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2d = (Graphics2D) g.create();
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        Color topColor = isHovered ? new Color(0, 120, 215) : new Color(33, 97, 140);
        Color bottomColor = isHovered ? new Color(0, 85, 170) : new Color(0, 51, 102);

        GradientPaint gradient = new GradientPaint(0, 0, topColor, 0, getHeight(), bottomColor);
        g2d.setPaint(gradient);
        g2d.fillRoundRect(0, 0, getWidth(), getHeight(), 30, 30);

        g2d.setColor(new Color(0, 0, 0, 50));
        g2d.fillRoundRect(2, 2, getWidth() - 4, getHeight() - 4, 30, 30);

        super.paintComponent(g2d);
        g2d.dispose();
    }

    @Override
    protected void paintBorder(Graphics g) {
    }
}
```

Archivo FrmPrincipal.java

El archivo FrmPrincipal.java representa la interfaz principal del analizador léxico, diseñada para simular un entorno de desarrollo integrado (IDE). Utilizando la biblioteca Swing, esta clase combina características interactivas y estéticas para ofrecer un espacio de trabajo funcional. Entre sus componentes destacan un área de entrada para escribir código, un panel de resultados donde se muestra el análisis realizado, una barra lateral que simula un explorador de archivos y una barra de estado que comunica el estado actual del programa.

Código Realizado:

```java
package codigo;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.logging.*;
import javax.swing.*;

public class FrmPrincipal extends javax.swing.JFrame {

    public FrmPrincipal() {
        initComponents();
        this.setLocationRelativeTo(null);
        this.setTitle("Analizador Léxico - IDE Simulado");
    }

    private void initComponents() {
        // Panel principal
        mainPanel = new JPanel(new BorderLayout());
        mainPanel.setBackground(new Color(30, 30, 30)); // Fondo oscuro principal

        // Barra lateral simulada (como árbol de archivos)
        JPanel sidebar = new JPanel();
        sidebar.setPreferredSize(new Dimension(150, 0));
        sidebar.setBackground(new Color(40, 40, 40));

        JLabel lblExplorer = new JLabel("Explorador");
        lblExplorer.setForeground(new Color(200, 200, 200));
        lblExplorer.setFont(new Font("Consolas", Font.BOLD, 14));
        lblExplorer.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        JList<String> fileTree = new JList<>(new String[]{"archivo.txt", "output.log", "README.md"});
        fileTree.setBackground(new Color(50, 50, 50));
```

```
        fileTree.setForeground(new Color(220, 220, 220));
        fileTree.setFont(new Font("Consolas", Font.PLAIN, 14));
        fileTree.setSelectionBackground(new Color(70, 130, 180));
        fileTree.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        sidebar.setLayout(new BorderLayout());
        sidebar.add(lblExplorer, BorderLayout.NORTH);
        sidebar.add(fileTree, BorderLayout.CENTER);

        // Barra de pestañas simulada
        tabBar = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 5));
        tabBar.setBackground(new Color(45, 45, 48));
        tabBar.setPreferredSize(new Dimension(0, 35));

        tab1 = new JLabel("archivo.txt");
        tab1.setForeground(new Color(220, 220, 220));
        tab1.setFont(new Font("Consolas", Font.BOLD, 14));
        tab1.setBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createMatteBorder(0, 0, 2, 0, new Color(97, 218, 251)),
            BorderFactory.createEmptyBorder(0, 10, 0, 10)));

        tabBar.add(tab1);

        // Panel para entrada y números de línea
        JPanel editorPanel = new JPanel(new BorderLayout());
        editorPanel.setBackground(new Color(28, 28, 28));

        // Números de línea
        lineNumberArea = new JTextArea("1\n2\n3\n4\n5\n");
        lineNumberArea.setFont(new Font("Consolas", Font.PLAIN, 16));
        lineNumberArea.setBackground(new Color(35, 35, 35));
        lineNumberArea.setForeground(new Color(150, 150, 150));
        lineNumberArea.setEditable(false);
        lineNumberArea.setMargin(new Insets(5, 5, 5, 5));

        // Área de entrada de código
        txtEntrada = new JTextArea();
        txtEntrada.setFont(new Font("Consolas", Font.PLAIN, 16));
        txtEntrada.setBackground(new Color(40, 44, 52));
        txtEntrada.setForeground(new Color(200, 200, 200));
        txtEntrada.setCaretColor(new Color(97, 218, 251));
        txtEntrada.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        txtEntrada.setText("// Escribe tu código aquí...");
        txtEntrada.setLineWrap(true);
        txtEntrada.setWrapStyleWord(true);

        JScrollPane scrollEntrada = new JScrollPane(txtEntrada);
        scrollEntrada.setRowHeaderView(lineNumberArea);
        scrollEntrada.setBorder(BorderFactory.createLineBorder(new Color(60, 63, 65)));

        editorPanel.add(scrollEntrada, BorderLayout.CENTER);

        // Área de resultados
        txtResultado = new JTextArea();
        txtResultado.setFont(new Font("Consolas", Font.PLAIN, 16));
        txtResultado.setBackground(new Color(35, 35, 35));
        txtResultado.setForeground(new Color(200, 200, 200));
        txtResultado.setCaretColor(new Color(97, 218, 251));
        txtResultado.setEditable(false);
        txtResultado.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        txtResultado.setLineWrap(true);
        txtResultado.setWrapStyleWord(true);
        txtResultado.setText("// Los resultados del análisis aparecerán aquí...");

        JScrollPane scrollResultado = new JScrollPane(txtResultado);
        scrollResultado.setBorder(BorderFactory.createLineBorder(new Color(60, 63, 65)));
```

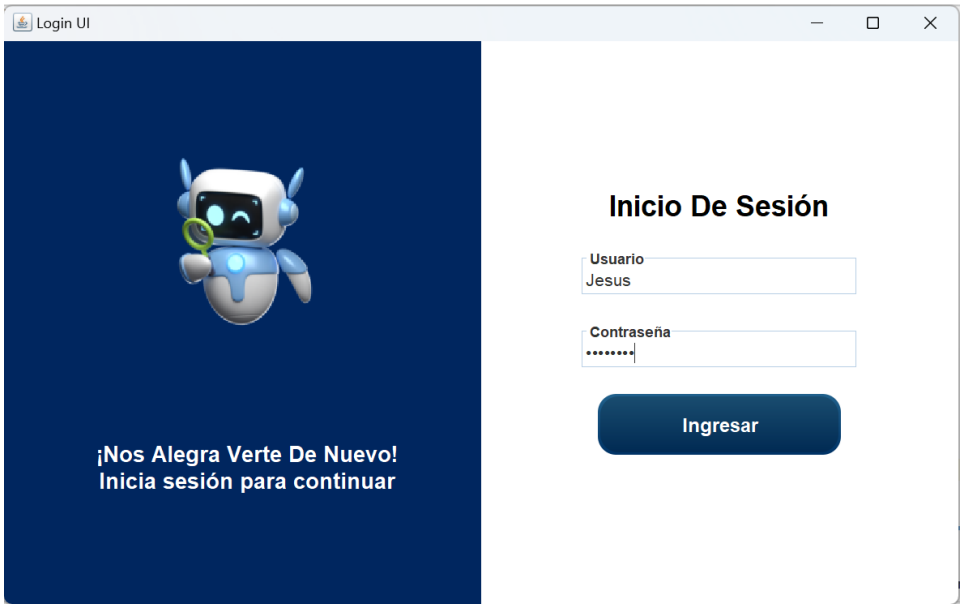
```
100
101         // Botón analizar
102         btnAnalizar = new JButton("Analizar Código");
103         btnAnalizar.setFont(new Font("Consolas", Font.BOLD, 16));
104         btnAnalizar.setBackground(new Color(97, 218, 251));
105         btnAnalizar.setForeground(new Color(28, 28, 28));
106         btnAnalizar.setFocusPainted(false);
107         btnAnalizar.setBorder(BorderFactory.createCompoundBorder(
108                 BorderFactory.createLineBorder(new Color(60, 63, 65)),
109                 BorderFactory.createEmptyBorder(10, 20, 10, 20)));
110
111         btnAnalizar.addActionListener(evt -> {
112             btnAnalizar.setText("Cargando...");
113             btnAnalizar.setEnabled(false);
114             btnAnalizar.setBackground(new Color(70, 130, 180));
115
116             Timer timer = new Timer(2000, e -> {
117                 btnAnalizarActionPerformed(evt);
118                 btnAnalizar.setText("Analizar Código");
119                 btnAnalizar.setEnabled(true);
120                 btnAnalizar.setBackground(new Color(97, 218, 251));
121             });
122             timer.setRepeats(false);
123             timer.start();
124         });
125
126         btnAnalizar.addMouseListener(new java.awt.event.MouseAdapter() {
127             @Override
129             public void mouseEntered(java.awt.event.MouseEvent evt) {
130                 btnAnalizar.setBackground(new Color(135, 206, 250));
131             }
132
                @Override
134             public void mouseExited(java.awt.event.MouseEvent evt) {
135                 btnAnalizar.setBackground(new Color(97, 218, 251));
136             }
137         });
138
139         // Barra de estado inferior
140         statusBar = new JLabel("Modo: Edición");
141         statusBar.setFont(new Font("Consolas", Font.PLAIN, 14));
142         statusBar.setForeground(new Color(220, 220, 220));
143         statusBar.setBackground(new Color(35, 35, 35));
144         statusBar.setOpaque(true);
145         statusBar.setBorder(BorderFactory.createEmptyBorder(5, 10, 5, 10));
146
147         // Panel inferior
148         JPanel bottomPanel = new JPanel(new BorderLayout());
149         bottomPanel.setBackground(new Color(28, 28, 28));
150         bottomPanel.add(btnAnalizar, BorderLayout.EAST);
151         bottomPanel.add(statusBar, BorderLayout.CENTER);
152
153         // Divisor ajustable
154         JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT, editorPanel, scrollResultado);
155         splitPane.setDividerLocation(300);
156         splitPane.setResizeWeight(0.7);
157         splitPane.setContinuousLayout(true);
158         splitPane.setBorder(null);
159
160         mainPanel.add(sidebar, BorderLayout.WEST);
161         mainPanel.add(tabBar, BorderLayout.NORTH);
162         mainPanel.add(splitPane, BorderLayout.CENTER);
163         mainPanel.add(bottomPanel, BorderLayout.SOUTH);
164
165         getContentPane().add(mainPanel);
166         pack();
```

```
166
167         // Atajos de teclado
168         addKeyboardShortcuts();
169     }
170
171     private void addKeyboardShortcuts() {
172         txtEntrada.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
173                 .put(KeyStroke.getKeyStroke(KeyEvent.VK_S, InputEvent.CTRL_DOWN_MASK), "save");
174         txtEntrada.getActionMap().put("save", new AbstractAction() {
175             @Override
176             public void actionPerformed(ActionEvent e) {
177                 statusBar.setText("Archivo guardado.");
178             }
179         });
180
181         txtEntrada.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
182                 .put(KeyStroke.getKeyStroke(KeyEvent.VK_R, InputEvent.CTRL_DOWN_MASK), "analyze");
183         txtEntrada.getActionMap().put("analyze", new AbstractAction() {
184             @Override
185             public void actionPerformed(ActionEvent e) {
186                 btnAnalizarActionPerformed(null);
187             }
188         });
189     }
190
191     private void btnAnalizarActionPerformed(java.awt.event.ActionEvent evt) {
192         statusBar.setText("Analizando...");
193         File archivo = new File("archivo.txt");
194         try (PrintWriter escribir = new PrintWriter(archivo)) {
195             escribir.print(txtEntrada.getText());
196         } catch (FileNotFoundException ex) {
197             Logger.getLogger(FrmPrincipal.class.getName()).log(Level.SEVERE, null, ex);
198         }
199
200         try (Reader lector = new BufferedReader(new FileReader("archivo.txt"))) {
201             Lexer lexer = new Lexer(lector);
202             StringBuilder resultado = new StringBuilder();
203             while (true) {
204                 Tokens tokens = lexer.yylex();
205                 if (tokens == null) {
206                     resultado.append("FIN");
207                     txtResultado.setText(resultado.toString());
208                     statusBar.setText("Análisis completado.");
209                     return;
210                 }
211                 switch (tokens) {
212                     case ERROR:
213                         resultado.append("Símbolo no definido\n");
214                         break;
215                     case Identificador:
216                     case Numero:
217                     case Reservadas:
218                         resultado.append(lexer.lexeme).append(": Es un ").append(tokens).append("\n");
219                         break;
220                     default:
221                         resultado.append("Token: ").append(tokens).append("\n");
222                         break;
223                 }
224             }
225         } catch (IOException ex) {
226             Logger.getLogger(FrmPrincipal.class.getName()).log(Level.SEVERE, null, ex);
227         }
228     }
229
230     public static void main(String args[]) {
231         java.awt.EventQueue.invokeLater(() -> new FrmPrincipal().setVisible(true));
232     }
233
234     // Variables
235     private JPanel mainPanel;
236     private JPanel tabBar;
237     private JLabel tab1;
238     private JTextArea txtEntrada;
239     private JTextArea txtResultado;
240     private JTextArea lineNumberArea;
241     private JButton btnAnalizar;
242     private JLabel statusBar;
243 }
```

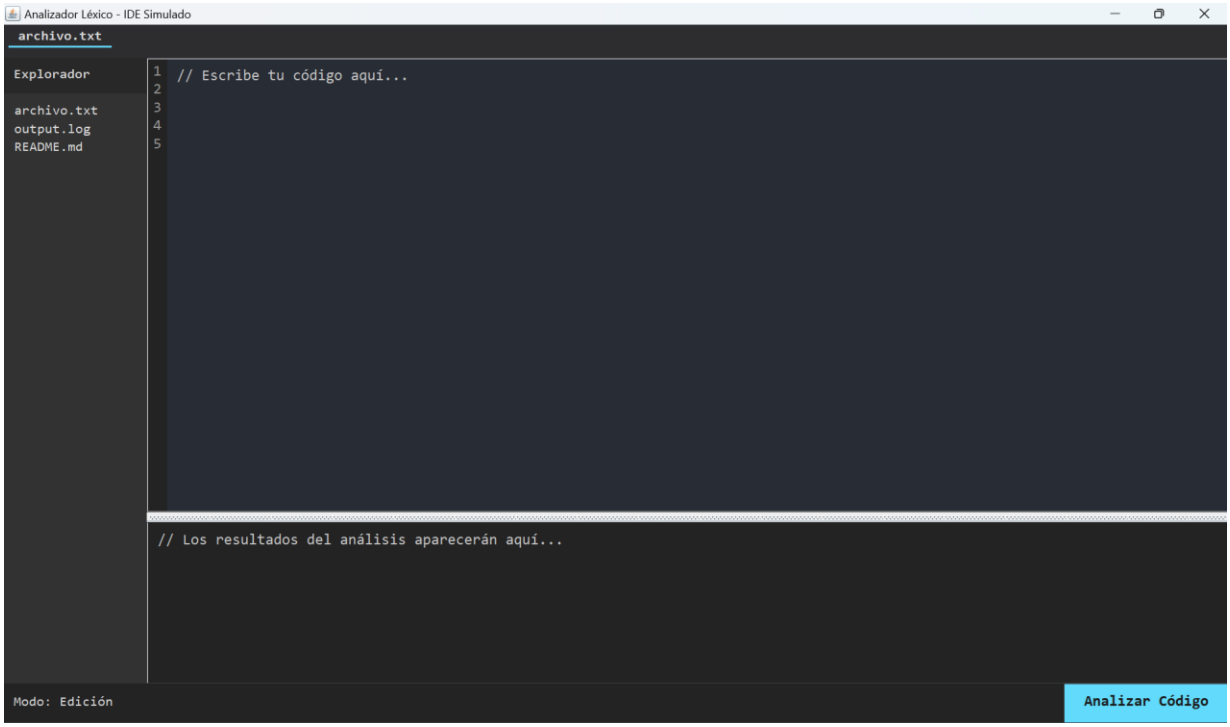
PANTALLAS RESULTANTES CON PRUEBAS DEL RECONOCIMIENTO DE LOS TOKENS

Pantalla Splash

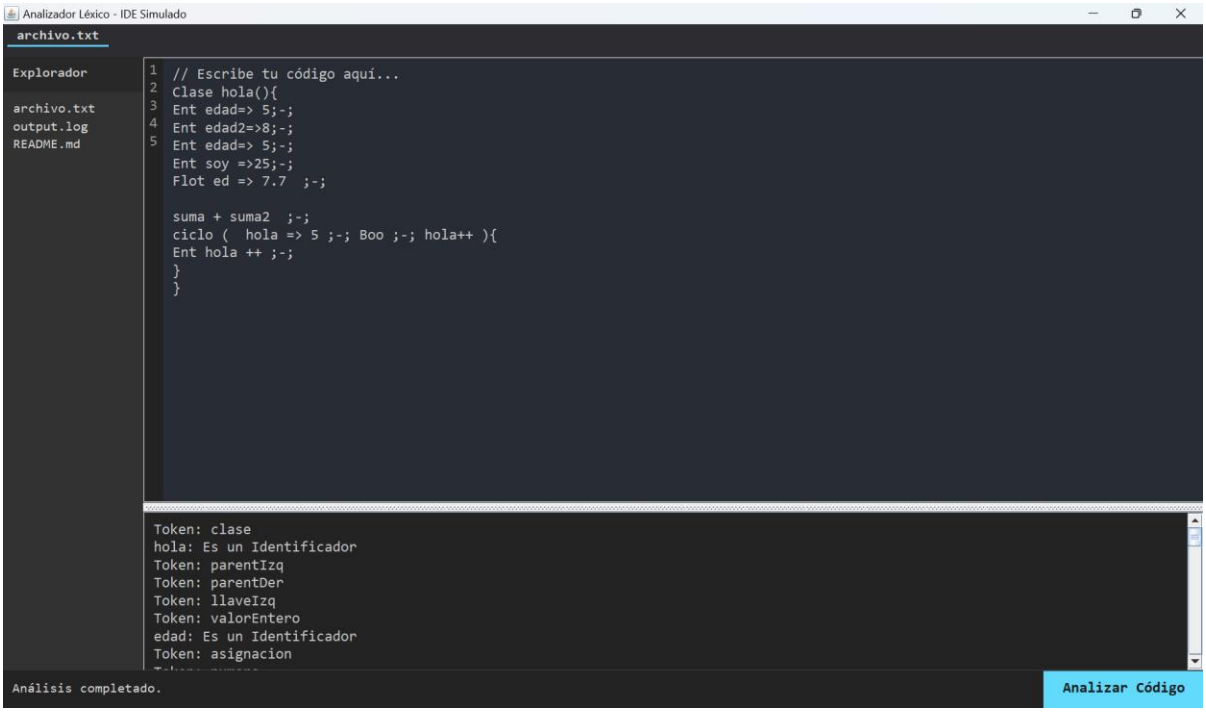
Pantalla Login



Pantalla Principal



Pantalla principal con demostración de la verificación de los tokens



V. Conclusiones:

El objetivo principal del desarrollo de este proyecto fue implementar un analizador léxico que permitiera identificar y clasificar elementos sintácticos de un lenguaje de programación simulado. Este proyecto buscó no solo reforzar los conocimientos teóricos adquiridos sobre compiladores, sino también aplicar habilidades prácticas en el desarrollo de software, trabajando con herramientas y tecnologías modernas.

Para llevar a cabo el proyecto, se utilizó el lenguaje de programación **Java**, el cual proporcionó una plataforma sólida y orientada a objetos para estructurar el sistema. Dentro de las herramientas y librerías empleadas, destaca **JFlex**, que permitió generar un analizador léxico basado en expresiones regulares definidas en un archivo .flex. Esta herramienta automatizó gran parte del proceso de construcción del lexer, transformando reglas léxicas en código ejecutable. Asimismo, se usó **Swing**, la biblioteca gráfica de Java, para diseñar una interfaz de usuario amigable que simula las funcionalidades de un IDE básico. Componentes personalizados, como botones estilizados y paneles con efectos visuales, añadieron un toque profesional al diseño de la aplicación. Además, se incorporaron herramientas estándar de Java para gestionar archivos, manejar eventos y mostrar resultados en tiempo real.

Durante el desarrollo del proyecto uno de los principales desafíos fue diseñar reglas léxicas precisas en el archivo .flex, pues cualquier error mínimo en las expresiones regulares podía causar un mal funcionamiento del lexer, lo que podía provocar varias ejecuciones de prueba y corrección. Otro reto significativo fue la integración de la interfaz gráfica con la lógica del analizador léxico. Garantizar que las acciones del usuario en la interfaz fueran correctamente interpretadas por el sistema requería una sincronización cuidadosa entre la capa visual y el backend. Asimismo, implementar una experiencia de usuario intuitiva, con mensajes de error claros y una respuesta visual atractiva, representó un esfuerzo adicional.
A pesar de las dificultades, este proyecto me permitió desarrollar algunas habilidades importantes, como la escritura de expresiones regulares, la programación de interfaces gráficas intuitivas y agradables visualmente ,así como la solución de problemas en tiempo real.