Universidad Autónoma de Coahuila

PROYECTO FINAL

GENERADOR DE TABLA DE TOKENS

Materia: Calidad Y Pruebas De Software

Docente: Carlos Nassif Trejo García

Integrantes:

Adriana Berenice Maldonado García

Abigail Silva Díaz

Yadira Judith Cordero Morales

Objetivo de la materia:

El estudiante realizara un proyecto/programa funcional donde mostrara y aplicara lo visto en clase a dicho proyecto.

Objetivo del proyecto:

Implementar un sistema de autenticación seguro mediante la generación y gestión de tokens únicos.

Glosario:

- Token: Elemento único que representa algo mas grande o un identificador para realizar una acción.
- Palabras reservadas: Son términos específicos dentro de un lenguaje de programación.
- Autómata: Modelo abstracto de una maquina que puede cambiar su estado en respuesta a ciertas entradas.
- Matriz de transición: Representa como un autómata cambia de un estado a otro en respuesta a ciertas entradas.

Procedimientos del programa:

Realizar una tabla en la cual se le asigna un valor a cada uno;
Los tokens siempre llevan de estados (-) al principio.

Palabras Reservadas		Operadores Aritméticos		Operadores Relacionales		Operadores Lógicos		Caracteres Que generan token		Identificadores		Constantes	
public	-1	+	-21	<	-31	!	-41	;	-51	String	-61	String	-71
private	-2	-	-22	<=	-32	&&	-42	[	-52	Real	-62	Real	-72
class	-3	*	-23	>	-33	||	-43	]	-53	Entero	-63	Entero	-73
void	-4	/	-24	>=	-34			,	-54	Rutina	-64	Comentario	-74
int	-5	%	-25	==	-35			:	-55				
double	-6	=	-26	!=	-36			(	-56				
String	-7							)	-57				
new	-8							{	-58				
nextLine	-9							}	-59				
nextInt	-10												
nextDouble	-11												
Println	-12												
if	-13												
else	-14												
do	-15												
while	-16												
switch	-17												
case	-18												
default	-19												

Una vez teniendo la tabla de tokens se diseña un autómata llamando cada estado y cada token.

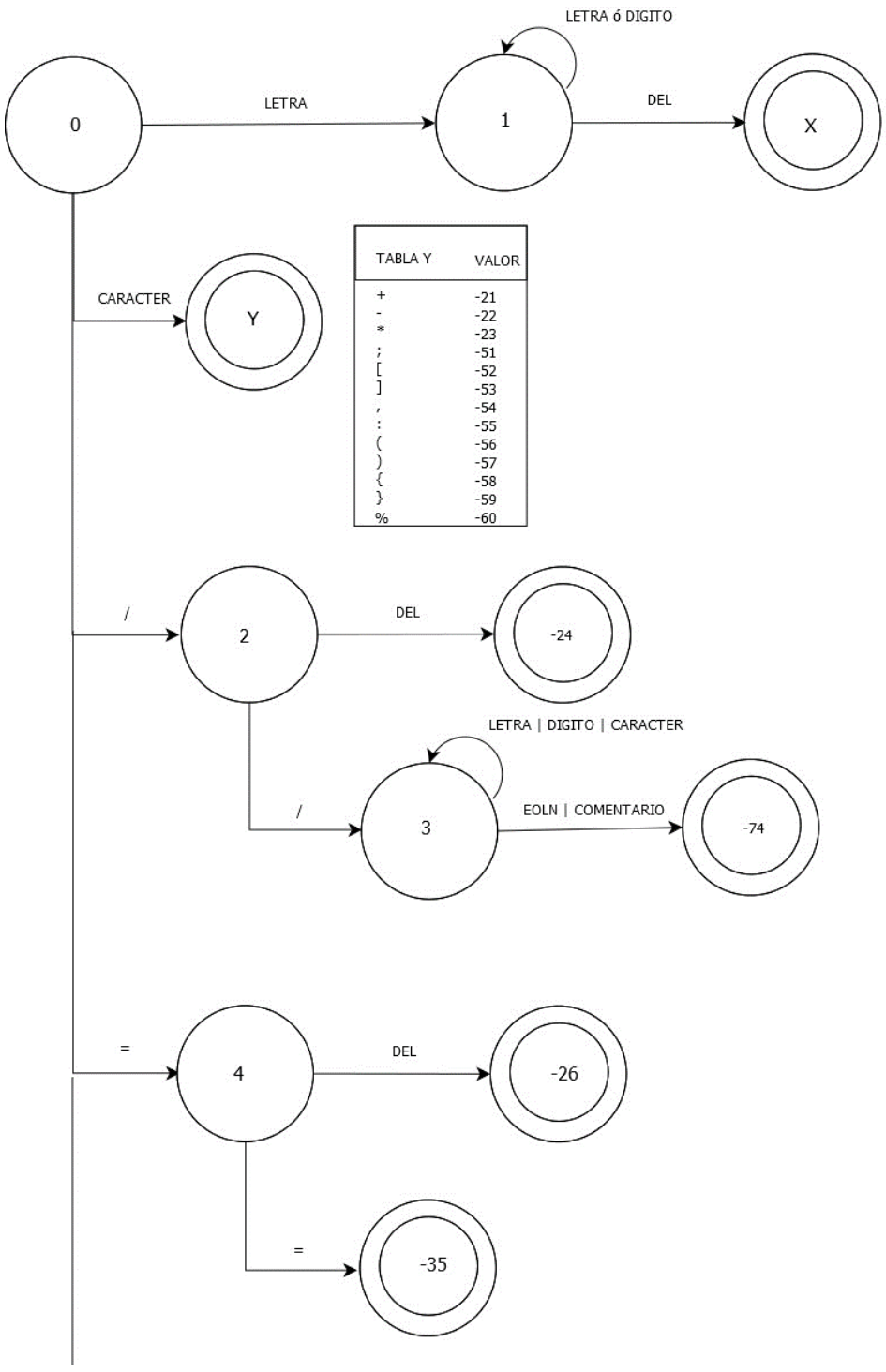
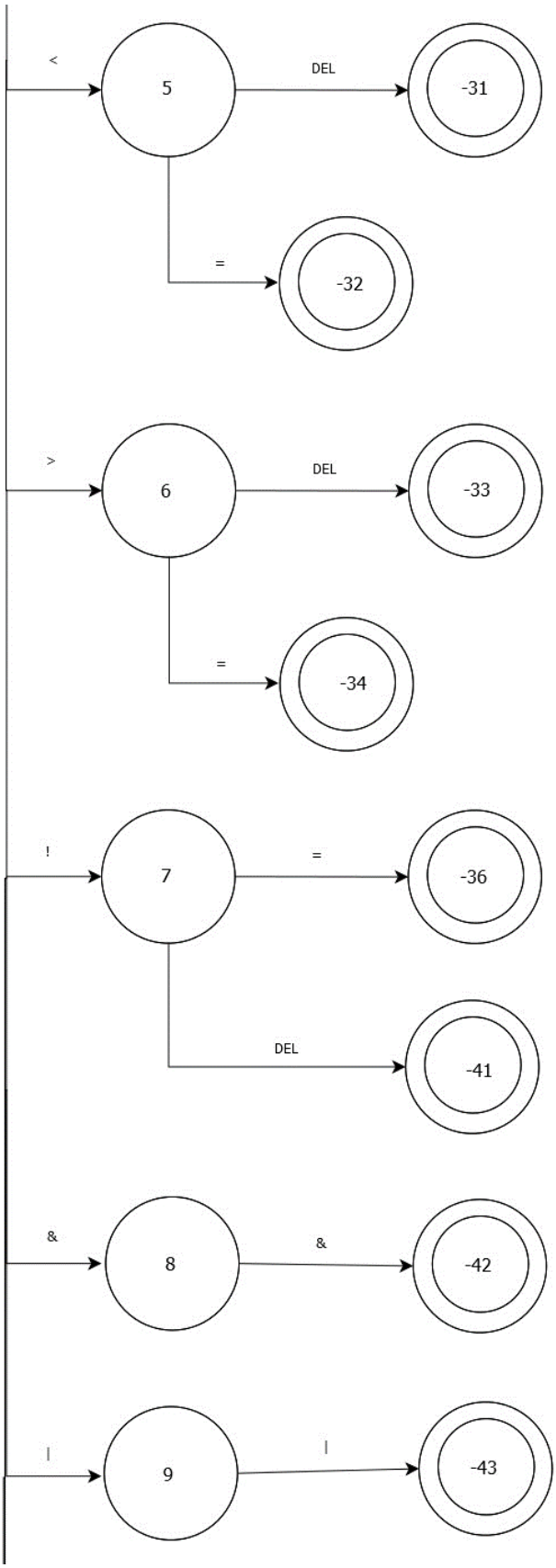
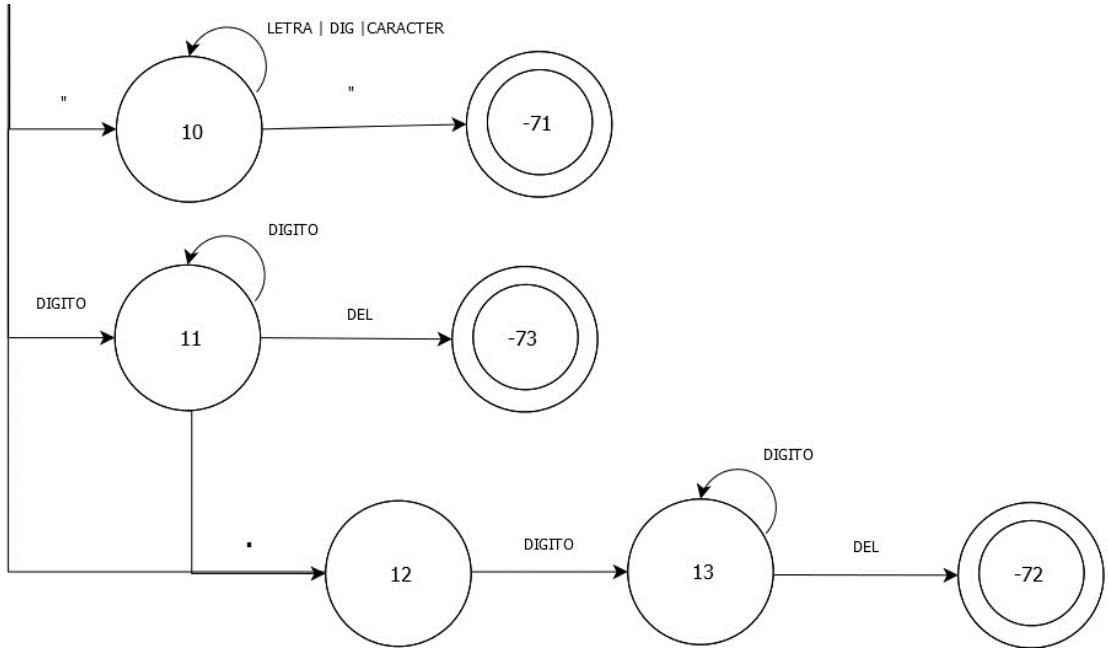


TABLA X	VALOR
default	-19
case	-18
public	-1
private	-2
class	-3
void	-4
int	-5
double	-6
string	-7
new	-8
nextLine	-9
nextInt	-10
nextDouble	-11
println	-12
if	-13
else	-14
do	-15
while	-16
switch	-17

TABLA Y	VALOR
+	-21
-	-22
*	-23
;	-51
[	-52
]	-53
,	-54
:	-55
(	-56
)	-57
{	-58
}	-59
%	-60


<
5
DEL
-31
=
-32
>
6
DEL
-33
=
-34
!
7
=
-36
DEL
-41
&
8
&
-42
|
9
|
-43



Realizado el autómata se pasa de forma matemática a la matriz de transición.

Estados	letras	digitos	carácter	/	=	<	>	!	&	|	"	.	\n
0	1	11	Y	2	4	5	6	7	8	9	10	12	0
1	X	X	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR
2	ERROR	ERROR	ERROR	3	-24	-24	-24	-24	-24	-24	-24	-24	-24
3	3	3	3	3	3	3	3	3	3	3	3	3	-74
4	-26	-26	-26	-26	-35	-26	-26	-26	-26	-26	-26	-26	-26
5	-31	-31	-31	-31	-32	ERROR	ERROR	-31	-31	-31	-31	-31	-31
6	-33	-33	-33	-33	-34	ERROR	ERROR	-33	-33	-33	-33	-33	-33
7	-41	-41	-41	-41	-36	-41	-41	-41	-41	-41	-41	-41	-41
8									-42				
9										-43			
10	10	10	10	10	10	10	10	10	10	10	-71	10	10
11		11	-73	-73	-73	-73	-73	-73	-73	-73	-73	12	-73
12		12											
13		13	-72	-72	-72	-72	-72	-72	-72	-72	-72	-72	-72

Programa diseñado en java:

```
package Lex;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Scanner;

public class Lex {
	String letras =
"ABCDEFGHIJKLMNÑOPQRSTUVWXYZabcdefghijklmnñopqrstuvwxyz";
	String digitos = "0123456789";
	String caracteres = "+-=;[],:(){}";
	String lenguaje = "LDC/%=<>!&|\". \t\n";
	String codigo;
	int NumeroDeLinea = 1;
	LinkedList<String[]> TablaDeTokens = new LinkedList();
	String matriz [][] = {
{"1", "9", "Y", "12", "-25", "2", "3", "4", "5", "6", "7", "8", "10",
"ERROR", "ERROR", "0", "0", "0", "0"},

{"1", "1", "X", "X", "-62", "X", "X", "X", "X", "-63", "X", "X", "X", "-
64", "-61", "X", "X", "X", "X"},

{"-26D", "-26D", "-26D", "-26D", "-26D", "-35", "-26D", "-26D", "-26D",
"-26D", "-26D", "-26D", "-26D", "-26D", "-26D", "-26D", "-26D", "-26D",
"-26D"},

{"-31D", "-31D", "-31D", "-31D", "-31D", "-32", "-31D", "-31D", "-31D",
"-31D", "-31D", "-31D", "-31D", "-31D", "-31D", "-31D", "-31D", "-31D",
"-31D"},

{"-33D", "-33D", "-33D", "-33D", "-33D", "-34", "-33D", "-33D", "-33D",
"-33D", "-33D", "-33D", "-33D", "-33D", "-33D", "-33D", "-33D", "-33D",
"-33D"},

{"-41D", "-41D", "-41D", "-41D", "-41D", "-36", "-41D", "-41D", "-41D",
"-41D", "-41D", "-41D", "-41D", "-41D", "-41D", "-41D", "-41D", "-41D",
"-41D"},

{"ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR",
"ERROR", "-42", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR",
"ERROR", "ERROR", "ERROR"},

{"ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR",
"ERROR", "ERROR", "-43", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR",
"ERROR", "ERROR", "ERROR"},

{"8", "8", "8", "8", "8", "8", "8", "8", "8", "8", "8", "-71", "8", "8",
"8", "8", "8", "ERROR", "ERROR"},

{"ERROR", "9", "-73D", "-73D", "-73D", "-73D", "-73D", "-73D", "-73D", "-
73D", "-73D", "-73D", "10", "-73D", "-73D", "-73D", "-73D", "-73D", "-
73D"},
```

```
{"ERROR", "11", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR",
"ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR",
"ERROR", "ERROR", "ERROR"},

{"ERROR", "11", "-72D", "-72D", "-72D", "-72D", "-72D", "-72D", "-72D",
"-72D", "-72D", "-72D", "ERROR", "-72D", "-72D", "-72D", "-72D", "-72D",
"-72D"},

{"-24", "-24", "-24", "13", "-24", "-24", "-24", "-24", "-24", "-24", "-
24", "-24", "-24", "-24", "-24", "-24", "-24", "-24", "-24"},

{"13", "13", "13", "13", "13", "13", "13", "13", "13", "13", "13", "13",
"13", "13", "13", "13", "13", "-74D", "-74D"}
};

String x [][] = {{"public", "-1"}, {"private", "-2"}, {"class", "-3"},
{"void", "-4"}, {"int", "-5"}, {"double", "-6"}, {"string", "-7"},
{"new", "-8"}, {"nextLine", "-9"}, {"nextInt", "-10"},{"nextDouble", "-
11"}, {"println", "-12"}, {"if", "-13"}, {"else", "-14"}, {"double", "-
15"}, {"while", "-16"}, {"switch", "-17"}, {"case", "-18"}, {"default",
"-19"}};

String y [][] = {{"+", "-21"}, {"-", "-22"}, {"*", "-23"}, {";", "-51"},
{"[", "-52"}, {"]", "-53"},{",", "-54"},{":", "-55"}, {"(", "-56"}, {")",
"-57"}, {"{", "-58"}, {"}", "-59"}, {"%", "-60"}};
      public void Lexemas(String Lexema, int posicion, int estado, int
Linea) {
            NumeroDeLinea = Linea;
            int columna = -1;
            if(estado == 0) {
                  Lexema = "";
            }
            if(codigo.length()<= posicion) {
                  return;
            }
            if(codigo.charAt(posicion) == '\n') {
                  Linea++;
            }
            if(letras.contains(codigo.charAt(posicion)+"")) {
                  columna = 0;
            }

            else if(lenguaje.contains(codigo.charAt(posicion)+"")) {
                  columna = lenguaje.indexOf(codigo.charAt(posicion));
            }
            else if(digitos.contains(codigo.charAt(posicion)+"")) {
                  columna = 1;
      }
            else if(caracteres.contains(codigo.charAt(posicion)+"")) {
                  columna = 2;
            }
            try {
                  int NuevoEstado =
Integer.parseInt(matriz[estado][columna]);
                  if (NuevoEstado>=0) {
```

```
                        this.Lexemas(Lexema+codigo.charAt(posicion),
posicion+1, NuevoEstado, Linea);
                  }

                  else {
                        if((int)(NuevoEstado/10) == -6) {
                              TablaDeTokens.add(new String[]
{Lexema+codigo.charAt(posicion), NuevoEstado+"" , "-2" , "" +
NumeroDeLinea });
                        }
                        else {
                              TablaDeTokens.add(new String[]
{Lexema+codigo.charAt(posicion), NuevoEstado+"" , "-1" , "" +
NumeroDeLinea });
                        }

                        this.Lexemas("", posicion+1, 0, Linea);

            }}
            catch(Exception exs ) {
                  try {
                        int NuevoEstado =
Integer.parseInt(matriz[estado][columna].substring(0, 3));
                        if(NuevoEstado == -73) {
                              System.out.println(Lexema);
                              if(Double.parseDouble(Lexema)>32767 ||
Double.parseDouble(Lexema)< -32768) {
                                    NuevoEstado = -72;
                              }
                        }
                        TablaDeTokens.add(new String[] {Lexema,
NuevoEstado+"" , "-1" , "" + NumeroDeLinea });
                        this.Lexemas("", posicion, 0, Linea);
                  }
                  catch(Exception exs2) {
                        String exe = matriz[estado][columna];
                        if(exe.compareTo("X") == 0) {
                              TablaDeTokens.add(new String[] {Lexema,
""+BuscarX(Lexema) , "-1" , "" + NumeroDeLinea });
                              this.Lexemas("", posicion, 0, Linea);
                        }
                        else if(exe.compareTo("Y") == 0) {
                              TablaDeTokens.add(new String[]
{Lexema+codigo.charAt(posicion),
""+BuscarY(Lexema+codigo.charAt(posicion)) , "-1" , "" + NumeroDeLinea
});
                              this.Lexemas("", posicion+1, 0, Linea);
                        }
                        else {
                              TablaDeTokens.add(new String[]
{Lexema+codigo.charAt(posicion), exe, "-1" , "" + NumeroDeLinea });
                              this.Lexemas("", posicion+1, 0, Linea);
                        }
                  }

            }
```

```
		}
	public String BuscarX(String Lexema) {
		for(String[] Fila: x) {
			if(Fila[0].compareTo(Lexema)==0){
				return Fila[1];
			}
		}
		return "ERROR";
	}
	public String BuscarY(String Lexema) {
		for(String[] Fila: y) {
			if(Fila[0].compareTo(Lexema)==0){
				return Fila[1];
			}
		}
		return
				"ERROR";
	}

	public void Escribir() throws IOException {
		File xd = new File("Tabla de tokens.txt");
		if(!xd.exists()) {
			xd.createNewFile();
		}
		FileWriter fw = new FileWriter(xd);
		BufferedWriter bw = new BufferedWriter(fw);
		bw.write("Lexema \t|Token\t|Pertenece\t|#Linea\n");
		for(String[] linea:TablaDeTokens) {
			for(String celda: linea) {
				bw.write(celda+"\t|");
			}
			bw.write("\n");
		}
		bw.close();
	}
	public void Leer(String nombre) throws FileNotFoundException {
		File xd = new File(nombre);
		if(xd.exists()) {
			Scanner s = new Scanner(xd);
			codigo = "";

			while(s.hasNext()) {
				codigo += s.nextLine() +  "\n";
			}
		}
	}
	public static void main (String[] args) throws IOException {

		Lex L = new Lex();
		L.Leer("texto.txt");
		System.out.print(L.codigo);
		L.Lexemas("", 0, 0, 1);
		L.Escribir();
	}
}
```

Pruebas unitarias:

Mock testing: reemplaza componentes del sistema que no están disponibles. Simula comportamiento especifico para probar el código de forma controlada y eficiente.

En el código:
Al momento de borrar o quitar componentes de la matriz (Linked List) esta misma sabe que no esta completo y que faltan datos.

```
String matriz [][] = {
        {"1", "9", "X", "12", "-25", "2", "3", "4", "5", "6", "7", "8", "10"},
        {"1", "1", "X", "X", "-62", "X", "X", "X", "X", "-63", "X", "X", "X", "-64", "-61", "X", "X", "X", "X"},
        {"-26D", "-26D", "-26D", "-26D", "-26D"},
        {"-31D", "-31D", "-31D", "-31D", "-31D", "-32", "-31D", "-31D", "-31D", "-31D", "-31D", "-31D", "-31D", "
        {"-33D", "-33D", "-33D", "-33D", "-33D", "-34",},
        {"-41D", "-41D", "-41D", "-41D", "-41D", "-36", "-41D", "-41D", "-41D", "-41D", "-41D", "-41D", "-41D", "
        {"ERROR",  "ERROR", "ERROR", "ERROR"},
        {"ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "-43", "ERROR"
        {"8", "8", "8", "8", "8", "8", "8", "8", "8","ERROR"},
        {"10", "-73D", "-73D", "-73D", "-73D", "-73D", "-73D"},
        {"ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR", "ERROR"},
        {"ERROR", "11", "-72D", "-72D", "-72D", "-72D", "-72D", "-72D", "-72D", "-72D", "-72D", "-72D", "ERROR",
        {"-24", "-24", "-24", "13", "-24", "-24", "-24", "-24", "-24", "-24", "-24", "-24", "-24", "-24", "-24",
        {"13", }
};
```

Y esto es lo que arrojaría de errores.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 15 out of bounds for length 13
        at Lex.Lex.Lexemas(Lex.java:95)
        at Lex.Lex.Lexemas(Lex.java:106)
        at Lex.Lex.Lexemas(Lex.java:106)
        at Lex.Lex.Lexemas(Lex.java:106)
        at Lex.Lex.Lexemas(Lex.java:106)
        at Lex.Lex.Lexemas(Lex.java:106)
        at Lex.Lex.Lexemas(Lex.java:106)
        at Lex.Lex.main(Lex.java:163)
```

Data Flow testing: Los datos se mueven y cambian a través del programa. Esta puede identificar áreas donde se utilizan datos no inicializados o se espera un cierto lujo de datos. El objectivo es cubrir diferentes rutas que los datos pueden tomar a través del código.

En el código:

La condición para el token -73 representa los números reales y enteros en lo cual tiene como limite 32767 y si este se cambia o modifica con números menores.

```
if(NuevoEstado == -73) {
    System.out.println(Lexema);
    if(Double.parseDouble(Lexema)>32 || Double.parseDouble(Lexema)< -32) {
        NuevoEstado = -72;
    }
}
```

El error no te hará cambios a números con decimal y el token será el mismo.

Lexema	Token	Pertenece		#Linea
32767	-72	-1	1	
327677856		-72	-1	3
32.4	-72	-1	5	

Unit testing: Se prueban unidades individuales o de componentes aislados como funciones, métodos o clases.

En el programa:
Se pueden encontrar métodos para la tabla X y tabla Y, si estas se modifican, no mandaran a ningún lado, ya que esas son únicas de token.

```
public String BuscarX(String Lexema) {
    for(String[] Fila: x) {
        if(Fila[0].compareTo(Lexema)==0){
            return Fila[1];
        }
    }
    return "ERROR";
}
public String BuscarY(String Lexema) {
    for(String[] Fila: y) {
        if(Fila[0].compareTo(Lexema)==0){
            return Fila[1];
        }
    }
    return "ERROR";
}
```

Conclusiones:

Abigail:

Para concluir este proyecto considero que es muy interesante ver que se necesita tener demasiados detalles para llegar a concluir el proyecto se manera satisfactoria, y no solo llevando a cabo un tipo de prueba, si no varios que conlleva poder terminar y obtener los mejores resultados posibles cumpliendo con los requerimientos puestos al inicio del trabajo. En lo personal fue muy interesante y a la vez complicado tener bien definidos los diferentes tipos de pruebas y métodos de llevar a cabo las pruebas de un solo proyecto, pero considero que hasta ahora esa información ya quedo más clara y el conocimiento más sólido.

Adriana:

El desarrollo de este proyecto, se centro en comprobar la calidad, confiabilidad y eficacia del código, la implementación de este código es esencial en la construcción de compiladores y traductores, y la calidad del análisis léxico influye directamente en la calidad del proceso de compilación en su conjunto. Ya que surge la necesidad de llevar a cabo pruebas para verificar la capacidad del analizador de léxico para reconocer y clasificar correctamente los diferentes lexemas, especialmente en situaciones específicas como lo fueron los casos que se mostraron en este trabajo.

Este proyecto no solo demostró la implementación práctica del código, sino también la importancia crucial de las pruebas de calidad en el proceso de desarrollo de software para garantizar la entrega de un producto final confiable y funcional. Este enfoque de desarrollo centrado en pruebas contribuye a la creación de software más sólido y resistente, fundamental en aplicaciones críticas como los compiladores.

Yadira:

En este proyecto, se ha desarrollado un programa para la generación de tokens. Con la finalidad de representar y gestionar los tokens, en la busca de optimización en el rendimiento y eficacia de este mismo.

Este proyecto ayuda a la mejora mas eficiente de datos y reduce la complejidad de almacenamiento, además de ofrecer una mayor flexibilidad y escalabilidad al sistema; esta a sido una estrategia para explorar y aprovechar herramientas y estructuras, dependiendo de las especificaciones del programa.