



Primer proyecto – Compiladores e intérpretes

Analizador léxico

Profesor:

Allan Rodríguez Dávila

Estudiantes:

Jairo González Hidalgo - 2024178535

Rafael Odio Mendoza - 2024083499

26 de diciembre – Verano 2025

# Manual de usuario

## Compilación

Para que el programa funcione, es necesario tener instalado el JDK de Java (se recomienda la versión 11 o superior). Si se usa VS Code, hay que instalar el entorno de Java, se recomienda el "Extension Pack for Java" de Microsoft. También pueden usar otros IDEs como IntelliJ IDEA o Eclipse sin problema.

El proyecto usa JFlex y CUP para generar el analizador léxico. Estos se encuentran como archivos .jar en la carpeta lib/ y no necesitan instalación adicional.

El proyecto fue diseñado para que no hubiera que estar usando comandos para compilar el Lexer y el Parser cada vez que se cambiara algo. En el menú principal, la Opción 1 se encarga de borrar los archivos .java viejos en la carpeta parserlexer, ejecutar el jar de cup para generar la clase de símbolos (sym.java) y ejecutar el jar de JFlex para generar el motor del scanner (Lexer.java).

Para crearlos manualmente sin usar main, primero hay que estar dentro de la carpeta programa: `\proyecto-compiladores-e-interpretes> cd programa`

Posteriormente se ejecutan

Lexer.java: `java -jar lib/jflex-full-1.9.1.jar parserlexer/Lexer.jflex`

sym.java (que también compila Parser.java): `java -jar lib/java-cup-11b.jar -destdir parserlexer/ -parser Parser parserlexer/Parser.cup`

Los archivos generados quedan en la carpeta parserlexer/

## Ejecución

Ejecutar el archivo Main.java, desplegará un menú.

## Uso

La opción 0 cierra el programa.

Para crear los archivos Lexer.java y sym.java se usa la opción 1.

Para ejecutar el análisis se usa la opción 2, se escogen los archivos, esta muestra una lista de todos los archivos .txt en la carpeta archivos\_prueba/, puede seleccionar un archivo específico o analizar todos mediante la opción 0, los archivos son listados individualmente con un número que permite seleccionarlos.

Por cada archivo analizado se genera un reporte en archivos\_salida/ con el formato reporte\_nombre.txt

El reporte incluye una tabla con cada token encontrado (identificador, lexema, línea, columna) y una lista de errores léxicos si los hubiera.

Se pueden ver más pruebas del uso en el siguiente apartado.

## Pruebas de funcionalidad

Se pueden apreciar completamente las pruebas en el video:

<https://www.youtube.com/watch?v=kATyjgQVY2M>

## 1. Generación de archivos .java con la opción 1

The screenshot shows an IDE interface with several tabs and panes:

- Project Tree:** Shows files like lib, parserlexer, Lexerflex, Parser.cup, Main.java, and info.txt.
- Code Editor:** Displays Java code for a lexer and parser. The code includes imports for java.util.List, java.io.File, and java.util.ArrayList. It defines a Main class with a main method that reads from 'info.txt' and prints errors to the console. A static method 'ejecutarLexer' takes a file as input and prints errors to the console.
- Terminal:** Shows a command-line interface with several error messages related to the Main.java file, indicating it cannot be found or loaded.
- Bottom Bar:** Includes links for OUTLINE, TIMELINE, and JAVA PROJECTS, along with status icons for main\*, Launchpad, Java Ready, and session information.
- Status Bar:** Shows the selection 'Seleccione una opción: 1]'.

The screenshot shows a code editor interface with several Java files listed in the left sidebar:

- parserlexer
- J Lexer.java
- E Lexer.jflex
- E Parser.cup
- J Parser.java
- J sym.java
- J Main.java
- info.txt

The terminal window at the bottom displays the following output:

```

PS C:\Users\Jairo\Desktop\Compiladores e interpretes\proyecto\proyecto
ladores-e-interpretes'; & 'C:\Program Files\Java\jre1.8.0_471\bin\java
2. Realizar Análisis Léxico
0. Salir
Seleccione una opción: 1
Borrando archivos antiguos...
Generando sym y Parser con CUP...
Generando Lexer con JFlex...
Archivos generados

----- Menu de análisis léxico -----
1. Generar Lexer.java, Parser.java y sym.java
2. Realizar Análisis Léxico
0. Salir
Seleccione una opción: 

```

Los errores que marca Parser.java es normal, aún no se desarrolla esa parte porque no es del análisis léxico, no afecta.

## 2. Análisis de un archivo sin errores

The terminal window shows the lexical analysis report for the file 'reporte\_prueba1.txt'. The report includes the following information:

LINEA	COLUMNA	IDENTIFICADOR (TIPO)	LEXEMA
1		REPORTE DE TOKENS: prueba1.txt	
2		=====	
3		IDENTIFICADOR (TIPO)	LEXEMA
4		-----	
5	1	2 (WORLD)	world
6	7	4 (INT)	int
7	11	51 (ASSIGN)	=
8	13	3 (LOCAL)	local
9	19	5 (FLOAT)	float
10	25	59 (ID)	x
11	27	51 (ASSIGN)	=
12	29	56 (FLOAT_LITERAL)	3.14
13	34	24 (ENDL)	endl
14	1	11 (GIFT)	gift
15	6	10 (COAL)	coal
16	11	9 (NAVIDAD)	navidad
17	19	29 (OPEN_PAREN)	(
18	21	30 (CLOSE_PAREN)	)
19	23	27 (OPEN_BLOCK)	i
20	5	22 (SHOW)	show
21	25	57 (STRING_LITERAL)	"Hola mundo 123"
22	30	50 (NOT)	Σ
23	32	24 (ENDL)	endl
24	1	28 (CLOSE_BLOCK)	!
25	3	2 (WORLD)	world
26		No se encontraron errores léxicos.	
27		FIN DEL ANALISIS	

### 3. Análisis de un archivo con errores léxicos

```

reporte_prueba2.txt ×
programa > archivos_salida > reporte_prueba2.txt
You, 4 hours ago | 2 authors (You and one other)
1 REPORTE DE TOKENS: prueba2.txt You, 8 hours ago • Feat: manejo de errores 1
2 =====
3 IDENTIFICADOR (TIPO) | LEXEMA | LINEA | COLUMNA
4 -----
5 2 (WORLD) | world | 1 | 1
6 4 (INT) | int | 1 | 7
7 51 (ASSIGN) | = | 1 | 11
8 3 (LOCAL) | local | 1 | 13
9 5 (FLOAT) | float | 1 | 19
10 59 (ID) | x | 1 | 25
11 51 (ASSIGN) | = | 1 | 27
12 3.14 (FLOAT_LITERAL) | 3.14 | 1 | 29
13 endl (ENDL) | endl | 1 | 34
14 gift (GIFT) | gift | 3 | 1
15 coal (COAL) | coal | 3 | 6
16 # (ERROR) | # | 3 | 11
17 show (SHOW) | show | 4 | 5
18 "Hola mundo 123" (STRING_LITERAL) | "Hola mundo 123" | 4 | 25
19 _ (NOT) | _ | 4 | 30
20 endl (ENDL) | endl | 4 | 32
21 # (ERROR) | # | 5 | 1
22 ! (CLOSE_BLOCK) | ! | 6 | 1
23 world (WORLD) | world | 6 | 3
24 -----
25 ERRORES ENCONTRADOS:
26 - Carácter ilegal <#> en línea 3
27 - Carácter ilegal <#> en línea 5
28 -----
29 FIN DEL ANALISIS
30

```

### 4. Análisis usando todos los tokens del lenguaje

	IDENTIFICADOR (TIPO)	LEXEMA	LINEA	COLUMNA
1		==	8	/
2	45 (NEQ)	=	8	10
3	46 (LT)	<	8	13
4	47 (GT)	>	8	15
5	48 (AND)	@	8	17
6	49 (OR)	~	8	19
7	50 (NOT)	_	8	21
8	51 (ASSIGN)	=	8	23
9	52 (ARROW)	->	8	25
10	53 (COMMA)	,	8	28
11	54 (SEMICOLON)	;	8	30
12	55 (INT_LITERAL)	12345	12	1
13	56 (FLOAT_LITERAL)	3.14	12	7
14	57 (STRING_LITERAL)	"Hola"	12	17
15	58 (CHAR_LITERAL)	A	12	19
16	59 (ID)	una_variable_123	12	23
17	60 (NAVIDAD)	navidad	13	1
18	61 (OPEN_BLOCK)	{	13	9
19	62 (DECIDE)	decide	13	11
20	63 (OF)	of	13	18
21	64 (OPEN_PAREN)	(	13	21
22	65 (BOOL)	bool	13	23
23	66 (OPEN_BLOCK)	{	13	28
24	67 (DECIDE)	decide	13	31
25	68 (OF)	of	13	38
26	69 (OPEN_PAREN)	(	13	41
27	70 (TRUE)	true	13	42
28	71 (ID)	x	13	48
29	72 (ASSIGN)	=	13	50
30	73 (TRUE)	true	13	52
31	74 (CLOSE_PAREN)	)	13	57
32	75 (ARROW)	->	13	59
33	76 (SHOW)	show	13	62
34	77 (STRING_LITERAL)	"ok"	13	58
35	78 (ENDL)	endl	13	52
36	79 (CLOSE_BLOCK)	!	13	57
37			84	
38		No se encontraron errores léxicos.	81	
39			82	
40		FIN DEL ANALISIS	83	

## 5. Análisis con todos los tokens pero incluyendo errores

IDENTIFICADOR (TIPO)	LEXEMA	LINEA	COLUMNA
2 (WORLD)	world	2	1
3 (LOCAL)	local	2	7
4 (INT)	int	2	13
5 (FLOAT)	float	2	17
6 (BOOL)	bool	2	23
60 (ERROR)	#	2	28
9 (NAVIDAD)	navidad	3	1
12 (COAL)	coal	3	9
13 (GIFT)	gift	3	14
12 (DECIDE)	decide	3	19
13 (OF)	of	3	26
14 (ELSE)	else	3	29
15 (END)	end	3	34
16 (LOOP)	loop	3	38
17 (EXIT)	exit	3	43
20 (WHEN)	when	3	48
19 (FOR)	for	4	1
22 (RETURN)	return	5	1
21 (BREAK)	break	5	8
22 (SHOW)	show	5	14
23 (GET)	get	5	19
24 (ENDL)	endl	5	23
60 (ERROR)	o	5	28
26 (FALSE)	false	6	1
60 (ERROR)		7	1
30 42 (LTEQ)	<=	8	1
31 43 (GTEQ)	>=	8	4
32 44 (EQEQ)	==	8	7
33 45 (NEQ)	Σ=	8	10
34 46 (LT)	<	8	13
35 47 (GT)	>	8	15
36 48 (AND)	@	8	17
37 49 (OR)	~	8	19
38 50 (NOT)	Σ	8	21
39 51 (ASSIGN)	=	8	23
40 52 (ARROW)	->	8	25
41 53 (COMMA)	,	8	28
42 54 (SEMICOLON)	:	8	30
43 55 (INT_LITERAL)	12345	9	1
44 56 (FLOAT_LITERAL)	3.14	9	7
45 57 (STRING_LITERAL)	"Hola"	9	17
46 58 (CHAR_LITERAL)	A	9	19
47 59 (ID)	perro_123	9	23
48 29 (OPEN_PAREN)	(	9	33
49 29 (OPEN_PAREN)	{	9	34
50 60 (ERROR)	Comentario multilinea no cerrado   11	11	23
51			
52	ERRORES ENCONTRADOS:		
53	- Carácter ilegal #> en línea 2		
54	- Carácter ilegal <> en línea 5		
55	- Carácter ilegal <> en línea 7		
56	- Carácter ilegal <Comentario multilinea no cerrado> en línea 11		
57			
58	FIN DEL ANALISIS		
59			

## Descripción del problema

El objetivo de este primer proyecto es desarrollar la fase de análisis léxico para un lenguaje de programación imperativo diseñado para configuración de chips en sistemas empotrados.

El lenguaje funcionará con ciertos símbolos no convencionales, como (¡ y !) para delimitar bloques, (¿ y ?) como paréntesis, también Σ, @ y ~ como operadores NOT, AND y OR, (ε y ə) como comentarios multilínea, también deben haber operadores de división entera, potencia y diferente.

El analizador léxico debe ser capaz de leer archivos fuente escritos en este lenguaje nuevo que se está desarrollando, debe identificar y clasificar cada elemento del código en tokens, reportar errores léxicos sin que el análisis se detenga y generar reportes detallados con información de los tokens y errores.