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

Contenido

Repositorio de github

Enlace: https://github.com/JairoGH16/proyecto-compiladores-e-interpretes

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

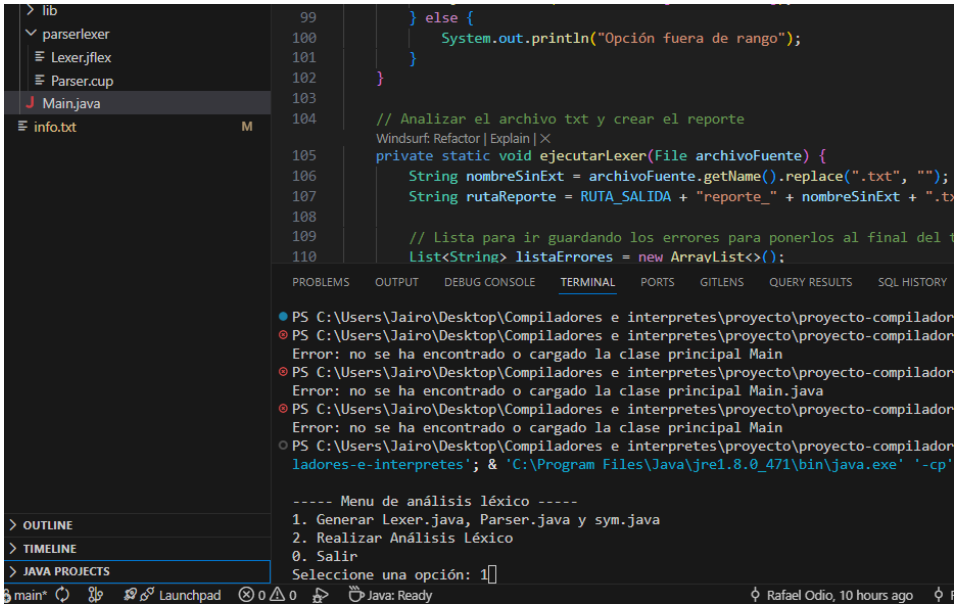
El reporte incluye una tabla con cada token encontrado (identificador, lexema, línea, columna) y una lista de errores léxicos si los hubiera.

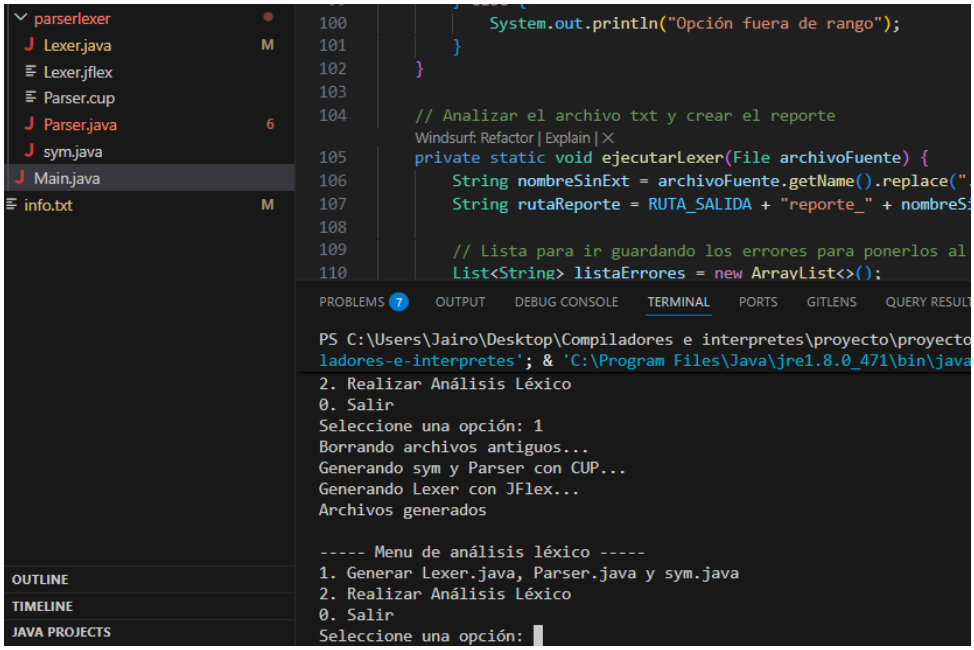
Se pueden ver más pruebas del uso en el siguiente apartado.

Pruebas de funcionalidad

Se pueden apreciar completamente las pruebas en el video: https://www.youtube.com/watch?v=kATyjgQVy2M

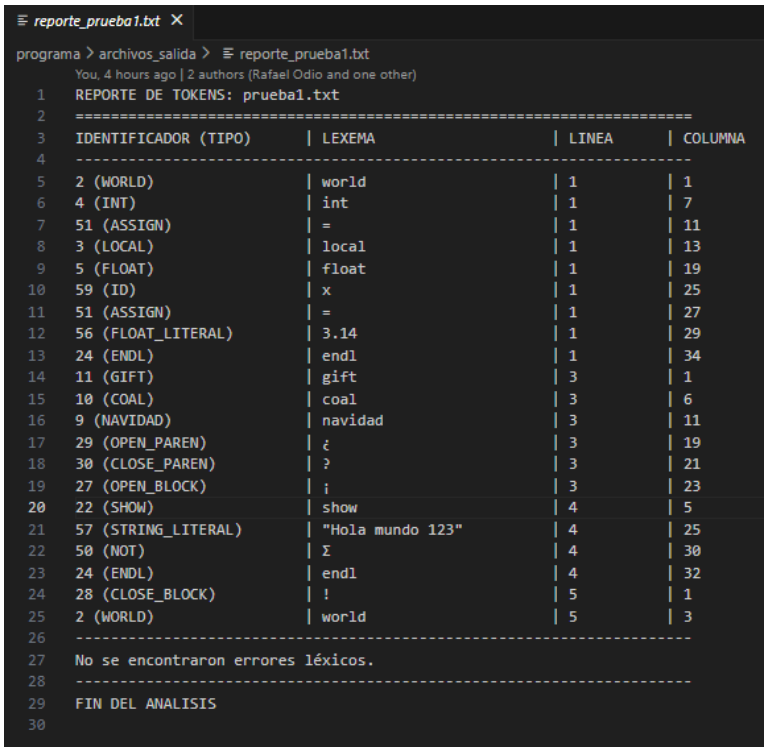
1. Generación de archivos .java con la opción 1





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

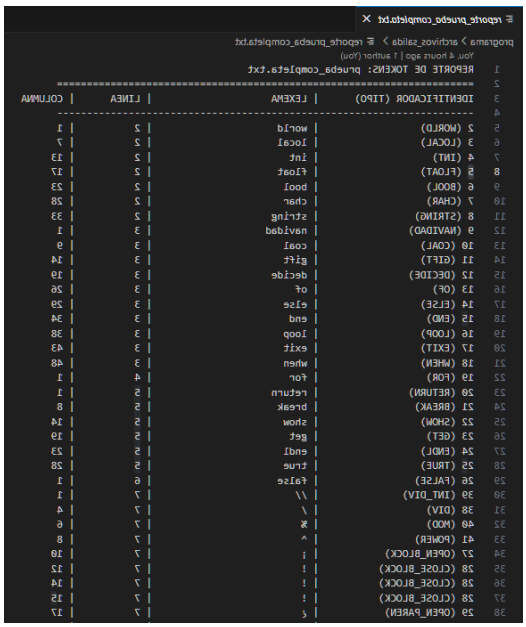


3. Análisis de un archivo con errores léxicos

reporte_prueba2.txt

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

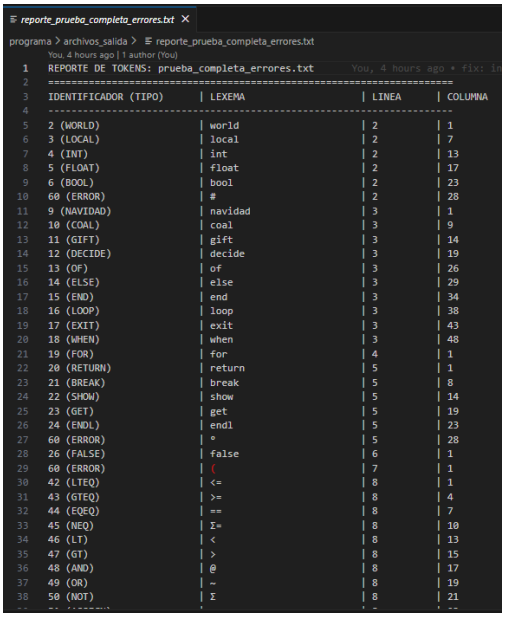
4. Análisis usando todos los tokens del lenguaje



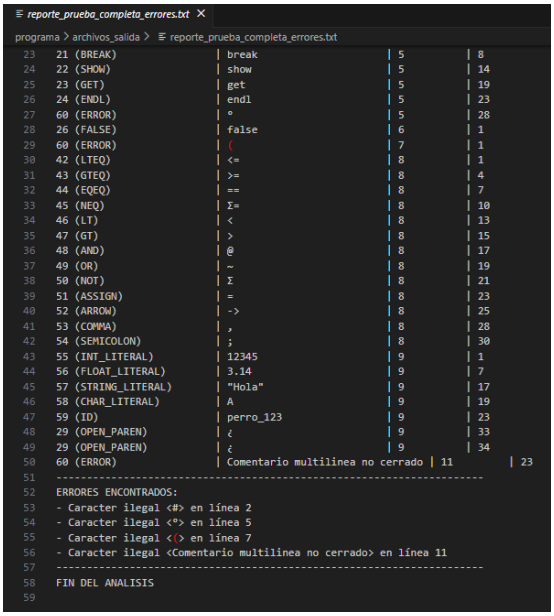
reporte_prueba_completa.txt

```
programa > archivos_salida > reporte_prueba_completa.txt
44 (EQEQ)                 | ==                        | 8          | 7
45 (NEQ)                  | Σ=                        | 8          | 10
46 (LT)                   | <                         | 8          | 13
47 (GT)                   | >                         | 8          | 15
48 (AND)                  | @                         | 8          | 17
49 (OR)                   | ~                         | 8          | 19
50 (NOT)                  | Σ                         | 8          | 21
51 (ASSIGN)               | =                         | 8          | 23
52 (ARROW)                | ->                        | 8          | 25
53 (COMMA)                | ,                         | 8          | 28
54 (SEMICOLON)            | ;                         | 8          | 30
55 (INT_LITERAL)          | 12345                     | 12         | 1
56 (FLOAT_LITERAL)        | 3.14                      | 12         | 7
57 (STRING_LITERAL)       | "Hola"                    | 12         | 17
58 (CHAR_LITERAL)         | A                         | 12         | 19
59 (ID)                   | una_variable_123          | 12         | 23
9 (NAVIDAD)               | navidad                   | 13         | 1
27 (OPEN_BLOCK)           | ¡                         | 13         | 9
12 (DECIDE)               | decide                    | 13         | 11
13 (OF)                   | of                        | 13         | 18
29 (OPEN_PAREN)           | ¿                         | 13         | 21
6 (BOOL)                  | bool                      | 13         | 23
59 (ID)                   | x                         | 13         | 28
51 (ASSIGN)               | =                         | 13         | 30
25 (TRUE)                 | true                      | 13         | 32
30 (CLOSE_PAREN)          | ?                         | 13         | 37
52 (ARROW)                | ->                        | 13         | 39
22 (SHOW)                 | show                      | 13         | 42
57 (STRING_LITERAL)       | "ok"                      | 13         | 50
24 (ENDL)                 | endl                      | 13         | 52
28 (CLOSE_BLOCK)          | !                         | 13         | 57
----------------------------------------------------------------------------
No se encontraron errores léxicos.
----------------------------------------------------------------------------
FIN DEL ANALISIS
```

5. Análisis con todos los tokens pero incluyendo errores



```
reporte_prueba_completa_errores.txt
programa > archivos_salida > reporte_prueba_completa_errores.txt
You, 4 hours ago | 1 author (You)
REPORTE DE TOKENS: prueba_completa_errores.txt        You, 4 hours ago • fix: 10
================================================================
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
10 (COAL)                 | coal                    | 3          | 9
11 (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
18 (WHEN)                 | when                    | 3          | 48
19 (FOR)                  | for                     | 4          | 1
20 (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)                | °                       | 5          | 28
26 (FALSE)                | false                   | 6          | 1
60 (ERROR)                | ¿                       | 7          | 1
42 (LTEQ)                 | <=                      | 8          | 1
43 (GTEQ)                 | >=                      | 8          | 4
44 (EQEQ)                 | ==                      | 8          | 7
45 (NEQ)                  | Σ=                      | 8          | 10
46 (LT)                   | <                       | 8          | 13
47 (GT)                   | >                       | 8          | 15
48 (AND)                  | @                       | 8          | 17
49 (OR)                   | ~                       | 8          | 19
50 (NOT)                  | Σ                       | 8          | 21
```



```
reporte_prueba_completa_errores.txt
programa > archivos_salida > reporte_prueba_completa_errores.txt
21 (BREAK)                | break                   | 5          | 8
22 (SHOW)                 | show                    | 5          | 14
23 (GET)                  | get                     | 5          | 19
24 (ENDL)                 | endl                    | 5          | 23
60 (ERROR)                | °                       | 5          | 28
26 (FALSE)                | false                   | 6          | 1
60 (ERROR)                | ¿                       | 7          | 1
42 (LTEQ)                 | <=                      | 8          | 1
43 (GTEQ)                 | >=                      | 8          | 4
44 (EQEQ)                 | ==                      | 8          | 7
45 (NEQ)                  | Σ=                      | 8          | 10
46 (LT)                   | <                       | 8          | 13
47 (GT)                   | >                       | 8          | 15
48 (AND)                  | @                       | 8          | 17
49 (OR)                   | ~                       | 8          | 19
50 (NOT)                  | Σ                       | 8          | 21
51 (ASSIGN)               | =                       | 8          | 23
52 (ARROW)                | ->                      | 8          | 25
53 (COMMA)                | ,                       | 8          | 28
54 (SEMICOLON)            | ;                       | 8          | 30
55 (INT_LITERAL)          | 12345                   | 9          | 1
56 (FLOAT_LITERAL)        | 3.14                    | 9          | 7
57 (STRING_LITERAL)       | "Hola"                  | 9          | 17
58 (CHAR_LITERAL)         | A                       | 9          | 19
59 (ID)                   | perro_123               | 9          | 23
29 (OPEN_PAREN)           | ¿                       | 9          | 33
29 (OPEN_PAREN)           | ¿                       | 9          | 34
60 (ERROR)                | Comentario multilinea no cerrado | 11         | 23
----------------------------------------------------------------------
ERRORES ENCONTRADOS:
- Caracter ilegal <#> en línea 2
- Caracter ilegal <°> en línea 5
- Caracter ilegal <¿> en línea 7
- Caracter ilegal <Comentario multilinea no cerrado> en línea 11
----------------------------------------------------------------------
FIN DEL ANALISIS
```

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, (є y э) 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.

Diseño del programa

Decisiones de diseño

El programa sigue una arquitectura modular separada en tres componentes principales, los cuales son Main.java, Lexer.jflex y Parser.cup.

Se utiliza UTF-8 para aceptar los símbolos especiales “є, э, Σ, ¡, ¿”, al inicio se presentaban problemas porque no detectaba esos caracteres y los tomaba como

errores por ser elementos desconocidos para el lenguaje, por lo que la solución fue forzar UTF-8 al ejecutar JFlex.

Para manejar errores se implementó un estado llamado “PANIC_MODE”, al encontrar un carácter ilegal, se entra en este modo, donde ignora el resto la línea actual y pasa a la siguiente, en la cual vuelve al estado normal, esto evita que el análisis se caiga por errores.

En lugar de cargar Lexer.java directamente, se usa reflexión, que permite cargarlo en tiempo de ejecución, invocar métodos que no tengan referencias directas en tiempo de compilación, extraer los nombres de los tokens de sym.java y regenerar el Lexer.java sin recompilar todo el proyecto desde cero.

Para los reportes se usó un formato de tablas, que facilita encontrar tokens específicos y leer su información.

Para el token de “diferente”, que en la mayoría de lenguajes de programación se representa su función con 

, se decidió usar Σ=, esto debido a que el Σ es el NOT en este lenguaje, así que se decidió que lo mejor era cambiar el NOT del diferente por un Σ también.

Algoritmos usados

1. Construcción de StringBuffer para cadenas: cuando el lexer encuentra una comilla doble, inicializa StringBuffer vacío, agrega la comilla inicial, cambia el estado a STRING, a partir de ahí si encuentra un \ lo toma como secuencia de escoma, si encuentra un “ agrega la comilla final y retornael token, cualquier otro elemento se agrega al buffer y al terminar se retorna el STRING_LITERAL que posee todo el contenido.
2. Recuperación de errores: cuando el lexer se encuentra un carácter ilegal, imprime el error, entra en el modo pánico (PANIC_MODE) que ya fue mencionado anteriormente, ignora todo hasta el salto de línea (\n) y al llegar a este vuelve a estado YYINITIAL y continua el análisis normalmente.
3. Búsqueda de nombres de tokens: usa reflexión, carga el sym.java, obtiene todos los campos, compara su valor con el id buscado y retorna su nombre al encontrarlo, si no lo encontrara retornaría un “UNKNOWN”
4. Análisis de un archivo txt: abre el archivo con UTF-8, se escribe un encabezado para la tabla del reporte, se entra en un bucle hasta que no se llegue al EOF (fin del archivo) y mientras que está en ese bucle obtiene el siguiente token, si es un error lo agrega a la lista de errores, extrae el lexema, el tipo de token y su línea y columna, lo escribe en una fila de la tabla y repite con el siguiente token… Al final se escriben los errores en la sección de errores si los hubiera y finalmente se cierran los archivos txt.

5. Menú: consiste en switch y en if y else básicos que leen un número y siguen el recorrido indicado.

Resumen y análisis del proceso que hace jflex para generar el analizador léxico

Lo primero que JFlex hace es una lectura de las especificaciones que le damos en el archivo lexer.jflex y lo divide en código de usuario, opciones y macros y reglas léxicas, dichas reglas funcionan de una forma patrón y acción: “patron” { acción },

Ejemplo: "world" { return symbol(sym.WORLD); }

Significa que cuando se encuentra el patrón indicado, realiza la acción que está entre las llaves { }.

Luego de esto se convierten las expresiones regulares, JFlex usa el algoritmo de Thompson para construir el Autómata Finito No Determinista (AFN), por ejemplo para el patrón {identifier} = [a-zA-Z_][a-zA-Z0-9_]* se crea un estado inicial, se crean las transiciones y ciclos entre estados intermedio y se marca cuál será el estado final como aceptador.

A partir de esto se convierte ese AFN en un AFD (Autómata Finito Determinista), primero se tiene el conjunto de estados alcanzables del AFN, se calcula a qué conjunto de estados se puede llegar y si es nuevo se agrega como estado del AFD y esto se va a repetir hasta que no haya conjuntos nuevos. Luego este estado se minimiza con un algoritmo llamado minimización de Hopcroft que básicamente lo que hace es agrupar estados equivalentes que se comportan de manera idéntica separando aceptadores de no aceptadores hasta reducir el grafo todo lo que sea posible. Así quedan muchos menos estados.

Al final a partir de esto se genera el código Lexer.java, que es el que permite hacer los análisis léxicos, tiene una tabla de transiciones y otra de acciones y lo que hace es ir leyendo los caracteres, consultar la tabla de transiciones, ver si es estado aceptador para ejecutar una acción y si hay transiciones inválidas reporta errores. También hay un buffer que maneja la lectura.

Librerías usadas

1. Librerías Externas (Archivos .jar)

JFlex 1.9.1 (jflex-full-1.9.1.jar): generar el analizador léxico a partir de Lexer.jflex.

Java CUP 11b - Binario (java-cup-11b.jar): procesa Parser.cup para generar sym.java con los IDs de tokens, también genera Parser.java (no usado en esta fase).

Java CUP 11b - Runtime (java-cup-11b-runtime.jar)**:** librería necesaria en tiempo de ejecución.

2. Paquetes del JDK (Biblioteca Estándar de Java)

java.io: gestión de rutas y archivos, lectura de archivos y generación de reportes con UTF-8.

java.util: para el menú interactivo por consola y poder usar listas.

java.lang.reflect: para cargar dinámicamente las clases generadas y extraer nombres de tokens de sym.java

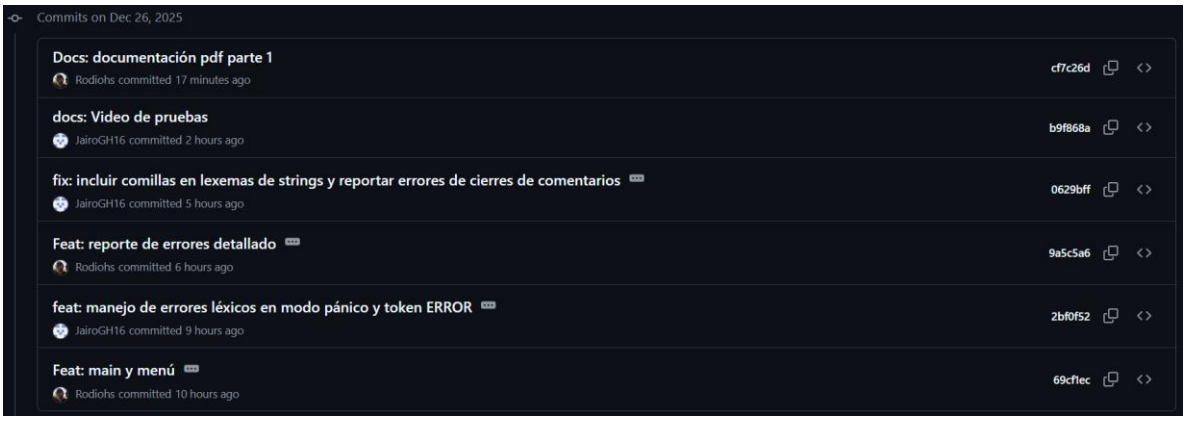
Análisis de resultados

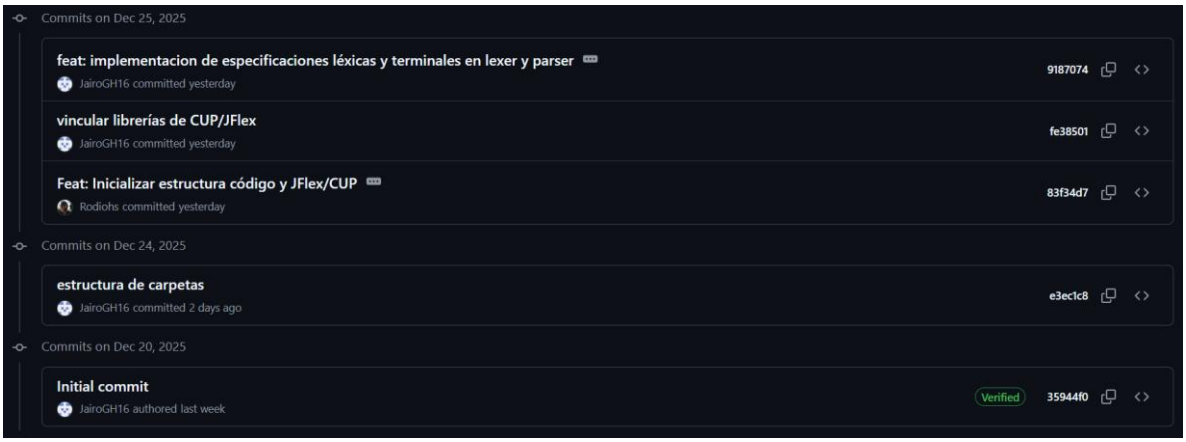
Se cumplieron todos los objetivos y requerimientos del proyecto.

Objetivos alcanzados

- **Construcción de un scanner funcional:** se logró desarrollar un analizador léxico completo utilizando JFlex, capaz de identificar todos los terminales especificados, incluyendo palabras reservadas, operadores, delimitadores y literales.
- **Gestión de caracteres especiales:** se implementó exitosamente el soporte para caracteres exigidos por el lenguaje, como los delimitadores de bloque (¡, !), paréntesis (¿, ?), operadores lógicos (Σ, @, ~) y símbolos de comentarios (ϵ, ə), usando UTF-8.
- **Recuperación de errores en modo pánico:** se logró no detener el análisis ante errores. El sistema identifica el error, lo reporta y sigue el análisis saltando a la siguiente línea del archivo fuente.
- **Generación de los reportes:** el programa produce archivos de salida estructurados que vinculan cada lexema con su identificador numérico, línea y columna, además de mostrar los errores si los hay.

Bitácora





Fecha	Actividad / Commit	Encargado
20 de diciembre de 2025	**Initial commit:** Creación del repositorio inicial.	Jairo
24 de diciembre de 2025	**Estructura de carpetas:** se estructuran las carpetas a cómo se van a usar en el proyecto.	Jairo
25 de diciembre de 2025	**Feat: Inicializar estructura código y JFlex/CUP**: se incluyen las bases para Lexer.jflex y Parser.cup, se agregan las librerías externas (JFlex y CUP) en el directorio lib	Rafael

25 de diciembre de 2025	**Vincular librerías de CUP/JFlex:** se conectan al proyecto las librerías externas .jar.	Jairo
25 de diciembre de 2025	**Feat: implementación de especificaciones léxicas y terminales en lexer y parser:** se definieron todos los tokens y operadores, se configuraron los terminales en Parser.cup para la generación de sym.java, se generaron las clases Lexer.java y sym.java, se creó un archivo de prueba inicial (fuente.txt) para usar después.	Jairo
26 de diciembre de 2025	**Feat: Main y menú:** Se implementó un main para generar los archivos lexer, sym y parser y hacer el análisis y reportes, se hizo un menú básico con soporte para excepciones básicas como error en el análisis o carpeta de archivos vacía.	Rafael
26 de diciembre de 2025	**Feat: manejo de errores léxicos en modo pánico y token ERROR:** Se agregó el terminal ERROR en Parser.cup para identificar lexemas inválidos, se implementó la recuperación en modo pánico en Lexer.jflex para ignorar el resto de la línea y saltar a la que sigue.	Jairo
26 de diciembre de 2025	**Feat: Reporte de errores detallado:** Se agregó al main agregar los errores a una lista conforme van saliendo y añadirlos al final de cada reporte.	Rafael
26 de diciembre de 2025	**fix: incluir comillas en lexemas de strings y reportar errores de cierres de comentarios:** Se corrigió el estado STRING para incluir las comillas como parte del lexema, se implementó el estado MULTICOMMENT para validar cierres para comentarios multilinea, reportando un error específico si se alcanza el EOF sin cerrar el bloque.	Jairo

26 de diciembre de 2025	**Docs: Video de pruebas:** video donde se prueba el analizador.	Rafael
26 de diciembre de 2025	**Docs: Documentación Parte 1:** se adjunta la primera mitad de la documentación.	Rafael
26 de diciembre de 2025 (commit actual)	**Docs: Documentación Parte 2:** se adjunta la segunda mitad de la documentación externa y se completa la documentación interna.	Jairo

Bibliografía

JFlex Manual (Version 1.9.1): https://jflex.de/manual.html

CUP Manual. Hudson, Scott E: http://www2.cs.tum.edu/projects/cup/

Oracle Java Documentation. Oracle Corporation: https://docs.oracle.com/en/java/