Segundo proyecto – Compiladores e intérpretes

Analizador Sintáctico

Profesor:

Allan Rodríguez Dávila

Estudiantes:

Jairo González Hidalgo - 2024178535
Rafael Odio Mendoza - 2024083499

17 de enero – Verano 2025-2026

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.

La opción 3 se utiliza para realizar el análisis sintáctico, al usar esta opción, se nos mostraran los archivos .txt disponibles en la carpeta archivos_prueba/, luego,

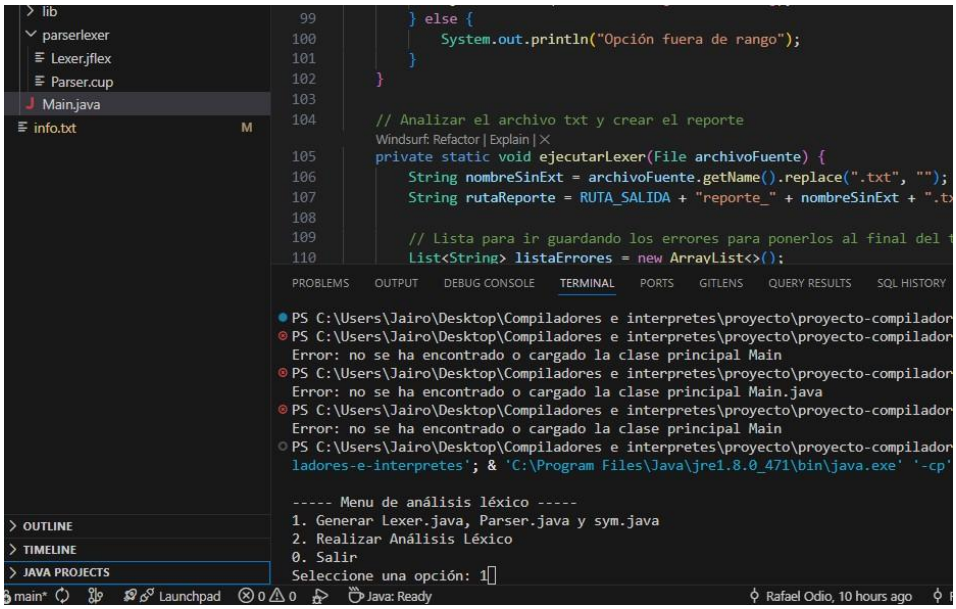
al seleccionar el archivo, se pasara automáticamente a realizar el análisis y también mostrara los resultados, estos siendo los errores que se encontraron, el AST concreto y la tabla de símbolos.

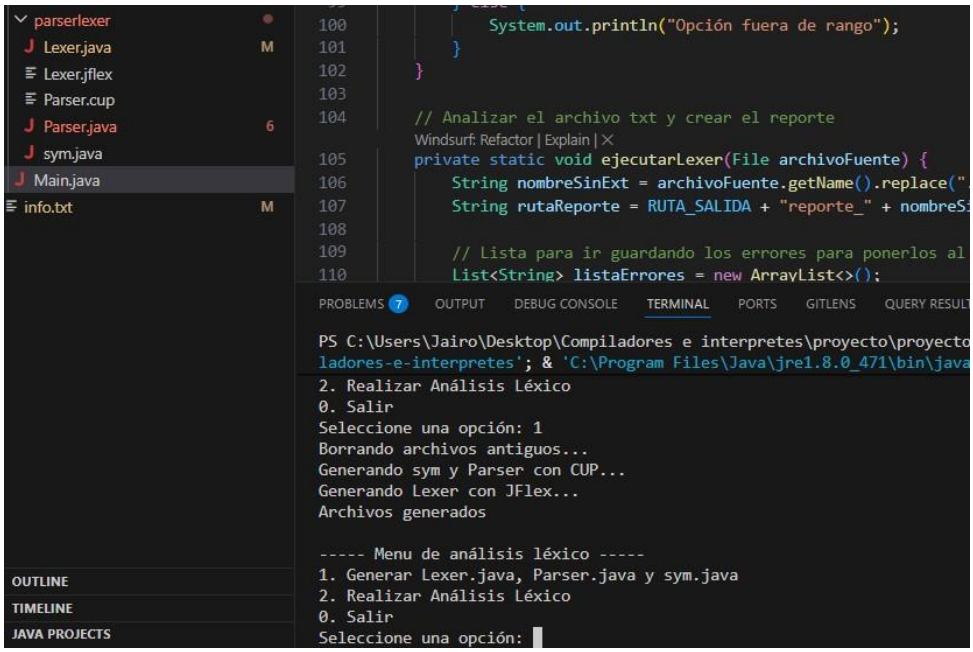
Por cada archivo analizado se genera un reporte en archivos_salida/ con el formato reporte_nombre.txt en el caso de usar la opción 2, y con el formato árbol_reporte_bombre.txt en el caso de la opción 3.

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

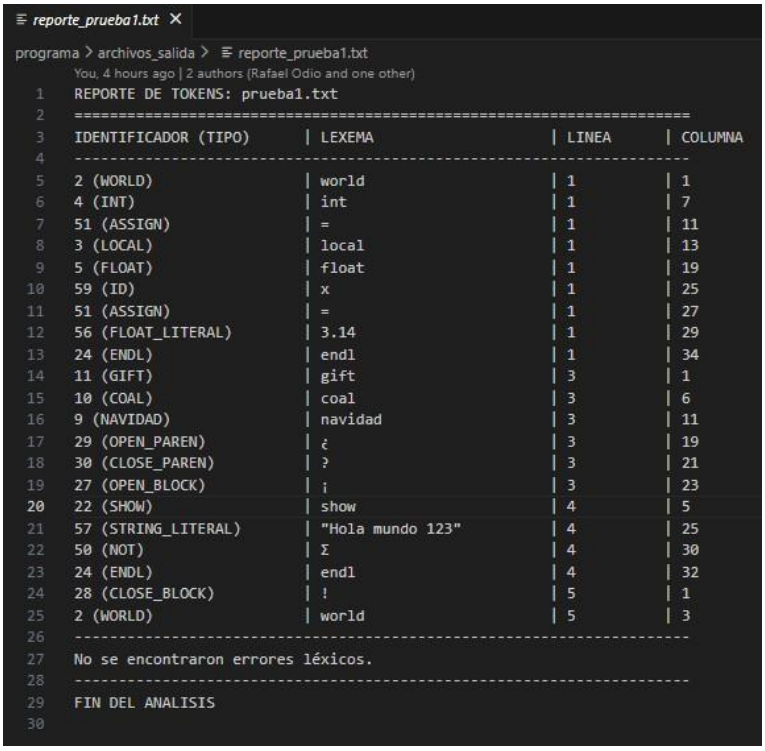
Pruebas de funcionalidad

1. Generación de archivos .java con la opción 1

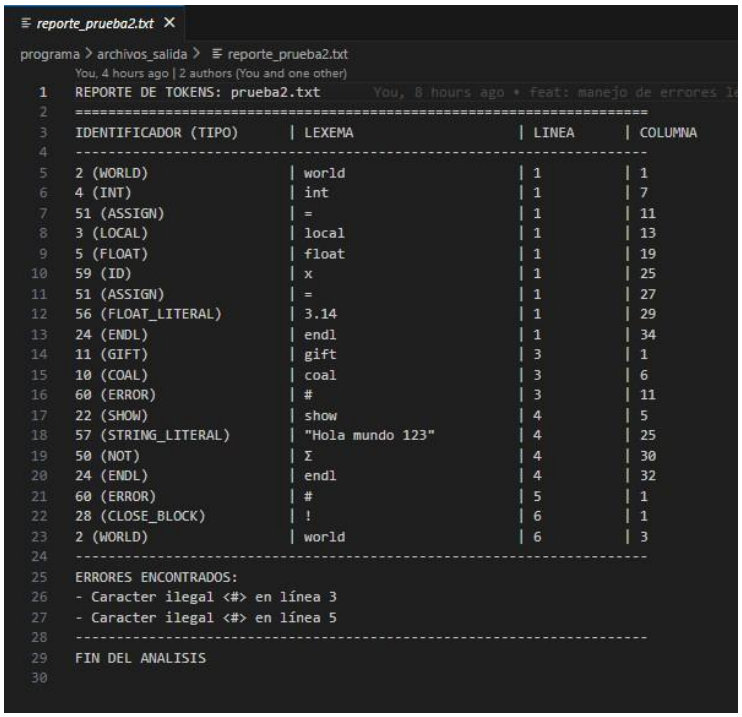




2. Análisis de un archivo sin errores

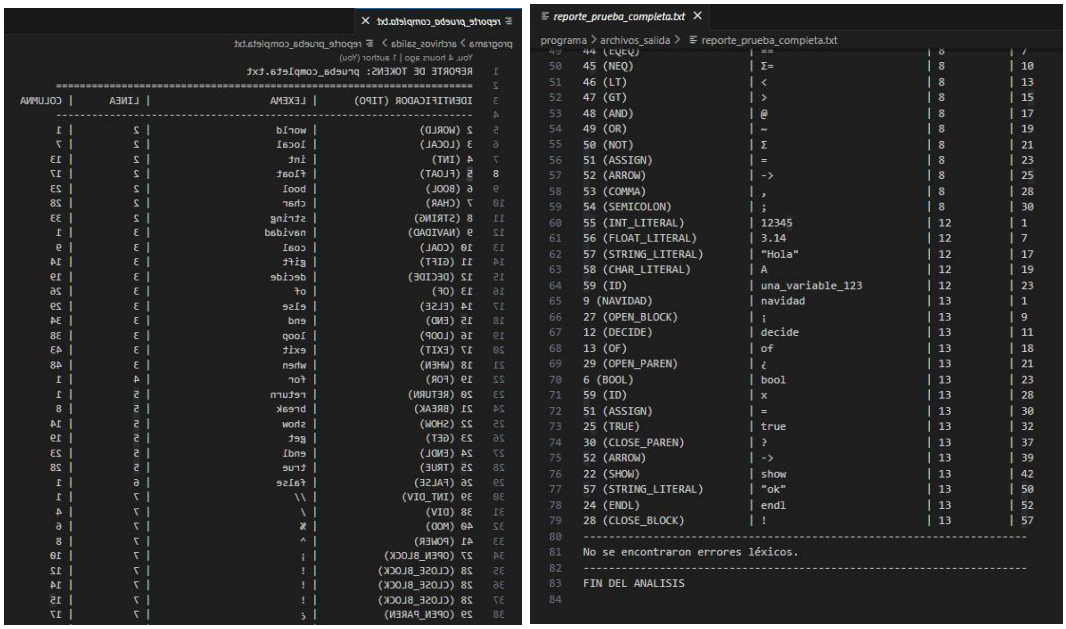


3. Análisis de un archivo con errores léxicos



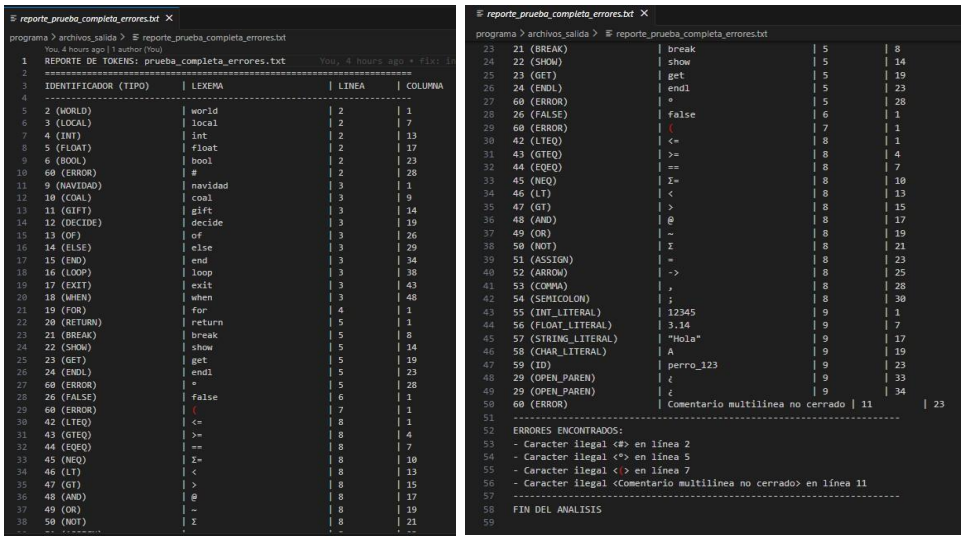
```
reporte_prueba2.txt
programa > archivos_salida > reporte_prueba2.txt
You, 4 hours ago | 2 authors (You and one other)
REPORTE DE TOKENS: prueba2.txt      You, 8 hours ago • feat: manejo de errores l
=================================================================
IDENTIFICADOR (TIPO)      | LEXEMA                    | 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
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
```

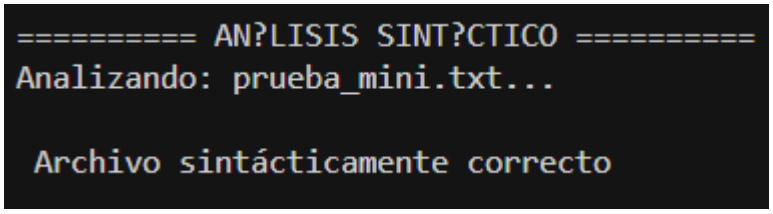
5. Análisis con todos los tokens pero incluyendo errores



```
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: i
================================================================================
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)                  | ≠=                    | 8          | 10
46 (LT)                   | <                     | 8          | 13
47 (GT)                   | >                     | 8          | 15
48 (AND)                  | @                     | 8          | 17
49 (OR)                   | ~                     | 8          | 19
50 (NOT)                  | ∑                     | 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)                  | ≠=                    | 8          | 10
46 (LT)                   | <                     | 8          | 13
47 (GT)                   | >                     | 8          | 15
48 (AND)                  | @                     | 8          | 17
49 (OR)                   | ~                     | 8          | 19
50 (NOT)                  | ∑                     | 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 multilínea no cerrado | 11         | 23
--------------------------------------------------------------------------------
ERRORES ENCONTRADOS:
- Caracter ilegal <#> en línea 2
- Caracter ilegal <°> en línea 5
- Caracter ilegal <¿> en línea 7
- Caracter ilegal <Comentario multilínea no cerrado> en línea 11
--------------------------------------------------------------------------------
FIN DEL ANALISIS
```

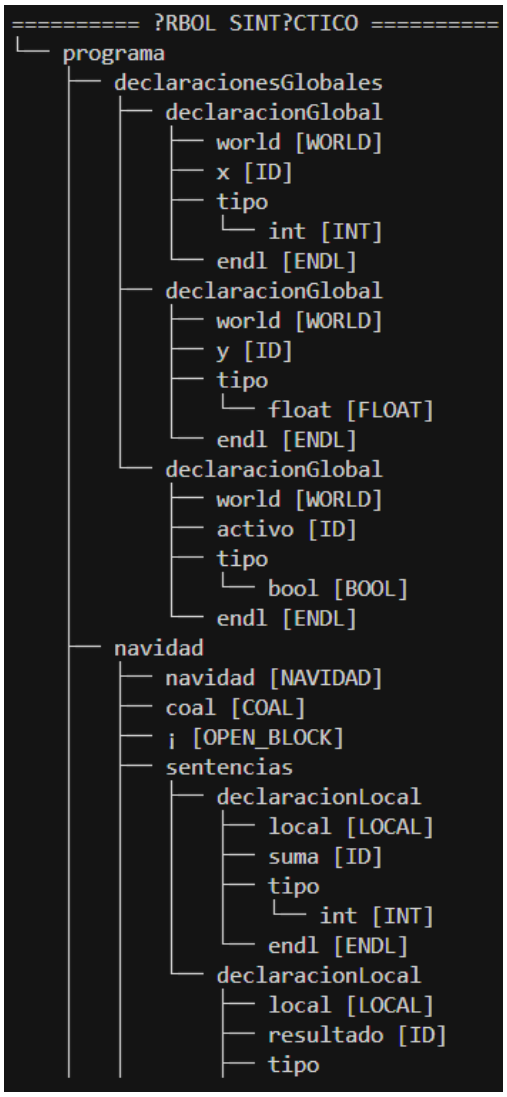
6. Iniciar analisis sintactico (archive sin errores)



```
========== AN?LISIS SINT?CTICO ==========
Analizando: prueba_mini.txt...

 Archivo sintácticamente correcto
```

7. AST Creado a partir del análisis



8. Tabla de Simbolos creada a partir del análisis

TABLA DE S?MBOLOS

ALCANCE: GLOBAL

NOMBRE	TIPO	L?NEA	CATEGOR?A
x	int	1	variable
y	float	2	variable
activo	bool	3	variable
sumar	int	10	funcion
multiplicar	float	15	funcion
factorial	int	19	funcion
multiplicarrrr	float	32	funcion

ALCANCE: NAVIDAD

NOMBRE	TIPO	L?NEA	CATEGOR?A
suma	int	6	variable
resultado	float	7	variable

ALCANCE: sumar

NOMBRE	TIPO	L?NEA	CATEGOR?A
a	int	10	parametro
b	int	10	parametro
temp	int	11	variable

9. Ejemplo de análisis de archivo con errores

```
========== AN?LISIS SINT?CTICO ==========
Analizando: prueba_modo_panico.txt...
Error sintáctico en línea 2, columna 9: Syntax error
instead expected token classes are [BOOL, CHAR, STRING]
Error sintáctico: Error en declaración global
Error sintáctico en línea 7, columna 13: Syntax error
instead expected token classes are [INT, FLOAT, BOOL, CHAR, STRING]
Error sintáctico: Error en declaración local
Error sintáctico en línea 11, columna 20: Syntax error
instead expected token classes are [CLOSE_PAREN, OPEN_BRACKET, COMMA]
Error sintáctico: Error en parametros de funcion 'sumar', procesando cuerpo
Error sintáctico en línea 16, columna 25: Syntax error
instead expected token classes are [INT, FLOAT, BOOL, CHAR, STRING]
Error sintáctico: Error en parametros de funcion 'restar', procesando cuerpo
Error sintáctico en línea 21, columna 21: Syntax error
instead expected token classes are [INT, FLOAT, BOOL, CHAR, STRING]
Error sintáctico: Error en declaración local

? Se encontraron 10 error(es) sintáctico(s)
```

Descripción del problema

Contexto

En el **Proyecto #1**, se implementó exitosamente la fase de análisis léxico, que estableció las bases del compilador. Este analizador léxico es capaz de reconocer tokens del lenguaje utilizando símbolos no convencionales como (¡ y !) para

delimitar bloques, (¿ y ?) como paréntesis, Σ, @ y ~ como operadores lógicos NOT, AND y OR, (є y ə) para comentarios multilínea, además de operadores especiales como división entera (//), potencia (^) y diferente (Σ=). El lexer desarrollado identifica y clasifica elementos del código fuente, reporta errores léxicos sin detener el análisis, y genera reportes detallados con información de tokens y errores.

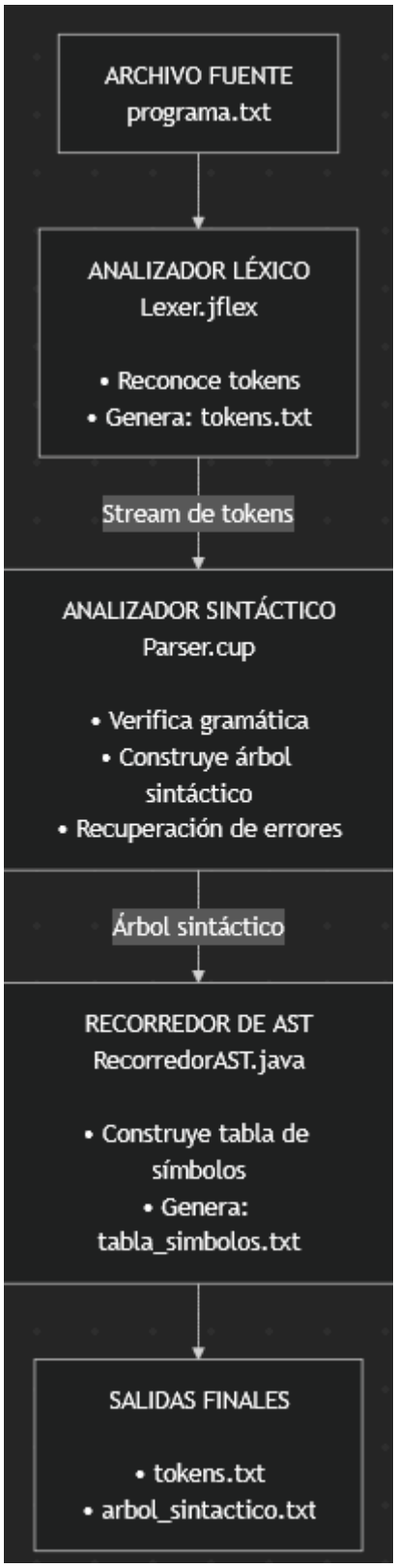
Objetivos del Proyecto

El **Proyecto #2** constituye la fase de **análisis sintáctico**, que representa el siguiente paso crítico en el desarrollo del compilador. Mientras el análisis léxico se enfoca en reconocer tokens individuales, el análisis sintáctico verifica que estos tokens estén organizados según las reglas gramaticales del lenguaje, construyendo una representación estructurada del programa.

Diseño del programa

Algunas de las decisiones que se tomaron en cuenta a la hora de desarrollar este proyecto fueron el hecho de que la construcción del AST concreto se da mientras CUP procesa las producciones, mediante acciones semánticas que crean y conectan nodos. De una forma similar, la tabla de símbolos se crea a partir del AST concreto, con la diferencia de que la tabla de símbolos espera a que este completo y saca la información desde este.
Tambien, cada fase se trabajo por separado para así poder probar su funcionalidad independientemente, para posteriormente juntarlo con las demás partes

Por otra parte, el compilador sigue una arquitectura en capas:


ARCHIVO FUENTE
programa.txt
ANALIZADOR LÉXICO
Lexer.jflex
• Reconoce tokens
• Genera: tokens.txt
Stream de tokens
ANALIZADOR SINTÁCTICO
Parser.cup
• Verifica gramática
• Construye árbol sintáctico
• Recuperación de errores
Árbol sintáctico
RECORREDOR DE AST
RecorredorAST.java
• Construye tabla de símbolos
• Genera: tabla_simbolos.txt
SALIDAS FINALES
• tokens.txt
• arbol_sintactico.txt

Librerias y Herramientas

El desarrollo del compilador se basó en tres componentes fundamentales que trabajaron de manera integrada para implementar las fases de análisis léxico y sintáctico.

JFlex generó el analizador léxico a partir de especificaciones en expresiones regulares. Procesó el archivo Lexer.jflex y produjo Lexer.java, implementando un autómata finito determinista que reconoce todos los tokens del lenguaje. La capacidad de manejar estados léxicos múltiples fue esencial para procesar strings con escapes, comentarios multilínea con símbolos especiales, y recuperación de errores mediante modo pánico.

CUP generó el analizador sintáctico. A partir de Parser.cup con la gramática y acciones semánticas, produjo Parser.java y sym.java. La herramienta manejó automáticamente precedencia de operadores y facilitó la recuperación de errores mediante el símbolo especial error.

Java Standard Library proporcionó las estructuras de datos necesarias sin dependencias externas. Se utilizaron clases de java.io para lectura y escritura de archivos (FileReader, PrintWriter), y de java.util para estructuras como ArrayList y HashMap en la implementación de la tabla de símbolos y el árbol sintáctico. Esta decisión mantuvo la simplicidad y portabilidad del proyecto.

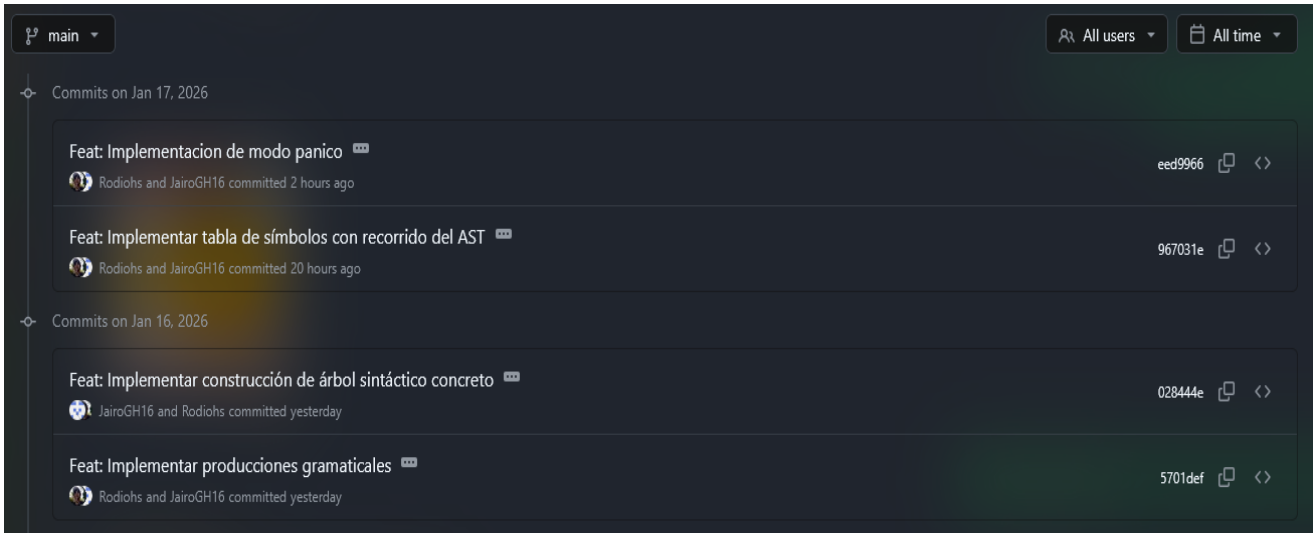
Analisis de Resultados

El proyecto cumplió exitosamente todos los objetivos establecidos. El analizador léxico reconoce correctamente todos los tokens del lenguaje, implementa restricciones adecuadas para literales numéricos (enteros sin ceros iniciales, flotantes con notación científica), maneja comentarios multilínea con caracteres especiales mediante estados léxicos, y detecta errores sin detener el análisis.

El analizador sintáctico verifica la gramática completa, construye el árbol sintáctico durante el parsing, maneja precedencia de operadores mediante estratificación, e implementa modo pánico con nueve reglas de error que sincronizan en puntos estratégicos (ENDL, CLOSE_BLOCK, etc.), permitiendo detectar múltiples errores en una sola pasada.

La tabla de símbolos se construye automáticamente recorriendo el árbol sintáctico, maneja correctamente la jerarquía de alcances (global, navidad, funciones) con búsqueda hacia arriba, y registra variables, funciones y parámetros con su tipo, categoría y línea. La estrategia agresiva de recuperación permite construir tablas más completas al procesar cuerpos de funciones incluso con errores en parámetros.

Bitácora



Algo importante de recalcar es que para este proyecto se trabajo en conjunto en todos los commits que se realizaron, de ahí el porqué se puede ver a ambos integrantes como autores en cada commit.