Segundo proyecto – Compiladores e intérpretes

Analizador Semántico y Generación de código MIPS

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__tabla_reporte_nombre.txt en el caso de la opción 3.

La opción 4 nos permite realizar un análisis semántico a un archivo, de la misma forma que con las anteriores opciones se nos mostrarán los archivos disponibles para analizar.

Al escoger uno de estos automáticamente se procederá a realizar el análisis y se nos mostraran los resultados en la consola.

En el caso de la opción 5, en esta se realizan los 3 análisis anteriores que vendrían siendo léxico, sintáctico y semántico. Al escoger un archivo se procederá automáticamente a realizar el analisis y en caso de pasar todos los analisis de manera exitosa se mostrara en consola que el archivo es valido para su traducción a TAC y posteriormente a MIPS.

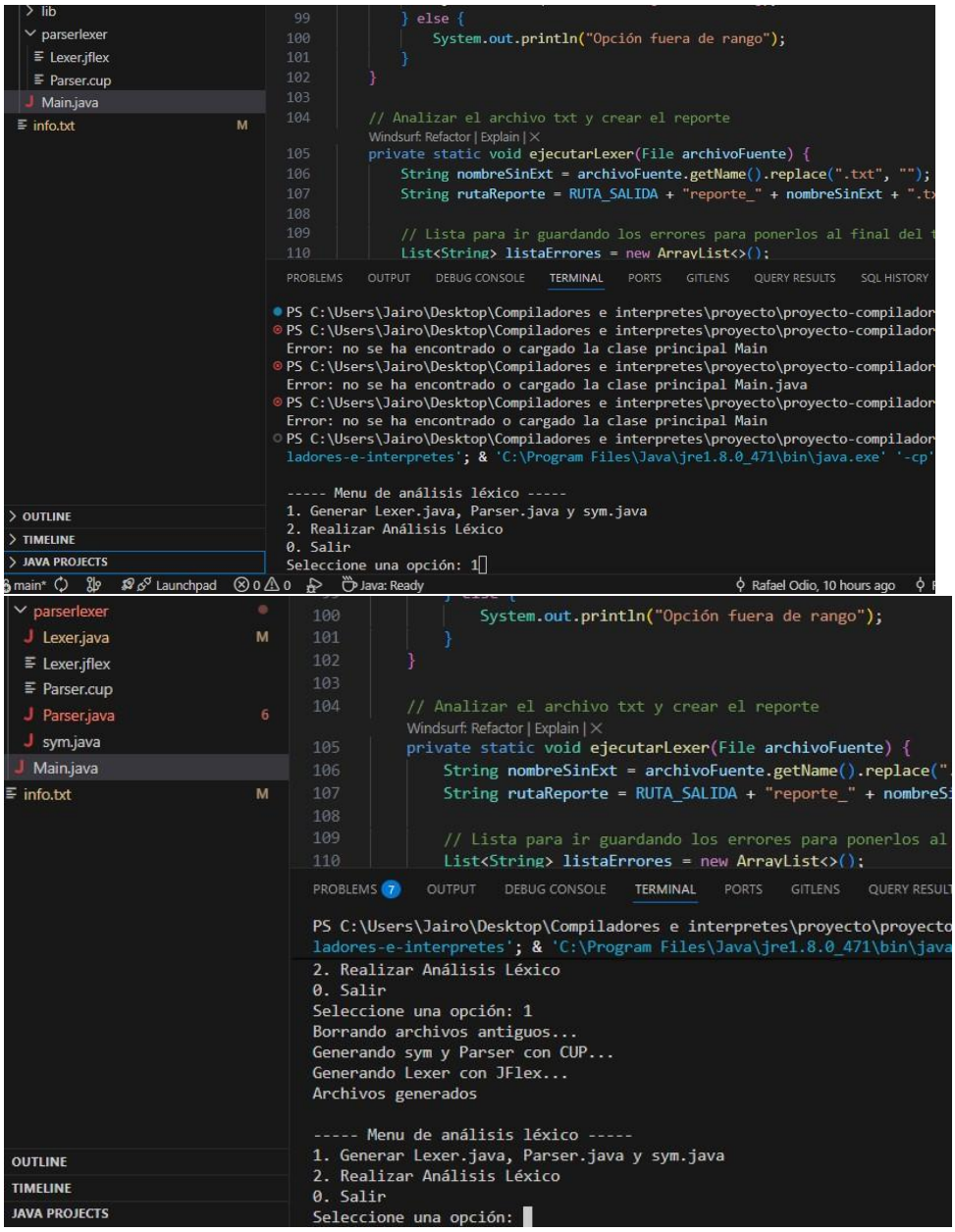
La opción 6 es la que nos permite realizar la traducción del código “normal” a un código intermedio de tres direcciones o TAC. Al escoger un archivo se realizara la traducción y se exportara el resultado a la carpeta de archivos_saida con el formato tac_nombrearchivo.txt

Por ultimo la opción 7 realiza la traducción de TAC a código MIPS utilizable en qtSpim, en este caso se crea un archivo con el formato mips_nombrearchivo.asm para que este se pueda utilizar.

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

reporte_prueba1.txt

programa > archivos_salida > reporte_prueba1.txt

You, 4 hours ago | 2 authors (Rafael Odio and one other)

```
REPORTE DE TOKENS: prueba1.txt
=====================================================================
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
9 (NAVIDAD)               | navidad               | 3          | 11
29 (OPEN_PAREN)           | ¿                     | 3          | 19
30 (CLOSE_PAREN)          | ?                     | 3          | 21
27 (OPEN_BLOCK)           | ¡                     | 3          | 23
22 (SHOW)                 | show                  | 4          | 5
57 (STRING_LITERAL)       | "Hola mundo 123"      | 4          | 25
50 (NOT)                  | Σ                     | 4          | 30
24 (ENDL)                 | endl                  | 4          | 32
28 (CLOSE_BLOCK)          | !                     | 5          | 1
2 (WORLD)                 | world                 | 5          | 3
---------------------------------------------------------------------
No se encontraron errores léxicos.
---------------------------------------------------------------------
FIN DEL ANALISIS
```

3. Análisis de un archivo con errores léxicos

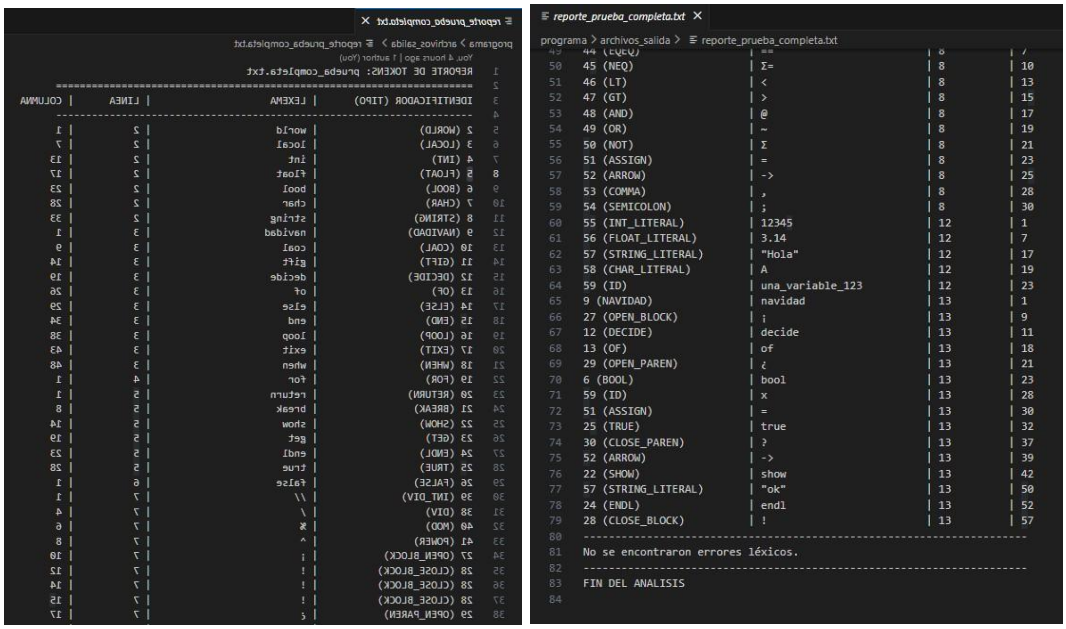
reporte_prueba2.txt

programa > archivos_salida > reporte_prueba2.txt

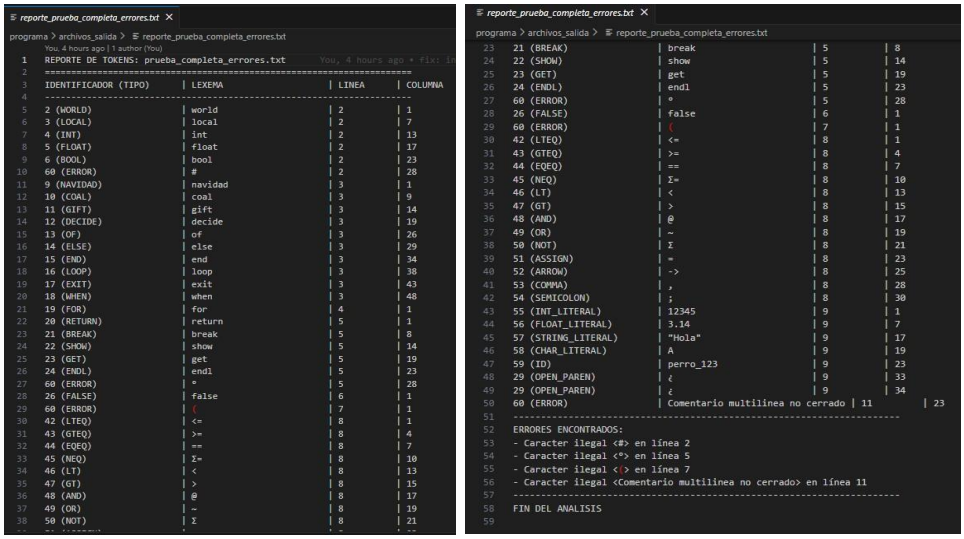
You, 4 hours ago | 2 authors (You and one other)

```
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



5. Análisis con todos los tokens pero incluyendo errores

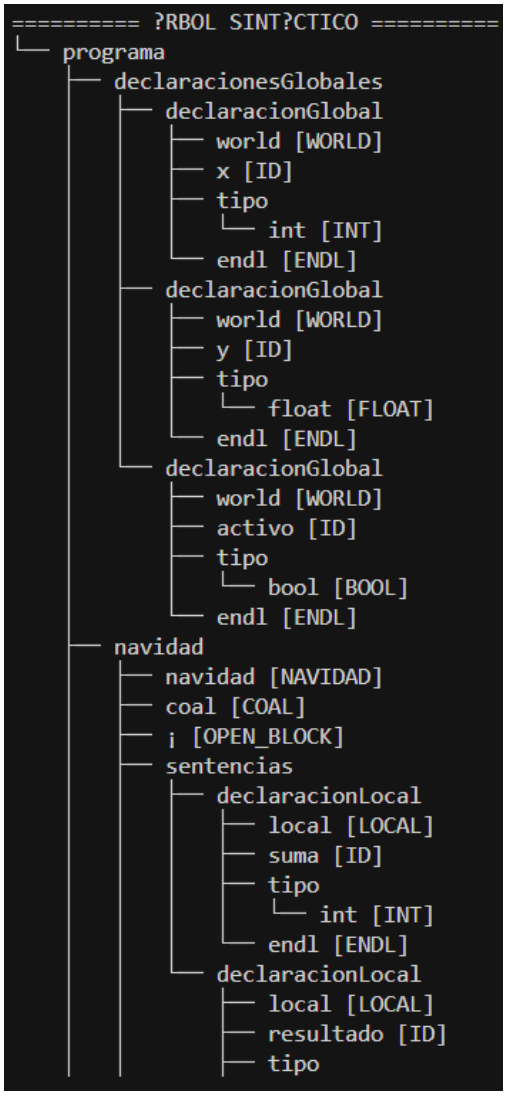


6. Iniciar analisis sintactico (archive sin errores)

```
========== 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)
```

10. Ejemplo de analisis semántico con errores

```
========== ANALISIS SEMANTICO ==========

ERRORES SEMANTICOS:
  - Línea 2: Variable global duplicada 'x'
  - Línea 8: Función duplicada 'test1'
  - Línea 9: Error de tipado fuerte - La función 'test1' declara un retorno 'int' pero intentó devolver 'float'. No se permite conversión implícita.
  - Línea 12: Parámetro duplicado 'a'
  - Línea 17: variable 'y' tipo 'int' pero asignado 'string'
  - Línea 18: variable 'z' tipo 'float' pero asignado 'int'
  - Línea 19: variable 'w' tipo 'bool' pero asignado 'int'
  - Línea 20: variable 'v' tipo 'string' pero asignado 'bool'
  - Línea 25: variable no declarada 'noExiste'
  - Línea 26: variable no declarada 'varInexistente'
  - Línea 31: variable no declarada 'variableInexistente'
  - Línea 37: asignación incompatible - variable 'num' tipo 'int' pero asignado 'string'
  - Línea 42: Error de análisis - función 'funcionQueNoExiste' no declarada.
  - Línea 51: Error de parámetros - la función 'suma' esperaba 2 argumentos pero recibió 1.
  - Línea 52: Error de parámetros - la función 'suma' esperaba 2 argumentos pero recibió 3.
  - Línea 61: Error de tipado fuerte - El argumento 1 de la función 'multiply' debe ser de tipo 'int' pero se recibió 'string'. No se permiten conversiones implícitas.
  - Línea 62: Error de tipado fuerte - El argumento 2 de la función 'multiply' debe ser de tipo 'int' pero se recibió 'bool'. No se permiten conversiones implícitas.
  - Línea 67: Error de tipado fuerte - La función 'retornoInt' declara un retorno 'int' pero intentó devolver 'string'. No se permite conversión implícita.
  - Línea 71: Error de tipado fuerte - La función 'retornoFloat' declara un retorno 'float' pero intentó devolver 'int'. No se permite conversión implícita.
  - Línea 75: Error de tipado fuerte - La función 'retornoBool' declara un retorno 'bool' pero intentó devolver 'int'. No se permite conversión implícita.
  - Línea 79: Error de tipado fuerte - Operación inválida entre 'string' y 'int'. Debes convertir uno de los valores.
  - Línea 85: Error de tipado fuerte - No se puede comparar 'int' con 'string'. Los tipos deben ser idénticos.
  - Línea 86: Error de tipado fuerte - No se puede comparar 'bool' con 'int'. Los tipos deben ser idénticos.
  - Línea 87: Error de tipado fuerte - No se puede comparar 'string' con 'int'. Los tipos deben ser idénticos.
  - Línea 92: operador lógico requiere 'bool' pero lado izquierdo es 'int'
  - Línea 93: operador lógico requiere 'bool' pero lado izquierdo es 'string'
  - Línea 94: operador lógico requiere 'bool' pero lado izquierdo es 'int'
  - Línea 94: operador lógico requiere 'bool' pero lado derecho es 'int'
  - Línea 99: operador NOT requiere operando 'bool' pero es 'int'
  - Línea 100: operador NOT requiere operando 'bool' pero es 'string'
  - Línea 106: condición debe ser tipo 'bool' pero es 'int'
  - Línea 117: exit when debe tener condición 'bool' pero es 'int'
  - Línea 123: array no declarado 'arrayInexistente'
  - Línea 129: índice de array debe ser 'int' pero es 'string'
  - Línea 130: índice de array debe ser 'int' pero es 'bool'
  - Línea 142: Variable local duplicada 'x' en alcance 'duplicadosLocales'
  - Línea 144: Variable local duplicada 'y' en alcance 'duplicadosLocales'
  - Línea 149: Array global duplicado 'arrGlobal'
  - Línea -1: Error de tipado fuerte - Operación inválida entre 'int' y 'float'. Debes convertir uno de los valores.
  - Línea 160: Variable local duplicada 'main1' en alcance 'NAVIDAD'
  - Línea 162: Error de parámetros - la función 'suma' esperaba 2 argumentos pero recibió 1.
  - Línea 166: Error de análisis - función 'funcionFantasma' no declarada.
  - Línea 164: variable no declarada 'noExiste'
  - Línea 169: asignación incompatible - variable 'texto' tipo 'string' pero asignado 'int'

Total: 44 errores encontrados
```

11. Ejemplo analisis semántico sin errores

```
========== ANALISIS SEMANTICO ==========

? Sin errores semánticos
Analisis semántico exitoso.
```

12. Ejemplo opción 5 con archivo valido

```
========== ANALISIS SEMANTICO ==========

? Sin errores semánticos
ARCHIVO VALIDO
```

13. Ejemplo opción 5 con archivo invalido

```
Total: 44 errores encontrados
CONTIENE ERRORES
```

14. Ejemplo de generación de TAC

```
========== C?DIGO INTERMEDIO (THREE-ADDRESS CODE) ==========

    ========== INICIO C?DIGO INTERMEDIO ==========

    DECLARE mi_lista : int[2x2]
    mi_lista[0][0] = 10
    mi_lista[0][1] = 20
    mi_lista[1][0] = 30
    mi_lista[1][1] = 40

FUNCTION factorial:
    t0 = n <= 1
    if t0 goto L1
    goto L2
L1:
    return 1
    goto L0
L2:
    t1 = n - 1
    param t1
    t2 = call factorial, 1
    t3 = n * t2
    return t3
L0:
END_FUNCTION


FUNCTION suma_rango:
    total = 0
    i = inicio
L3:
    t4 = i <= fin
    if t4 goto L4
    goto L5
L4:
    t5 = total + i
    total = t5
    t6 = i + 1
    i = t6
    goto L3
L5:
    return total
END_FUNCTION


FUNCTION NAVIDAD:
    param 5
    t7 = call factorial, 1
    f = t7
    show f
```

15. Ejemplo generación MIPS

```
--- C?DIGO MIPS GENERADO ---
.data
    .align 2
    newline: .asciiz "\n"
    resultado_global: .word 0
    mi_lista: .word 0, 0, 0, 0

.text
    .align 2
    .globl main

main:
    # Asignar a array global mi_lista[0][0] = 10
    li $t0, 10
    sw $t0, mi_lista + 0
    # Asignar a array global mi_lista[0][1] = 20
    li $t0, 20
    sw $t0, mi_lista + 4
    # Asignar a array global mi_lista[1][0] = 30
    li $t0, 30
    sw $t0, mi_lista + 8
    # Asignar a array global mi_lista[1][1] = 40
    li $t0, 40
    sw $t0, mi_lista + 12

    jal NAVIDAD
    li $v0, 10
    syscall

factorial:
    # Prólogo
    addi $sp, $sp, -128    # Reservar espacio en pila
    sw $ra, 124($sp)       # Guardar dirección de retorno
    sw $fp, 120($sp)       # Guardar frame pointer
    addi $fp, $sp, 128     # Establecer nuevo frame pointer
    sw $a0, 0($sp)    # Guardar parámetro n

    lw $t0, 0($sp)
    li $t1, 1
    sle $t2, $t0, $t1    # n <= 1
    sw $t2, 4($sp)
    lw $t0, 4($sp)
    bnez $t0, L1    # Saltar si t0 != 0
    j L2
L1:
    li $v0, 1
    # Return
    lw $ra, 124($sp)
    lw $fp, 120($sp)
```

Descripción del problema

Contexto

En los **Proyectos #1 y #2**, se establecieron las bases del compilador mediante el análisis léxico y sintáctico. El sistema ya es capaz de reconocer tokens complejos y validar que la estructura del programa siga la gramática BNF, construyendo un Árbol de Sintaxis Abstracta (AST) y gestionando la recuperación de errores en modo pánico. El lenguaje utiliza símbolos únicos como ¿ ?, ¡ !, Σ, @, y ~, lo que requiere un manejo preciso de la codificación y la jerarquía de operadores.

Objetivos del Proyecto

El **Proyecto #3** representa la culminación del compilador, integrando las fases de **análisis semántico** y **generación de código destino**. Mientras las fases anteriores se enfocaban en la forma y estructura, el análisis semántico verifica la validez lógica del programa, aplicando reglas de **tipado fuerte y explícito**. Finalmente, el generador de código traduce la representación interna (AST y Código Intermedio) al lenguaje ensamblador **MIPS**, permitiendo que el programa sea ejecutado en hardware simulado mediante la herramienta **QtSpim**.

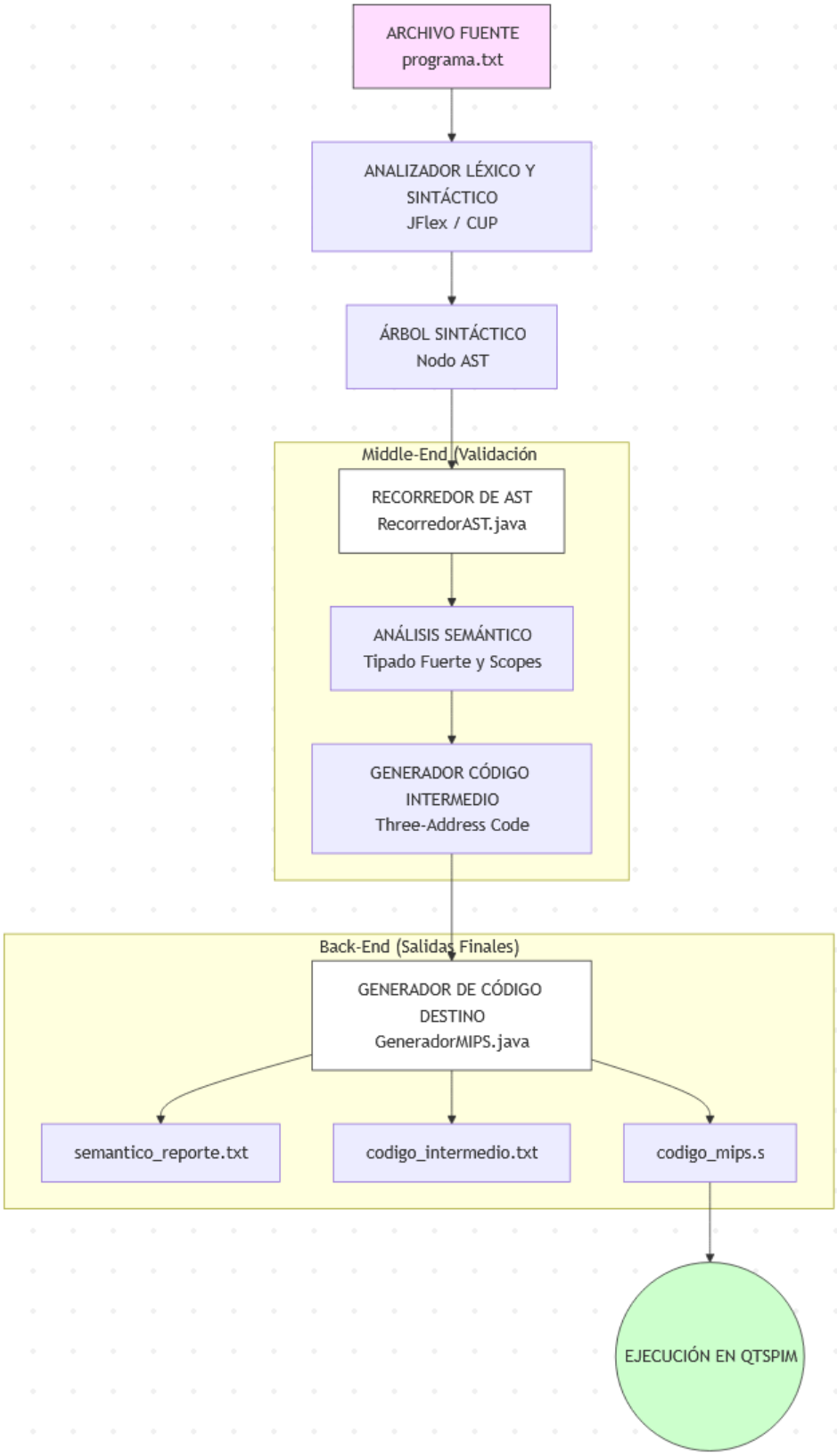
Diseño del programa

El desarrollo de esta fase final se basó en la implementación de un motor de validación lógica y un traductor de bajo nivel. Una de las decisiones de diseño más críticas fue la adopción de una etapa de **Código Intermedio (TAC - Three Address Code)** antes de la generación MIPS; esto permitió desacoplar la lógica del lenguaje de las restricciones físicas de los registros del procesador.

El análisis semántico se integró directamente en el recorredor del AST, validando que:

1. Las variables y funciones estén declaradas antes de su uso.
2. No existan asignaciones entre tipos incompatibles (ej. asignar float a int sin conversión).
3. Se respete el ámbito (scope) de las variables en funciones, bucles for y bloques loop.

Por otra parte, el compilador extendió su arquitectura en capas para incluir el backend:


ARCHIVO FUENTE
programa.txt
ANALIZADOR LÉXICO Y SINTÁCTICO
JFlex / CUP
ÁRBOL SINTÁCTICO
Nodo AST
Middle-End (Validación
RECORREDOR DE AST
RecorredorAST.java
ANÁLISIS SEMÁNTICO
Tipado Fuerte y Scopes
GENERADOR CÓDIGO INTERMEDIO
Three-Address Code
Back-End (Salidas Finales)
GENERADOR DE CÓDIGO DESTINO
GeneradorMIPS.java
semantico_reporte.txt
codigo_intermedio.txt
codigo_mips.s
EJECUCIÓN EN QTSPIM

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.

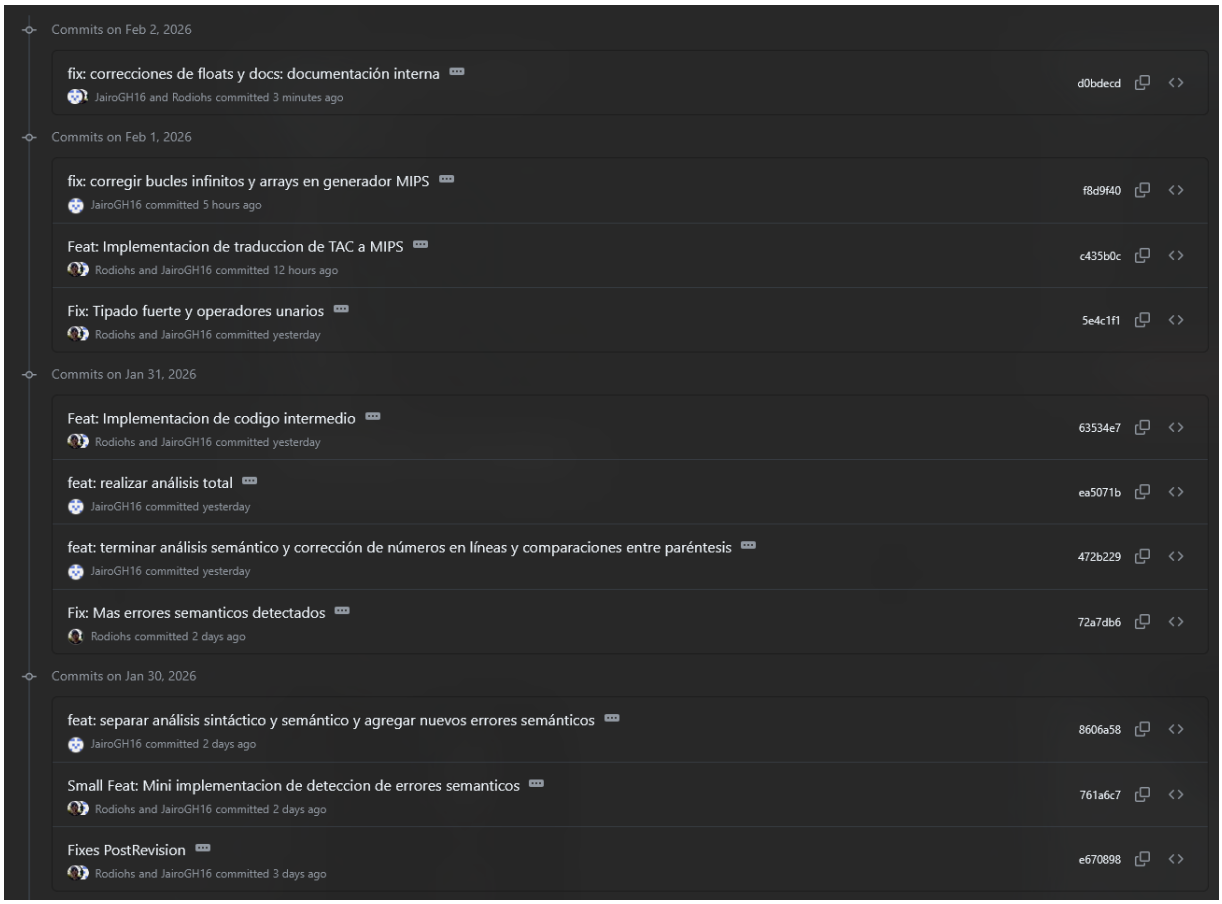
Java Standard Library proporcionó las estructuras de datos necesarias sin dependencias externas. Se utilizaron clases de java.io para lectura y escritura de archivos (FileReader, PrintWriter), y de java.util para estructuras como ArrayList y HashMap en la implementación de la tabla de símbolos y el árbol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

Analisis de Resultados

Archivo	Características	Estado
Main.java	Completamente funcional con el menú para todos los casos	✓ **Completo**
GeneradorCodigoIntermedio.java	Generacion correcta de TAC	✓ **Completo**
GeneradorMIPS.java	Traducción completa a MIPS	✓ **Completo**

RecorredorAST.java	Clase principal para recorrer el AST y Tabla SImbolos	**✓ Completo**

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.