Informe Técnico – Compilador para C

Materia: Técnicas de Compilación
Integrantes: Sergio Agustin Lopez

Docente: Francisco Ameri
Fecha: 03/10/2025

1. Introducción

El presente trabajo consiste en la implementación de un compilador para un subconjunto del lenguaje C, desarrollado en Java utilizando la herramienta ANTLR4 para la generación de lexer y parser.

El objetivo fue integrar los contenidos de la materia en un proyecto completo: desde el análisis léxico y sintáctico hasta el chequeo semántico y la generación de un código intermedio en tres direcciones (TAC).

2. Especificación del subconjunto de C

Tipos soportados: int, char, double, bool, float, void (en funciones).
Estructuras de control: if/else, for, while.
Elementos del lenguaje:
- Declaración de variables (simples y arrays).
- Declaración de funciones con parámetros.
- Expresiones aritméticas y lógicas.
- Asignaciones y llamadas a funciones.
- Sentencias return.

Limitaciones conocidas:

- No se llegó a optimizaciones del TAC.

3. Arquitectura del compilador

1. Léxico (ANTLR4 Lexer): definición de tokens, manejo básico de comentarios y espacios.
2. Sintáctico (ANTLR4 Parser): gramática sin ambigüedades, reglas para declaraciones,

expresiones y estructuras de control.
3. Semántico:
  - Tabla de símbolos jerárquica con scopes anidados.
  - Chequeos de tipos y doble declaración.
  - Distinción entre errores y warnings (ej: variable no usada).
4. Intermedio (TAC):
  - Generación de código de tres direcciones.
  - Expresiones binarias, llamadas, retornos, if/else, for, while.
5. Optimizador: no implementado.

4. Implementación

4.1 Gramática ANTLR4

Se desarrolló una gramática .g4 con soporte para expresiones aritméticas/lógicas, precedencia y asociatividad correctas. Se agregaron reglas específicas para for, if, while, y declaraciones de funciones.

4.2 AST

El árbol de sintaxis se maneja directamente con las clases generadas por ANTLR4, y se imprime para depuración.

4.3 Análisis Semántico

- Tabla de símbolos implementada.

- Validación de identificadores duplicados, variables no declaradas, variables no inicializadas.
- Warnings para variables declaradas y no usadas.
- Manejo de errores y advertencias con colores y formato unificado.

4.4 Código Intermedio (TAC)

- Forma de tres direcciones.
- Labels generados para saltos en if, while y for.
- Temporales para expresiones aritméticas.

...

- Código:
- if (a < b) { a = a+1; } else { b = b-1; }

- TAC:
- t1 = a < b
- if t1 goto L1
- goto L2
- L1: a=a+1; goto L3
- L2: b=b-1
- L3:

L2:

5. Pruebas y resultados

Se probaron casos de:
- Variables globales y locales.
- Doble declaración (error).
- Variables no usadas (warning).
- Uso de variables no inicializadas (error).
- Llamadas correctas e incorrectas a funciones.
- Estructuras de control anidadas.

La salida se valida en consola y en archivos reports/semantic.txt, reports/warnings.txt y reports/intermediate.txt.

6. Dificultades y soluciones

Lo más desafiante fue retomar la materia después de casi 5 años. Hubo que recuperar los trabajos previos (TP1 y TP2), entenderlos en detalle y luego extenderlos hasta un compilador completo.

Otro punto complicado fue manejar la gramática y los scopes de la tabla de símbolos, pero con iteraciones y depuración se logró estabilizar el análisis semántico.

7. Conclusiones

El proyecto permitió aplicar de forma práctica los conceptos de la materia: desde gramática y ANTLR hasta análisis semántico y generación de código intermedio.

Como trabajo futuro quedaría agregar un optimizador (propagación de constantes,

eliminación de código muerto) y quizás un backend que traduzca el TAC a ensamblador o LLVM IR.

8. Referencias

- Documentación de ANTLR4.
- Apuntes de la materia “Técnicas de Compilación”.
- Ejemplos de compiladores de C-mini en repositorios públicos.