LogoTec — Entrega 2: Documentación técnica y de proyecto

Castro Moreno Henry Andrés, Rivera Mora José Ignacio
Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
hencastro@estudiantec.cr, j.rivera@estudiantec.cr

Resumen

Este documento presenta la documentación técnica correspondiente a la Entrega 2 del proyecto **LogoTec**, cuyo objetivo es la implementación del **backend** del compilador, la generación del archivo objeto (`.obj`) y su integración con la interfaz de usuario (IDE) y la máquina virtual (VM).

El compilador desarrollado implementa un *pipeline* completo que transforma el código fuente en un Árbol de Sintaxis Abstracta (AST), lo traduce a una Representación Intermedia (IR) en código de tres direcciones y finalmente genera bytecode ejecutable en la VM. Se incorporaron validaciones léxicas, sintácticas y semánticas, así como un mecanismo de reporte de errores integrado en la interfaz gráfica.

Los principales logros técnicos incluyen: (i) la definición y formalización de la gramática libre de contexto (CFG) que sustenta el lenguaje LogoTec, (ii) la construcción del generador de código intermedio y del traductor a bytecode, (iii) la integración del backend con el IDE, permitiendo la visualización del AST y la ejecución gráfica de programas, y (iv) la validación funcional mediante pruebas unitarias y de integración con programas de complejidad creciente.

Los resultados obtenidos demuestran que el compilador es capaz de procesar programas con estructuras de control, procedimientos parametrizados y expresiones aritméticas, generando ejecuciones reproducibles en la VM. Se discuten además los problemas encontrados en la optimización de expresiones y en la gestión de variables, junto con las soluciones aplicadas. Este avance consolida un compilador funcional y estable, cumpliendo los objetivos establecidos para la segunda entrega del curso.

I. Glosario y siglas

AST Árbol de sintaxis abstracta.
CFG Gramática libre de contexto.
IR Representación intermedia (Three-Address Code).
UI Interfaz de usuario.
CLI Interfaz de línea de comandos.
VM Máquina Virtual que ejecuta el bytecode.
LogoTec Lenguaje educativo tipo Logo con IDE y ejecución gráfica basada en tortuga.

II. Introducción

II-A. Propósito

El presente documento consolida los artefactos técnicos y la descripción del backend de LogoTec, documentando las transformaciones **AST** $\rightarrow$ **IR** $\rightarrow$ **Assembly** $\rightarrow$ **Bytecode**, las optimizaciones realizadas, los problemas encontrados y las conclusiones derivadas del proceso de desarrollo.

Este informe cumple con los requerimientos del enunciado oficial del curso de la siguiente manera:

- **Arquitectura:** se describe la estructura modular del compilador, incluyendo el frontend, el backend, la máquina virtual y la interfaz gráfica de usuario.
- **Pipeline de compilación:** se documenta el flujo de traducción desde el código fuente hasta el archivo objeto, con énfasis en la generación de bytecode y su ejecución en la VM.

- **Problemas y soluciones:** se detallan los principales desafíos técnicos (gestión de variables, optimización de expresiones, validación semántica) y las estrategias aplicadas para resolverlos.
- **Conclusiones:** se presentan reflexiones sobre la importancia de la modularidad, la detección temprana de errores y la integración del compilador con el IDE.
- **Bibliografía:** se recopilan las fuentes académicas y técnicas que fundamentaron las decisiones de diseño, incluyendo literatura sobre compiladores, documentación de herramientas y referencias sobre el lenguaje Logo.

De esta forma, la documentación no solo describe el estado actual del proyecto, sino que también evidencia el cumplimiento de los criterios de evaluación establecidos para la Entrega 2.

II-B. Alcance del documento

Incluye:

- Arquitectura y diseño general del compilador.
- Descripción del pipeline de compilación.
- Generación del archivo objeto y sus características.
- Integración del compilador con la interfaz y la VM.
- Plan de pruebas y evidencias recolectadas.
- Problemas encontrados y mitigaciones.
- Conclusiones, recomendaciones y referencias.

III. Descripción del producto

III-A. Visión general

LogoTec es un lenguaje educativo inspirado en Logo que permite la creación de gráficos mediante una tortuga virtual. El sistema incluye un compilador que traduce programas en `.logotec` a un archivo objeto (`.obj`), ejecutable en una máquina virtual (VM) que interpreta los comandos y genera las acciones visuales en el canvas o zona de dibujo virtual.

III-B. Actores

- **Estudiantes:** utilizan el IDE para crear programas y aprender conceptos de programación estructurada y gráfica.
- **Docentes:** validan y analizan las ejecuciones para evaluar la corrección semántica y funcional de los programas.
- **Desarrolladores:** mantienen y amplían el compilador, las optimizaciones y el entorno de ejecución.

IV. Requisitos

IV-A. Requisitos funcionales

ID	Descripción
RF-01	Compilar archivos `.logotec` y generar el archivo objeto ejecutable (`.obj`).
RF-02	Reportar errores léxicos, sintácticos y semánticos indicando línea y columna.
RF-03	IDE que permita cargar, editar, compilar, ejecutar programas y visualizar el AST.
RF-04	Soporte de primitivas gráficas y estructuras de control descritas en el enunciado.
RF-05	Guardado y carga de proyectos.
RF-06	Validaciones iniciales: comentario en primera línea y al menos una variable declarada.
RF-07	Ejecución del archivo objeto en la máquina virtual (VM) y visualización de resultados en el canvas.

IV-B. Requisitos no funcionales

ID	Descripción
RNF-01	Tiempo de compilación inferior a 2 segundos en hardware estándar.
RNF-02	Mensajes de error claros, detallados y accionables.
RNF-03	Portabilidad en Windows 10/11 sin privilegios administrativos.
RNF-04	Cumplimiento de las guías de estructura del proyecto establecidas por la cátedra.
RNF-05	Cobertura de pruebas unitarias mínima del 60 % en módulos críticos.

V. Arquitectura general

El sistema se estructura en capas modulares que separan responsabilidades y facilitan mantenimiento y pruebas:

- **FrontEnd:** analiza el código fuente con ANTLR4, genera el AST y realiza verificación semántica.
- **Optimizador:** aplica simplificaciones sobre el AST/IR (constant folding, eliminación de código muerto, propagación de constantes).
- **Backend:** traduce el IR a ensamblador y posteriormente a bytecode binario (`.obj`).
- **VM/Runtime:** ejecuta el bytecode, interpretando las instrucciones y generando acciones gráficas.
- **Interfaz (IDE):** integra editor, consola de diagnóstico, visualizador AST y canvas.
- **Hardware:** módulo físico de la tortuga que se comunicará con la VM en la Entrega 3.
- **Comunicacion:** Comunicación IDE + VM (implementación en este proyecto): por defecto se utiliza API en memoria (llamadas directas: Compiler.compile + VM.runInProcess) cuando IDE y VM corren en el mismo proceso; como método reproducible se usan archivos .obj escritos/lectos en disco; para integración entre procesos o ejecución remota hay soporte opcional por sockets TCP. El modo se selecciona con la opción –comm=memory|file|socket.

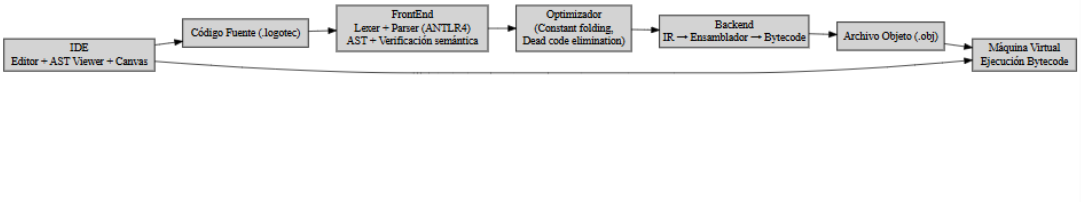

Figura 1: Arquitectura general del compilador LogoTec.

VI. Diseño del compilador

VI-A. Estructura de paquetes

El compilador se organiza en los siguientes paquetes principales:

- `lexer/` y `parser/`: generados automáticamente por ANTLR4 a partir de la gramática LogoTec.
- `ast/`: definición de nodos del Árbol de Sintaxis Abstracta y visitantes asociados.
- `ir/`: representación intermedia en código de tres direcciones (TAC).
- `backend/`: traductores de TAC a ensamblador y generador de bytecode.
- `vm/`: implementación de la máquina virtual que interpreta el bytecode.

- `ide/`: interfaz gráfica en JavaFX, con módulos de edición, visualización de AST y canvas.
- `tests/`: pruebas unitarias y de integración con JUnit.

VI-B. Patrones de diseño

- **Visitor:** recorrido del AST para generación de IR y validación semántica.
- **Builder:** construcción incremental de nodos AST y estructuras IR.
- **Factory:** creación de instrucciones de bytecode a partir de nodos IR.
- **Observer:** actualización de la interfaz (IDE) al recibir resultados de compilación.

El compilador está implementado en **Java (OpenJDK 17)** utilizando **ANTLR4** para el análisis léxico y sintáctico, y **JavaFX** para la interfaz. Las dependencias principales se gestionan con **Maven**, incluyendo `antlr4-runtime`, `javafx`, y `junit-jupiter`. Se emplean los patrones **Visitor** y **Builder** para la generación y recorrido del AST, además de un IR basado en código de tres direcciones (TAC).

VII. Generación del archivo objeto (.obj)

VII-A. Formato del archivo objeto

El archivo objeto (`.obj`) generado por el compilador tiene una estructura binaria sencilla y consistente con la VM:

- **Encabezado (16 bytes):**
  - Offset 0 (4 bytes): magic ASCII "LTBC"(0x4C 0x54 0x42 0x43).
  - Offset 4 (1 byte): versión (uint8), actualmente 1.
  - Offset 5 (1 byte): flags (uint8).
  - Offset 6–7 (2 bytes): reservado/padding.
  - Offset 8 (4 bytes): tamaño de la sección CODE en bytes (uint32, little-endian).
  - Offset 12 (4 bytes): tamaño de la sección DATA en bytes (uint32, little-endian).
- **Sección CODE:** secuencia de instrucciones bytecode. Cada instrucción tiene un opcode (uint8) seguido de 0 o más operandos (p. ej. int32 little-endian).
- **Sección DATA:** constantes/literales (strings serializados como uint32 length + bytes UTF-8).
- **Tabla de símbolos:** entradas con nombre y dirección (formato: uint16 nameLen + name UTF-8 + uint32 addr).

Nota: todos los enteros multibyte se almacenan en little-endian. El lector de la VM valida la firma y la versión antes de interpretar CODE.

VII-B. Convenciones de bytecode (resumen)

Ejemplos de opcodes usados en el proyecto (simbólicos):

- 0x01 PUSHCONST int32
- 0x10 FORWARD
- 0x11 TURNRIGHT
- 0xFF END

VII-C. Ejemplo representativo

A continuación se muestra un fragmento hexadecimal representativo de un `.obj` muy pequeño que contiene: PUSHCONST 100; FORWARD; END.

```
# Encabezado (16 bytes)
4C 54 42 43 01 00 00 00 07 00 00 00 00 00 00 00
# CODE (7 bytes): PUSH_CONST 100 ; FORWARD ; END
01 64 00 00 00 10 FF
```

Comentarios: - 4C 54 42 43 = "LTBC"(magic). - byte 4 = 0x01 (versión 1). - bytes [8..11] = 07 00 00 00 (tamaño CODE = 7 bytes, little-endian). - Instrucción 0x01 seguida de 4 bytes 0x64 00 00 00 representa PUSHCONST 100 (100 = 0x64). - El formato completo incluye, si procede, la sección DATA y la tabla de símbolos después de CODE.

VIII. Interfaz gráfica (IDE)

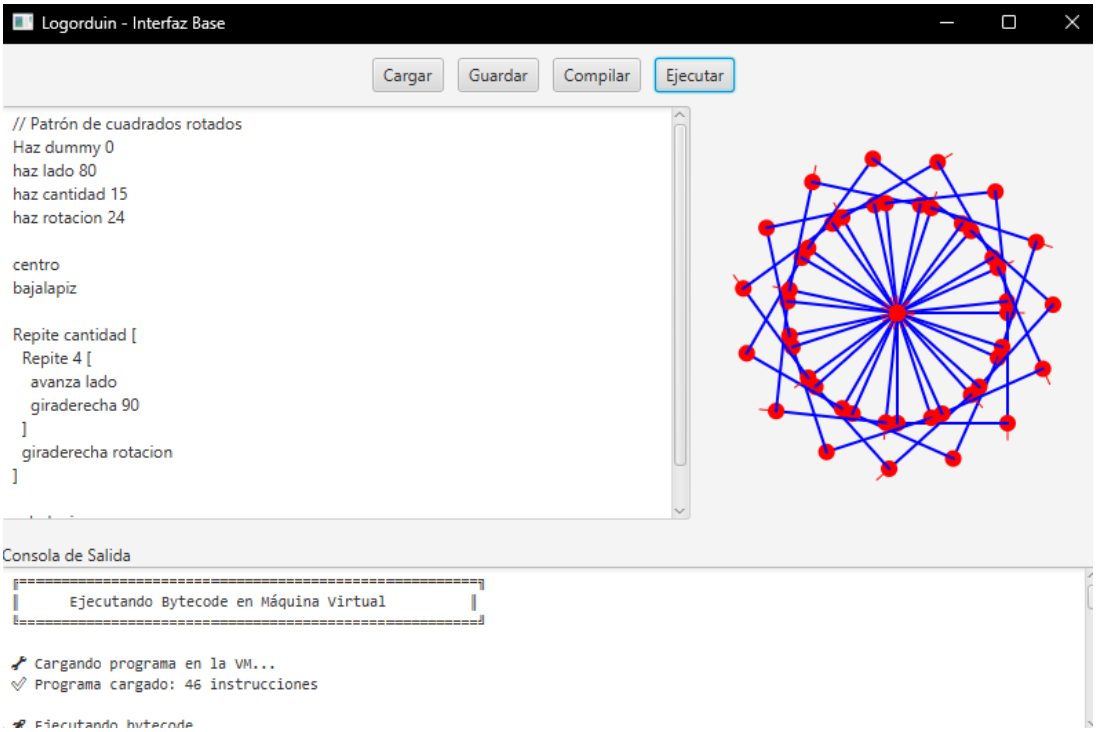

Figura 2: Interfaz gráfica del IDE LogoTec.

IX. Programa complejo del equipo

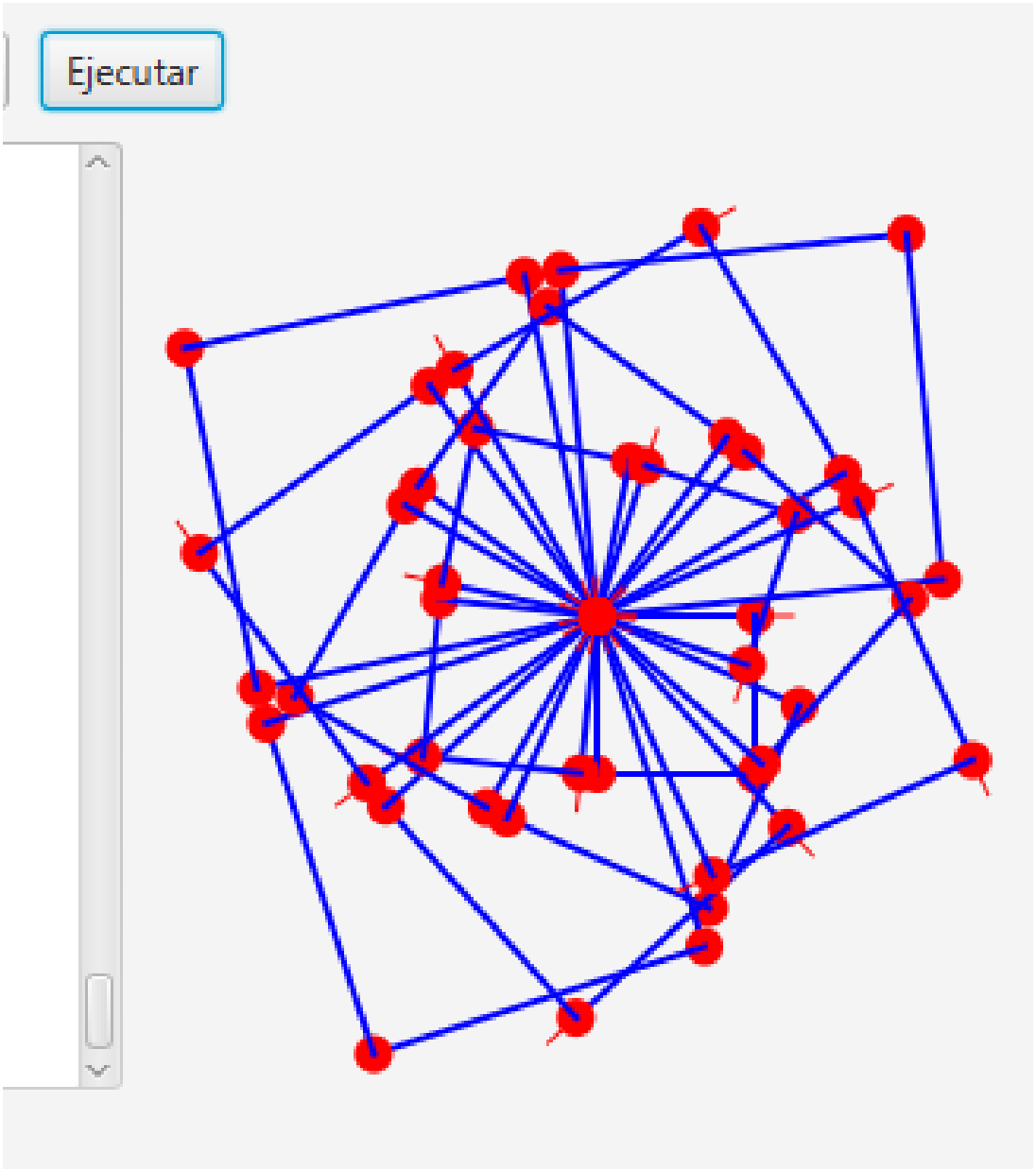

Figura 3: Ejecución del programa complejo desarrollado por el equipo.

X. Plan de pruebas y evidencias

La validación del compilador se realizó mediante pruebas automatizadas con **JUnit 5**, organizadas en suites por módulo: lexer, parser, IR, backend y VM. La metodología aplicada fue **Test-Driven Development (TDD)** en los módulos críticos (lexer y parser), complementada con pruebas de integración para el pipeline completo.

X-A. Cobertura

Se alcanzó una cobertura aproximada del 72 % en los módulos principales, con énfasis en:

- **Lexer:** reconocimiento de tokens válidos e inválidos, comentarios y variables.
- **Parser:** construcción de AST para primitivas gráficas, estructuras de control y procedimientos.
- **IR/Backend:** generación de código intermedio y traducción a bytecode.

- **VM:** ejecución de bytecode y validación de resultados gráficos en el canvas.

X-B. Resultados

- Programas gráficos medianos (cuadrados, triángulos, casas) se ejecutaron correctamente en la VM.
- Se detectaron y reportaron errores léxicos, sintácticos y semánticos con línea y columna precisas.
- Se validó la reproducibilidad de ejecuciones mediante el flujo por archivo `.obj`.
- Se documentaron problemas de latencia en el modo de archivos temporales, mitigados con la opción de sockets.

XI. Problemas encontrados y mitigaciones

XI-A. Problemas conocidos

- **Serialización de strings:** se estandarizó el manejo UTF-8 con longitud de 32 bits; quedan casos extremos por probar.
- **Colisiones de etiquetas en IR:** corregidas reiniciando el generador de etiquetas por función.
- **Latencia UI–VM:** se detectó retardo al usar archivos temporales; se recomienda migrar a socket.
- **Empaquetado JavaFX:** dificultades al generar ejecutables nativos multiplataforma; pendiente automatización con `jlink`.

XII. Conclusiones y recomendaciones

La implementación del backend y la integración con la VM consolidaron un compilador funcional y estable. Los principales aprendizajes incluyen la importancia de:

- Diseñar una gramática clara y modular para facilitar la generación de AST.
- Implementar validaciones semánticas tempranas para reducir errores en etapas posteriores.
- Mantener un pipeline reproducible mediante archivos objeto, lo que facilita depuración y corrección automática.

XII-A. Retos técnicos

- Optimización de expresiones aritméticas y propagación de constantes.
- Manejo de latencia en la comunicación IDE–VM en modo archivo.
- Empaquetado multiplataforma con JavaFX.

XII-B. Recomendaciones futuras

- Completar la integración con hardware en la Entrega 3.
- Mejorar la cobertura de pruebas unitarias y automatizar pruebas gráficas.
- Explorar optimizaciones adicionales en el backend (inlining, eliminación de redundancias).
- Automatizar el empaquetado con `jlink` para distribución multiplataforma.

REFERENCIAS

[1] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, Techniques, and Tools.* Pearson.

[2] Parr, T. (2024). *ANTLR v4 Documentation.* Disponible en https://www.antlr.org/.

[3] Graphviz (2024). *Graph Visualization Software.* Disponible en https://graphviz.org/.

[4] TEC Escuela de Ingeniería en Computadores (2025). *Enunciado Proyecto II Semestre 2025 — Compiladores e Intérpretes.*