UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Diseño e implementación de un lenguaje de programación y su compilador

Presentado por Adrián Zamora Sánchez
en Universidad de Burgos — 3 de octubre de 2025
Tutor: Dr. César Ignacio García Osorio

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Adrián Zamora Sánchez, con DNI 48838775T, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 3 de octubre de 2025

Vº. Bº. del Tutor:

César Ignacio García Osorio

Resumen

En este proyecto se aborda la creación de un lenguaje de programación, herramienta fundamental dentro del campo de la ingeniería informática. Para que este lenguaje sea utilizable, se implemnetará un compilador propio, capaz de llevar este lenguaje a ejecución. Este proyecto se limitará a algunas funcionalidades clave, puesto que querer abordar cuestiones como OOP podrían aumentar la complejidad del proyecto demasiado.

Para la realización del proyecto se tratan de utilizar las herramientas más modernas y extendidas en la creación de lenguajes y compiladores: ANTLR, LLVM, CMake y C++ como lenguaje de programación.

Descriptores

Compilador, lenguaje de programación, ANTLR, LLVM

Abstract

This project focuses on the creation of a programming language, a fundamental tool within the field of computer engineering. To make this language usable, a custom compiler will be implemented, capable of executing programs written in it. The scope of the project will be limited to a set of key functionalities, as tackling more advanced features such as OOP could increase the complexity beyond what is feasible.

For the development of the project, the most modern and widely adopted tools in language and compiler design will be used: ANTLR, LLVM, CMake, and C++ as the programming language.

Keywords

Compiler, programmin language, ANTLR, LLVM

Índice general

Índice de figuras

Índice de tablas

1. Introducción

En la informática moderna estamos acostumbrados a disponer de numerosos lenguajes de programación, los cuales empleamos sin detenernos a analizar más allá de sus potenciales usos y librerías disponibles. Sin embargo, estas complejas herramientas cuentan con un gran trabajo detrás, el cual permite a los programadores utilizar estas herramientas sin preocuparse de cómo funcionan realmente.

La creación de los compiladores es una ciencia que ha evolucionado mucho a lo largo de los años, no solo por el surgimiento de nuevos lenguajes y sus compiladores asociados, muchos lenguajes que llevan con nosotros desde los 70s han experimentado cambios significativos en sus compiladores, además de versiones alternativas, cada una enfocanda en unos objetivos diferentes.

Ejemplos de proyectos que comenzaron siendo pequeños lenguajes de programación o lenguajes de dominio concreto que han crecido hasta ser herramientas ampliamente utilizadas son:

- **HTML + CSS**: Nacieron como lenguajes muy específicos que hoy día son estándares globales, tienen un ecosistema inmenso, disponen de importantes frameworks e incluso han inspirado a lenguajes y herramientas fuera del entorno web.
- **MATLAB**: Diseñado en los 80s para ser un lenguaje específico de algebra lineal, que hoy día se utiliza ampliamente en ingeniería, robótica, procesamiento de señales, visión por computadora, etc.
- **SQL**: Comenzó en los 70s como un lenguaje que IBM desarrollaría para hacer consultas en sus bases de datos relacionales, sin embargo,

fue ampliamente adoptado como un estándar que lo ha llevado a estar presente en prácticamente todos los motores de bases de datos relaciones modernos.

Memoria

Esta memoria aborda los siguientes apartados:

- **Objetivos**:
- **Conceptos teóricos**:
- **Técnicas y herramientas**:
- **Aspectos relevantes del desarrollo del proyecto**:
- **Trabajos relacionados**:
- **Conclusiones y líneas de trabajo futuras**:

Anexos

Adicionalmente, se adjuntan los siguientes anexos:

- **Anexo A. Plan de proyecto**:
- **Anexo B. Requisitos**:
- **Anexo C. Diseño**:
- **Anexo D. Manual de programador**:
- **Anexo E. Manual de usuario**:
- **Anexo F. Sostenibilización curricular**:

2. Objetivos del proyecto

Requisitos software

- **Diseñar un lenguaje**:
- **Compilador capaz de reconocer el lenguaje**:
- **Generación de errores encontrados en la entrada**:
- **Herramienta de análisis (debug) visual**:
- **Compilación desde IR**:
- **Utilidades temporales propias del lenguaje**:

Objetivos técnicos

Utilizar ANTLR, puesto que es una herramienta moderna y reconocida entre los programas generadores de reconocedores de lenguajes.

Utilizar LLVM, también una de las herramientas más actuales y utilizadas en compiladores reales, capaz de generar un IR que permite crear un compilador extremadamente portable independientemente de la arquitectura.

Utilizar C++ como entorno, pues tiene buena integración con ANTLR (C++ ANTLR Runtime Environment) y con LLVM, además de una gran capacidad para manejar OOP.

Crear al menos una funcionalidad con la que se permita al usuario visualizar el programa escrito junto a algunos detalles minimos, por ejemplo, mediante la representación del AST en un grafo que se pueda guardar como

una imagen del programa. Sirviendo tanto como una funcionalidad de debug visual como una característica interesante desde el punto de vista academico.

3. Conceptos teóricos

Sobre lenguajes

Clasificación de lenguajes : Los lenguajes de programación suelen clasificarse según su nivel de abstracción (alto o bajo nivel), paradigma (imperativo, declarativo, OOP, funcional), propósito del lenguaje (propósito general, lenguaje específico del dominio) y por forma de ejecución (compilados o interpretados).

Tipado : El tipado de un lenguaje se refiere a la forma en la que se gestionan los tipos de los datos, en los lenguajes con tipado estático, los tipos deben ser definidos en las declaraciones y se hacen comprobaciones estrictas durante las asignaciones, por otro lado, los lenguajes de tipado dinámico permiten declaraciones sin especificar el tipo, en estos lenguajes normalmente se hacen transformaciones de datos en tiempo de ejecución.

Lenguaje máquina : El lenguaje máquina es aquel que puede ser ejecutado por un ordenador, este tipo de lenguaje suele ser generado por un compilador, puesto que sería extremadamente ineficiente de escribir por un ser humano.

Gramaticas formales : Las gramáticas formales son estructuras de reglas con las cuales se pueden generar las cadenas de caracteres permitidas por un lenguaje formal, esta gramática no tiene en cuenta el significado de las fórmulas bien formadas, únicamente la admisión de su forma.

Sobre compiladores

Front end :

Back end :

Análisis léxico : consiste en el proceso por el cual se analizan las secuencias de caracteres de un texto y se separan en tokens o lexemas. A la herramienta que realiza este proceso se le llama scaner, lexer o tokenizer

Análsis sintáctico : este proceso trata de agrupar los tokens obtenidos durante el análisis léxico, para ello se vale de reglas que generan otras estructuras de datos (como AST) desde la cadena de tokens.

AST : el Abstract Syntax Tree (árbol de sintaxis abstracta), es una estructura de datos que se utiliza comúnmente en compiladores para representar producciones sintácticas, donde los tokens forman una estructura gerárquica de árbol que representa el programa y ayuda a su posterior interpretación.

Tabla de símbolos : Es una estructura de datos utilizada en compiladores para asociar cada símbolo de un programa con su ubicación, alcance y tipo de dato. Dentro de un compilador cumple un papel fundamental entre el front end y el back end del compilador.

Análisis semántico : El proceso de análisis semántico comprueba la corrección de las producciones válidas formadas en el análisis sintáctico, algunas correcciones podrían ser la verificación de tipos y corrección en asignaciones y expresiones.

Representación intermedia : la representación intermedia o IR por sus siglas en inglés, hace referencia a una representación que queda a medio camino entre el lenguaje fuente y el lenguaje máquina. Esta representación nos permite tener una mejor base para realizar algunas optimizaciones y conversiones a diferentes formas de lenguaje máquina, según la arquitectura objetivo.

Optimización : las optimizaciones consisten en cambios o mejoras en la forma del lenguaje, sin alterar el significado original. Estas pueden ser eliminaciones de código no alcanzable, redefiniciones de algunas estructuras para evitar generar más variables de las necesarias, gestión eficiente de la memoria, etc.

Generación de código final : Este paso es el que convierte el IR en código máquina listo para ser ejecutado.

4. Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

5. Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

6. Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

7. Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

[1]

Bibliografía

[1] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.