

**TRABAJO FINAL**

**CURSO DE FUNDAMENTOS TEORÍA DE COMPILADORES**

Carrera de Ciencias de la Computación

Sección: CC61

Alumnos:

* Gabriel Alonso Reyna Alvarado
* Ian Joaquin Sanchez Alva
* Ibrahim Imanol Jordi Arquinigo Jacinto
* Alejandro Olaf López Flores

Septiembre 2024

Índice

# 

[**1. Problema y motivación 3**](#_heskwha8s7er)

[1.1. Problema 3](#_a43c6vvukrxb)

[1.2. Motivación 3](#_wc67y9efrjd)

[1.2.1. Automatización del análisis léxico y sintáctico 3](#_6a860iimhzfo)

[1.2.2. Flexibilidad en la evolución del lenguaje 3](#_e3xwhh5ua6qa)

[1.2.3. Optimización mediante integración con C++ 3](#_jkbxfho3yr4e)

[1.2.4.Reducción de la complejidad en la definición de la gramática 3](#_3hrvffmunvq)

[**2. Objetivos 4**](#_1ek7wv6mn62o)

[2.1. Automatizar la creación del análisis léxico y sintáctico del lenguaje 4](#_celyd9l4ca69)

[2.2. Facilitar la evolución y modificación del lenguaje 4](#_8l0bzekjji9v)

[2.3. Optimizar el rendimiento del lenguaje mediante la integración con C++ 4](#_8it29ufwh6ll)

[2.4. Reducir la complejidad en la definición y manejo de la gramática 4](#_wm76cy8afikx)

[**3. Gramática en ANTLR4 4**](#_48s6om4s7llu)

[3.1. Struct 5](#_denas4srl3wh)

[3.2. Test #1 5](#_ut9745np5vkn)

[3.3. Test #2 6](#_vui2cpvic3ms)

[3.4. Clases 6](#_7nzb90sisbhe)

[**4. Referencias Bibliográficas: 7**](#_4nabsrr9xnbd)

# 

# Problema y motivación

## 1.1. Problema

Este trabajo consiste en la creación de un lenguaje de programación personalizado utilizando **ANTLR** para generar los analizadores léxicos y sintácticos, y planear implementar el lenguaje en **C++** en un sentido más natural y para **hispanohablantes**. El reto principal es definir una gramática robusta que pueda procesar las construcciones del lenguaje, generar el código correspondiente y hacerlo eficiente en un entorno C++. Ademas usaremos **LLVM** como backend para poder generar el **código IR de nuestro lenguaje**.

## 1.2. Motivación

### 1.2.1. Automatización del análisis léxico y sintáctico

La construcción manual de analizadores léxicos y sintácticos para un nuevo lenguaje de programación es un proceso complejo y propenso a errores. ANTLR proporciona una solución automatizada que permite definir la gramática del lenguaje de manera estructurada, generando automáticamente el código necesario para procesar las entradas de manera precisa. Esto no solo ahorra tiempo, sino que también mejora la consistencia y precisión del análisis, especialmente en lenguajes con sintaxis intrincada.

### 1.2.2. Flexibilidad en la evolución del lenguaje

Los lenguajes de programación suelen pasar por varias iteraciones y ajustes a medida que evolucionan. ANTLR facilita este proceso al permitir la modificación de la gramática de manera sencilla y rápida. Cada vez que se introducen nuevos elementos o reglas en el lenguaje, ANTLR regenera el analizador léxico y sintáctico sin necesidad de rehacer completamente la base del compilador. Esto aporta flexibilidad al desarrollo y garantiza que los cambios puedan ser implementados eficientemente.

### 1.2.3. Optimización mediante integración con C++

C++ es conocido por su capacidad de manejar memoria y recursos con gran precisión, lo que lo convierte en una opción ideal para crear compiladores y lenguajes de programación orientados al alto rendimiento. Al integrar ANTLR con C++, se aprovecha la eficiencia y el control que ofrece el lenguaje, optimizando el desempeño del código generado. Esto es especialmente relevante en aplicaciones de sistemas donde la eficiencia y el control de bajo nivel son esenciales.

### 1.2.4.Reducción de la complejidad en la definición de la gramática

Definir una gramática para un lenguaje de programación personalizado puede ser una tarea monumental, especialmente si el lenguaje incluye características avanzadas como expresiones anidadas o macros. ANTLR facilita este proceso al proporcionar una plataforma robusta para la definición y validación de gramáticas, reduciendo la complejidad y permitiendo al desarrollador centrarse en las funcionalidades esenciales del lenguaje en lugar de los detalles técnicos del análisis.

# Objetivos

## 2.1. Automatizar la creación del análisis léxico y sintáctico del lenguaje

El objetivo es utilizar ANTLR para generar automáticamente los analizadores léxicos y sintácticos del lenguaje, eliminando la necesidad de desarrollar manualmente estas partes fundamentales de un compilador o intérprete. Esto permitirá un procesamiento eficiente y preciso del código fuente.

## 2.2. Facilitar la evolución y modificación del lenguaje

Definir una gramática flexible y escalable que pueda ser modificada con facilidad es fundamental para permitir la evolución del lenguaje. ANTLR facilitará la actualización del análisis sintáctico a medida que se añaden nuevas características o se ajusten las reglas gramaticales.

## 2.3. Optimizar el rendimiento del lenguaje mediante la integración con C++

El objetivo es aprovechar las capacidades de C++ para optimizar el rendimiento del código generado por ANTLR. Esto implica garantizar que los analizadores generados sean eficientes y se integren perfectamente con el entorno de C++ para maximizar el control y la gestión de recursos en la fase de ejecución.

## 2.4. Reducir la complejidad en la definición y manejo de la gramática

Otro objetivo clave es simplificar la definición de la gramática del nuevo lenguaje utilizando ANTLR, asegurando que el proceso de crear un conjunto de reglas gramaticales sea claro y manejable. Esto permitirá reducir los errores y aumentar la productividad en el desarrollo del compilador o intérprete.

# Gramática en ANTLR4

La gramática que hemos usado en un inicio es de c++, pero corregiremos y juntaremos los códigos de cada uno del equipo en un solo archivo. Además de poder agregarle más lenguaje nativo español. Se hizo también un driver con lenguaje de c++ en donde trabajamos con memoria y operaciones como ++ID, –ID, ID++ y ID–. Nuestro sistema detecta cuando es en un inicio la operación ++ y cuando va después del ID. Además la gramática de cada uno del equipo se enfocó en funciones específicas para repartir el trabajo. Tenemos gramática para identificar estructuras como STRUCT, CLASS, Operaciones Matemáticas, IF else statements, while statements, for statements, iostreams, declaración, asignación, operadores ternarios. Además tenemos test de cada una de estas estructuras en el github para que cualquiera pueda probar la gramática que hemos definido.

# Referencias Bibliográficas:

* GitHub. (2024). *TP-Compiladores* [Repositorio de código]. GitHub.<https://github.com/forestgump22/TP-Compiladores>
* Guerrero, M. (s. f.). *Lenguaje latino*. <https://www.lenguajelatino.org/>
* Antlr4. (s. f.). *antlr4/doc/index.md at master · antlr/antlr4*. GitHub. <https://github.com/antlr/antlr4/blob/master/doc/index.md>
* Jeffery, C. L. (2021). *Build your own programming language*. O’Reilly Online Learning. <https://learning.oreilly.com/library/view/build-your-own/9781800204805/>

# 