

# Universidad Rafael Landívar

---

Ing. Pedro David Wong  
Compiladores

## Proyecto Final – Analizador Sintáctico

---

Andrés cleaves 12241923

Javier Ruano 1207022

Julio Portillo 1160822

Guatemala, 21 de mayo de 2025

## INDICE

1. 1. Introducción
2. 2. Objetivos
3. 2.1 Objetivo General
4. 2.2 Objetivos Específicos
5. 3. Descripción del Proyecto
6. 4. Arquitectura del Sistema
7. 5. Diseño e Implementación
8. 6. Pruebas y Validación
9. 7. Manual de Usuario
10. 8. Conclusiones
11. 9. Anexos
12. 10. Referencias

## 1. Introducción

El presente proyecto corresponde a la segunda fase del desarrollo de un compilador, como parte del curso de Compiladores impartido en la Facultad de Ingeniería. Esta fase se enfoca específicamente en la implementación de un **analizador sintáctico**, una pieza clave en la arquitectura de cualquier compilador, encargada de verificar que las expresiones del lenguaje fuente estén construidas correctamente de acuerdo con una gramática formal.

Para lograr esto, se utilizó la herramienta **ANTLR** (**A**nother **T**ool **f**or **L**anguage **R**ecognition), la cual facilita la creación de analizadores léxicos y sintácticos a partir de gramáticas definidas por el usuario. El proyecto admite expresiones algebraicas con operadores básicos (+, -, \*, /, ^) y asignaciones, utilizando agrupadores () y el símbolo de asignación => según lo requerido para el Grupo 09. También se desarrolló un **sistema evaluador** que permite calcular el resultado de las expresiones si estas no contienen errores o variables indefinidas.

Además, se construyó una **interfaz gráfica en React** que permite al usuario ingresar código fuente y recibir retroalimentación visual sobre tokens, errores y resultados. Esta interfaz se comunica con un **backend implementado en FastAPI**, que integra el motor de análisis generado con ANTLR. Este proyecto no solo pone en práctica conceptos fundamentales de la teoría de lenguajes y compiladores, sino que también desarrolla habilidades en diseño modular, trabajo con herramientas modernas y construcción de soluciones software orientadas al análisis de lenguaje.

## 2.1 Objetivo General

Desarrollar un sistema de análisis sintáctico utilizando ANTLR que valide expresiones algebraicas con variables y opere su evaluación mediante una arquitectura modular web.

## 2.2 Objetivos Específicos

- Definir una gramática ANTLR según los operadores, agrupadores y asignadores del Grupo 09.
- Implementar un backend que detecte errores léxicos y sintácticos.
- Integrar un frontend para mostrar resultados de análisis y evaluación.
- Incluir pruebas funcionales que demuestren el correcto desempeño del sistema.
- Elaborar documentación técnica y manual de usuario.

### 3. Descripción del Proyecto

El sistema desarrollado permite al usuario ingresar una serie de expresiones algebraicas mediante una interfaz gráfica. Estas expresiones son procesadas por el backend, el cual implementa una arquitectura modular con distintas etapas del análisis de código: primero se realiza el análisis léxico, luego el análisis sintáctico y finalmente, si no se detectan errores, se procede a la evaluación de la expresión.

Las expresiones aceptadas pueden contener números, variables, operadores aritméticos (+, -, \*, /, ^) y agrupadores. Según la normativa del Grupo 09, se utiliza el símbolo {} para agrupar expresiones y => para realizar asignaciones. La gramática fue definida utilizando ANTLR4, permitiendo generar el parser y el árbol de análisis correspondiente. Para evaluar las expresiones, se implementó un visitor que interpreta las asignaciones y permite reutilizar variables previamente definidas.

Desde el punto de vista del usuario, el sistema brinda una experiencia interactiva: permite escribir código, enviarlo al backend mediante solicitudes HTTP, y visualizar en tiempo real los tokens generados, posibles errores sintácticos y el resultado numérico de las expresiones si son válidas. En caso de errores, estos se presentan de forma clara, indicando el tipo de problema y su ubicación en el código.

Este proyecto representa una evolución respecto a la primera fase, donde se implementó únicamente el análisis léxico. Ahora, se introduce un nivel más avanzado de procesamiento del lenguaje, validando estructuras gramaticales completas y añadiendo capacidades de interpretación.

## 4. Arquitectura del Sistema

El sistema está diseñado bajo una arquitectura cliente-servidor, separando claramente las responsabilidades entre la interfaz de usuario (frontend) y el procesamiento lógico (backend). Esta separación permite una mayor escalabilidad, mantenibilidad y facilidad de pruebas.

El frontend fue construido utilizando **ReactJS** con el entorno de desarrollo **Vite** para una mayor velocidad de compilación y experiencia de desarrollo. Se aplicó **TailwindCSS** para una interfaz responsiva, moderna y de rápida personalización. Desde el navegador, el usuario puede escribir expresiones algebraicas en un editor visual e interactuar con el sistema mediante botones para analizar o limpiar el código.

Por su parte, el backend fue implementado en **Python** utilizando el framework **FastAPI**, seleccionado por su eficiencia, asincronía y fácil integración con herramientas modernas. La parte crítica del sistema se basa en **ANTLR4**, que permite generar automáticamente un parser y un visitor a partir de una gramática definida. Este visitor personalizado es el encargado de recorrer el árbol sintáctico generado por ANTLR para validar y evaluar la expresión.

La comunicación entre ambos componentes se realiza mediante solicitudes **HTTP (POST)** usando **Axios**, permitiendo enviar el código fuente desde el frontend y recibir de vuelta el análisis léxico, el análisis sintáctico, los posibles errores y el resultado evaluado de la expresión.

Esta arquitectura no solo facilita la separación de preocupaciones, sino que también permite futuras mejoras modulares. Por ejemplo, podría extenderse el backend con persistencia de variables, autenticación de usuarios o incluso generación de código intermedio, sin necesidad de reescribir el frontend. Todo esto hace que el sistema sea robusto, modular y fácilmente ampliable.

## 5. Diseño e Implementación

Para el desarrollo del analizador sintáctico, se definió una gramática .g4 compatible con ANTLR4, la cual contempla asignaciones, expresiones aritméticas y agrupaciones con los símbolos específicos del Grupo 09. Esta gramática admite operaciones básicas (+, -, \*, /, ^), uso de variables, números enteros y reales, así como el uso obligatorio de agrupadores () y el operador de asignación =>.

El archivo .g4 se compone de dos grandes bloques: el **lexer**, que define los tokens como números, identificadores, operadores, espacios, etc.; y el **parser**, que define las reglas gramaticales para expresiones, agrupaciones, potencia y asignaciones. Se utilizó la estrategia LL(\*) que ANTLR ofrece por defecto para procesar este tipo de gramáticas.

Una vez generados los archivos de parser y visitor, se implementó un **visitor personalizado en Python**, encargado de recorrer el árbol sintáctico (parse tree) y realizar la evaluación de las expresiones. Este visitor mantiene un diccionario de variables declaradas para permitir su reutilización dentro de otras expresiones. Además, se incluye la detección de errores como **división por cero**, uso de variables no declaradas y errores de agrupación.

A nivel de backend, el proyecto está estructurado modularmente, separando componentes por responsabilidad: análisis léxico, análisis sintáctico, evaluación, manejo de errores y controladores de entrada/salida. Esto facilita tanto el mantenimiento del código como la incorporación de nuevas funcionalidades.

En el frontend, se emplearon formularios reactivos y componentes dinámicos que permiten mostrar los resultados de forma amigable y clara: lista de tokens, errores encontrados, resultados evaluados, e incluso

## 6. Pruebas y Validación

Para validar el correcto funcionamiento del sistema, se llevaron a cabo **pruebas funcionales** diseñadas para cubrir los principales escenarios posibles: expresiones válidas, expresiones con errores matemáticos, y expresiones con errores sintácticos.

Las pruebas se realizaron de forma manual desde la interfaz web, ingresando código en el área correspondiente y observando los resultados retornados por el backend. A continuación, se presentan tres de las pruebas más representativas:

1.  $A \Rightarrow (5+3);$

Resultado esperado:  $A = 8$

Resultado obtenido:   $A = 8$

Validación: Se confirmó que el sistema reconoce correctamente la asignación, analiza la sintaxis, evalúa la expresión y guarda el valor de la variable.

2.  $B \Rightarrow (2^3);$


Resultado esperado:  $B = 8$

Resultado obtenido:   $B = 8$

Validación: La operación de potencia fue procesada correctamente. Se confirma que el evaluador interpreta correctamente la jerarquía de operadores.

3.  $X \Rightarrow (10/(5-5));$

Resultado esperado: Error: división por cero

Resultado obtenido:  Error detectado y reportado con mensaje claro

Validación: El sistema maneja errores de ejecución de forma segura, evita el fallo del backend y reporta el error de forma comprensible para el usuario.



## 7. Manual de Usuario

1. Ingresar a la aplicación web.
2. Escribir el código fuente con asignaciones y expresiones.
3. Seleccionar la opción de analizar.
4. Visualizar tokens, errores y resultados en pantalla.
5. Modificar el código si se detectan errores.
6. Interpretar los resultados para fines académicos.

## 7. Conclusiones

La implementación del analizador sintáctico utilizando **ANTLR4**, en conjunto con **FastAPI** y **ReactJS**, permitió construir una solución completa, modular y funcional para validar expresiones algebraicas e interpretar su resultado de manera clara y estructurada. Se logró integrar de forma efectiva distintas tecnologías modernas, aplicando los conceptos aprendidos durante el curso de Compiladores en un entorno práctico y real.

El proyecto no solo permitió consolidar conocimientos sobre gramáticas, análisis léxico y sintáctico, sino también mejorar habilidades técnicas como el trabajo con parsers, generación de árboles de análisis, manejo de errores y diseño de sistemas distribuidos. El uso de visitantes personalizados en ANTLR facilitó la evaluación de expresiones, mientras que el frontend en React mejoró significativamente la experiencia del usuario.

Durante el desarrollo se presentaron desafíos como el manejo de errores sintácticos precisos, la interpretación jerárquica de expresiones y la implementación modular del evaluador. Sin embargo, estos obstáculos fueron resueltos mediante investigación, pruebas iterativas y revisión colaborativa del código.

Como valor agregado, el proyecto sienta una base sólida para futuras extensiones, tales como la **generación de código intermedio**, **soporte para nuevas instrucciones semánticas**, **persistencia de variables**, e incluso **autenticación de usuarios**. En definitiva, este trabajo representa una experiencia formativa significativa en el proceso de convertirse en ingeniero de software, integrando teoría con práctica de forma coherente y aplicada.

## 8. Anexos

En esta sección se incluyen los anexos necesarios para complementar la documentación técnica del proyecto. Los archivos aquí presentados respaldan el funcionamiento del sistema desde una perspectiva estructural (gramática), lógica (evaluador) y visual (interfaz de usuario y resultados).

### **Anexo A – Gramática ANTLR**

Este anexo contiene el archivo .g4 que define la gramática del lenguaje aceptado por el compilador. Incluye las reglas léxicas y sintácticas necesarias para reconocer identificadores, números, operadores aritméticos, agrupadores {} y el símbolo de asignación =, específicos del Grupo 09. La gramática está diseñada para aceptar expresiones algebraicas válidas con precedencia de operadores y uso de variables.

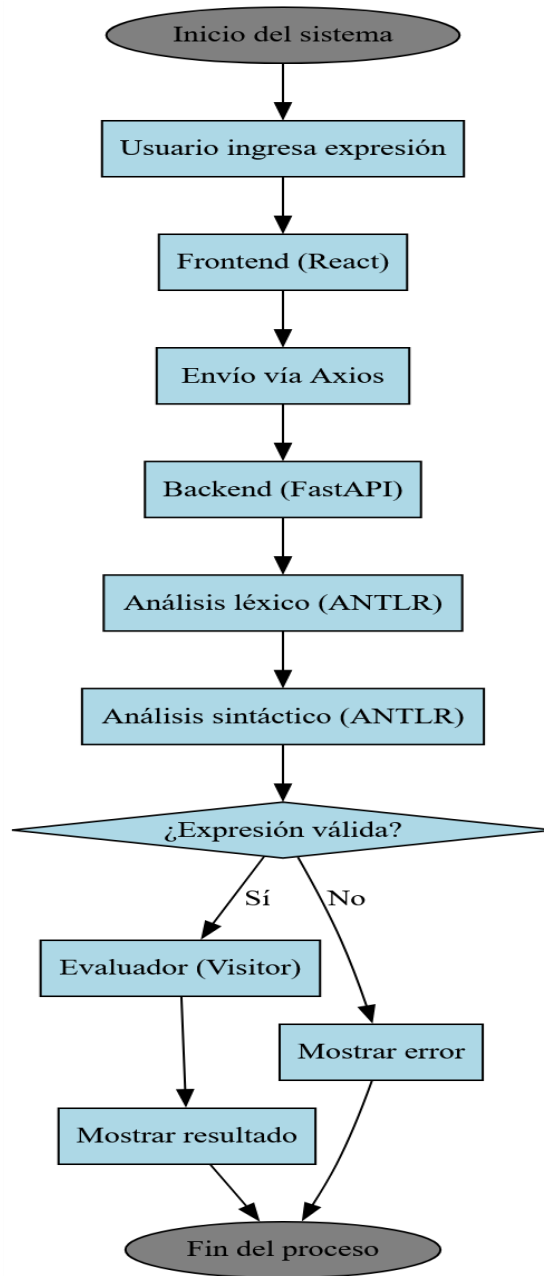
### **Anexo B – Visitor de Evaluación**

Aquí se encuentra el código fuente del visitor personalizado en Python, generado a partir del parser de ANTLR. Este visitor recorre el árbol sintáctico y evalúa las expresiones algebraicas, tomando en cuenta jerarquía de operadores y valores previamente asignados a variables. También incluye validaciones como detección de división por cero y uso de variables no definidas.

### **Anexo C – Capturas del Frontend y Ejemplos**

Este anexo presenta capturas de pantalla de la interfaz gráfica del sistema, mostrando ejemplos reales de uso. Incluye: ingreso de código fuente, visualización de tokens generados, errores sintácticos reportados por ANTLR, y el resultado evaluado de las expresiones válidas. También se muestran casos de error con retroalimentación clara para el usuario.

Estos anexos sirven para ilustrar de forma visual y técnica la funcionalidad desarrollada, y permitir una revisión más detallada de la implementación del proyecto.



## 9. Referencias

- ANTLR v4 Documentation - <https://github.com/antlr/antlr4>
- FastAPI Documentation - <https://fastapi.tiangolo.com>
- ReactJS Official - <https://react.dev>
- Documentación oficial del curso de Compiladores, 2025