# TP1 - Técnicas de Compilación

Este trabajo práctico implementa un compilador simplificado para un subconjunto de C, al que llamamos Cmini, utilizando ANTLR4 y Java (Maven). El objetivo del TP1 es definir la gramática léxica y sintáctica que permita reconocer programas básicos con funciones, sentencias de control, expresiones y operaciones aritméticas/lógicas. El flujo de funcionamiento es el siguiente: 1. Definimos una gramática en ANTLR (`Cmini.g4`). 2. Ejecutamos mvn generate-sources para generar automáticamente el lexer y parser en Java. 3. Escribimos una clase principal (`App.java`) que utiliza el parser generado para leer un programa y construir el árbol sintáctico. 4. Probamos con programas de ejemplo (`samples/*.c`).

## Archivos desarrollados por nosotros:

- `src/main/antlr4/tc/grammar/Cmini.g4` → gramática léxica y sintáctica.
- `src/main/java/tc/App.java` → clase principal que carga el archivo, invoca el parser y muestra el árbol.
- `src/main/java/tc/errors/BaseErrorListener.java` → listener personalizado para errores. - `samples/*.c` → programas de prueba escritos en Cmini.
- `README.md`, `.gitignore`, `pom.xml` → documentación y configuración.

## Archivos generados automáticamente por ANTLR:

- `CminiLexer.java`, `CminiParser.java`, `CminiBaseVisitor.java`, etc. → creados en `target/generated-sources/antlr4`.
- Archivos auxiliares como `*.tokens`, `*.interp`.
- Todo el contenido compilado en `target/`.

Estos archivos generados no se versionan en Git, porque siempre pueden regenerarse.

## Funcionamiento:

1. El usuario escribe un programa en Cmini (ej: `samples/ok_func.c`).
2. ANTLR analiza el texto con el lexer (convierte caracteres en tokens).
3. El parser usa la gramática para construir el árbol sintáctico.
4. `App.java` imprime el árbol en consola.
5. Si hay errores léxicos o sintácticos, los captura nuestro `BaseErrorListener`.

De esta forma, el TP1 cumple con el objetivo de validar la sintaxis de un programa escrito en Cmini.

## Teoría esencial de ANTLR:

- Lexer: convierte el código fuente en una secuencia de tokens (palabras clave, identificadores, números, símbolos).
- Parser: aplica las reglas de la gramática para formar estructuras sintácticas válidas.
- Árbol sintáctico (Parse Tree): representación jerárquica de cómo se interpreta el código.

- Listeners y Visitors: permiten recorrer el árbol y realizar acciones adicionales (validación, generación de código).
- Errores: por defecto ANTLR imprime mensajes, pero se personaliza con un ErrorListener propio.

## Conclusión:

El TP1 permitió sentar las bases del compilador: una gramática definida en ANTLR y un analizador sintáctico capaz de validar programas en Cmini. En próximos pasos (TP2 y Final) se extenderá para agregar tabla de símbolos, chequeos semánticos y generación de código intermedio. De esta forma, se completa la primera etapa del compilador y se prepara el camino para la defensa del proyecto.


# Integrantes:

- Maximiliano Eric Gonzalez
-Sergio Agustin Lopez