Taller lenguajes y gramáticas

Integrantes:
José Manuel Guerrero López
Sebastián Vargas Casquete
Andrés Loreto Quiros
Santiago Vargas Alegria

Pontificia Universidad Javeriana

Materia: Teoría de la Computación
Profesor: Jorge Valenzuela Buitrago

30/08/2025
Bogotá, D. C.

1. Introducción

El estudio de los lenguajes formales constituye un pilar fundamental en la teoria de la computación, pues permite comprender cómo se construyen compiladores e intérpretes a partir de modelos matemáticos bien definidos. Este taller tiene como propósito llevar a la teoría de la práctica mediante el diseño e implementación de un lenguaje de programación sencillo, definido en español y con un paradigma imperativo. A lo largo del documento se describen las motivaciones, objetivos, diseño del lenguaje, las gramáticas empleadas, así como los detalles de la implementación del analizador e interprete. Finalmente se presentan programas de prueba, los resultados esperados y conclusiones del proceso.

---

2. Motivación

La teoría de lenguajes formales provee la base para entender cómo funcionan los compiladores e intérpretes. A través de este ejercicio práctico se busca reforzar estos conceptos, pasando de las definiciones teóricas a una implementación funcional que permita reconocer, analizar y ejecutar código en un lenguaje diseñado por el grupo. El diseño de un lenguaje propio facilita la comprensión de temas como el análisis léxico, el análisis sintáctico y la semántica de un programa. Además, promueve la capacidad de abstraer reglas gramaticales, implementar analizadores con herramientas modernas como ANTLR4 y construir interpretes que materializan dichas abstracciones en un entorno de ejecución real.

---

3. Objetivos de aprendizaje

- Comprender la relación entre teoría de lenguajes y el diseño de lenguajes de programación.
- Definir un lenguaje imperativo sencillo en español, con soporte para expresiones, sentencias y funciones.
- Construir las gramáticas regulares y libres de contexto que lo describen.
- Implementar un analizador léxico, sintáctico y semántico utilizando ANTLR4.
- Desarrollar un intérprete en Java para ejecutar programas escritos en el lenguaje diseñado.

---

4. Diseño del lenguaje

4.1 Características principales

- **Tipos de datos:** enteros, decimales, booleanos, cadenas de texto.
- **Estructuras de control:** condicionales (si, sino), bucles (para, mientras).
- **Operadores:** aritméticos (+, -, *, /, %), de potencia, lógicos y relacionales.
- **Identificadores:** hasta 10 caracteres, comienzan con letra o guion bajo, seguidos de letras, dígitos o guiones bajos.
- **Palabras reservadas:** si, sino, para, mientras, imprimir, funcion, etc.

4.2 Limitaciones

- No se permite conversión automática entre tipos.

- Los programas deben finalizar con punto y coma ; en cada sentencia.
- El lenguaje es interpretado y no cuenta con optimizaciones avanzadas.

---

5. Gramática propuesta (archivo .g4)

```
grammar MiLenguaje;

// -------- Raíz --------
program
  : (declaracion | statement)* EOF
  ;

// -------- Declaraciones --------
declaracion
  : funcionDecl
  ;

funcionDecl
  : FUNCION IDENT '(' parametros? ')' bloque
  ;

parametros
  : IDENT (',' IDENT)*
  ;

// -------- Sentencias --------
statement
  : asignacion ';'
  | imprimir ';'
  | siStmt
  | mientrasStmt
  | paraStmt
  | retornarStmt ';'
  | bloque
  ;

asignacion
  : IDENT '=' expr
  ;
```

```
imprimir
  : IMPRIMIR '(' expr? ')'
  ;

siStmt
  : SI '(' expr ')' bloque (SINO bloque)?
  ;

mientrasStmt
  : MIENTRAS '(' expr ')' bloque
  ;

// Forma tipo C para simplificar el intérprete:
paraStmt
  : PARA '(' asignacion ';' expr ';' asignacion ')' bloque
  ;

retornarStmt
  : RETORNAR expr?
  ;

bloque
  : '{' statement* '}'
  ;
```

```
61    // -------- Expresiones (precedencia y asociatividad) --------
62    // OR lógico
63    expr
64      : expr OR expr                       # orExpr
65      | expr AND expr                      # andExpr
66      | expr (IGUAL | DIF) expr            # eqExpr
67      | expr (MENOR | MENORIG | MAYOR | MAYORIG) expr # relExpr
68      | expr (MAS | MENOS) expr            # addExpr
69      | expr (POR | DIV | MOD) expr        # mulExpr
70      | (MENOS | NO | MAS) expr            # unaryExpr     // -x, !x, +x
71      | expr POW expr                      # powExpr       // asociativa a la derecha (se maneja en visitor)
72      | prim                               # primaryExpr
73      ;
74
75    prim
76      : literal
77      | IDENT
78      | '(' expr ')'
79      ;
80
81    literal
82      : ENTERO
83      | DECIMAL
84      | VERDADERO
85      | FALSO
86      | CADENA
87      ;
88
```

```
89      // -------- Palabras reservadas (tokens de palabras) --------
90      FUNCION   : 'funcion';
91      RETORNAR  : 'retornar';
92      IMPRIMIR  : 'imprimir';
93      SI        : 'si';
94      SINO      : 'sino';
95      MIENTRAS  : 'mientras';
96      PARA      : 'para';
97      VERDADERO : 'verdadero';
98      FALSO     : 'falso';
99
100     // -------- Operadores y signos --------
101     IGUAL     : '==';
102     DIF       : '!=';
103     MENORIG   : '<=';
104     MAYORIG   : '>=';
105     MENOR     : '<';
106     MAYOR     : '>';
107
108     MAS       : '+';
109     MENOS     : '-';
110     POR       : '*';
111     DIV       : '/';
112     MOD       : '%';
113     POW       : '^';
114     AND       : '&&';
115     OR        : '||';
116     NO        : '!';
117
```

```
118     // -------- Identificadores y literales --------
119     // Identificadores: 1 a 10 caracteres: letra o '_' seguido de letras, dígitos o '_'
120     IDENT
121       : [a-zA-Z_][a-zA-Z0-9_]*
122         {getText().length() <= 10}?   // acepta solo si longitud <= 10
123       ;
124
125     // Números
126     DECIMAL   : [0-9]+ '.' [0-9]+;
127     ENTERO    : [0-9]+;
128
129     // Cadenas
130     CADENA    : '"' (~["\\] | '\\' .)* '"' ;
131
132     // -------- Espacios y comentarios --------
133     WS        : [ \t\r\n]+ -> skip;
134     LINE_COMMENT
135               : '//' ~[\r\n]* -> skip;
136     BLOCK_COMMENT
137               : '/*' .*? '*/' -> skip;
```

6. Implementación

6.1 Analizador léxico y sintáctico

Generado con ANTLR4 a partir del archivo. G4 (MiLenguaje.g4) Esta herramienta nos permitió crear el lexer y el parser.

6.2 Evaluador semántico (Visitor en Java)

El evaluador semántico se implementa en Java siguiendo el patrón **Visitor**, donde cada nodo del árbol sintáctico corresponde a una operación del lenguaje. Este módulo permite evaluar expresiones, verificar tipos y ejecutar instrucciones de control de flujo.

6.3 Intérprete

El intérprete se implementa como una clase principal que recibe el código fuente escrito en el lenguaje, invoca al parser, recorre el árbol sintáctico mediante el Visitor y finalmente ejecuta las instrucciones. También se manejan errores mediante **BaseErrorListener** para proporcionar mensajes comprensibles al usuario.

7. Programas de prueba

7.1 Área de un círculo

```
radio = 10;
area = 3 * radio * radio;
imprimir(area);
```

7.2 Sucesión de Fibonacci (50 primeros números)

```
n = 50; a = 0; b = 1; i = 0;
mientras (i < n) {
    imprimir(a);
    temp = a + b;
    a = b;
    b = temp;
    i = i + 1;
}
```

8. Resultados esperados

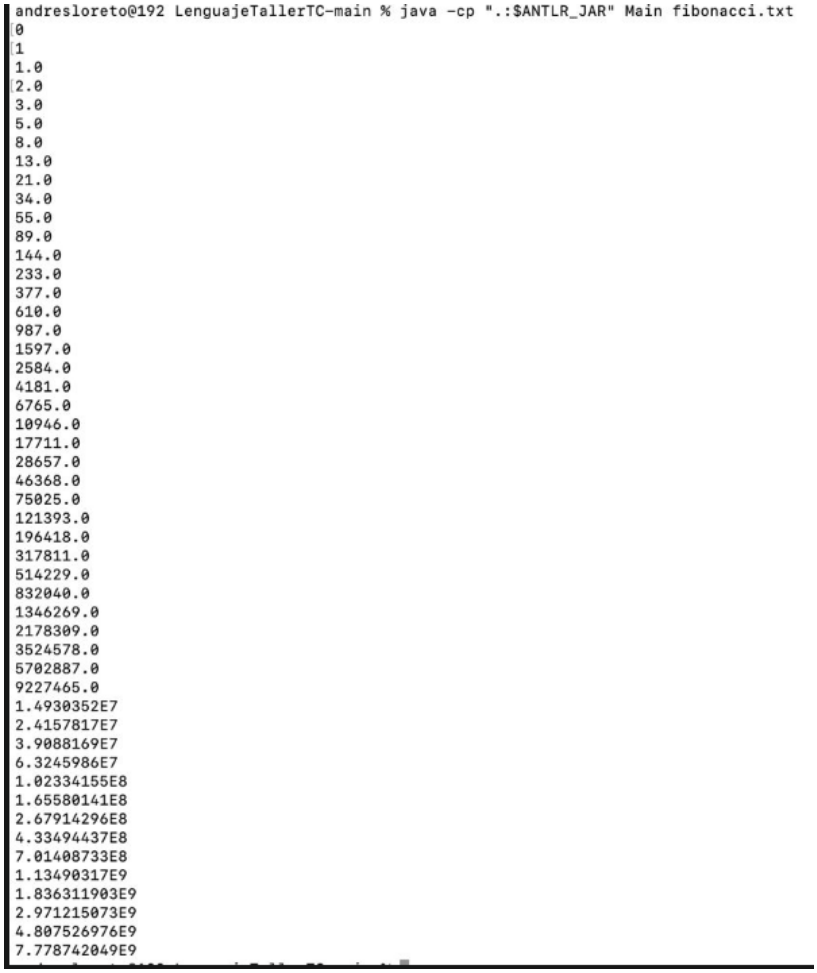
- El programa del círculo debe imprimir el área correcta para el radio definido.

  Resultado obtenido:

```
andresloreto@192 LenguajeTallerTC-main % java -cp ".:$ANTLR_JAR" Main area.txt
300.0
```

- El programa de Fibonacci debe imprimir la secuencia de los primeros 50 números.

  Resultado Obtenido:



---

9. Conclusiones

1. El diseño del lenguaje y su implementación con ANTLR4 nos permitió pasar de los conceptos teóricos de lenguajes regulares y libres de contexto a una aplicación concreta. Esto se reflejó en la construcción del lexer y el parser, donde fue evidente la relación con los autómatas finitos y de pila estudiados en clase.

2. El hecho de optar por un lenguaje imperativo, interpretado y en español nos obligó a tomar decisiones sobre tipos de datos, estructuras de control y operadores que garantizaran simplicidad y legibilidad. Estas restricciones redujeron ambigüedades y facilitaron la implementación inicial de la gramática.

3. Los programas de área del círculo y la sucesión de Fibonacci permitieron validar el recorrido completo, Con ellos comprobamos el manejo de expresiones, variables,

estructuras de control e impresión de resultados, lo que confirma que el lenguaje cumple con lo solicitado en el taller.

4. El taller reforzó la comprensión de cómo se construyen compiladores e intérpretes en la práctica, mostrando que detrás de cada lenguaje existe una base formal que guía su diseño. También fortaleció el trabajo en equipo, ya que la colaboración fue clave para completar cada fase del proyecto.