Universidad Rafael Landívar
Facultad de Ingeniería
Compiladores
Sección 01
Ing. Pedro David Gabriel Wong

PROYECTO DE APLICACIÓN
" ANALIZADOR SINTÁCTICO"

Jhonatan Velasquez Fuentes - 1137521
José Rodrigo Valle Riveiro - 1149123
Esly Merileydi Ajsivinac Cacatzí - 1308723
Melanie Hernández Prado -1111622

Guatemala, 21 de Mayo del 2025

ÍNDICE

ANÁLISIS

1. **Nombre del programa:**

   Compilador - Analizador Sintáctico.

2. **Objetivo General:**

Desarrollar la fase de análisis sintáctico de un compilados utilizando la herramienta ANTLR y los conceptos aprendidos en clase, permitiendo el ingreso de código fuente, la ejecución del análisis léxico y sintáctico con retroalimentación clara para el usuario, y la evaluación de expresiones algebraicas que involucran números y/o variables, con el fin de validar la estructura del lenguaje de entrada, preparar su traducción al lenguaje intermedio y calcular los resultados correspondientes,

3. **Alcance del programa.**

Este software consiste en un intérprete de expresiones algebraicas construido sobre ANTLR que permite:

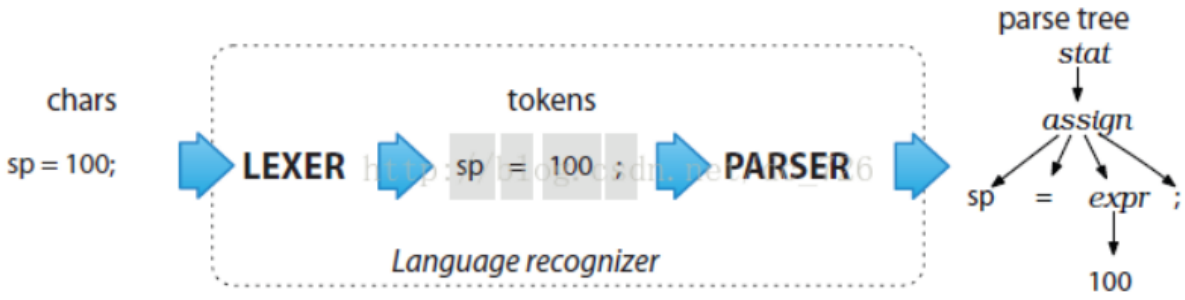
- Ingreso de código fuente con asignaciones y expresiones aritméticas.
- Análisis léxico y sintáctico con reportes de errores.
- Evaluación de expresiones (suma, resta, multiplicación, división, potencia) y asignación de variables.
- Impresión de los valores finales de las variables definidas.

ANTLR

Podemos interpretar a ANTLR como un analizador de sintaxis de código abierto que puede generar automáticamente un árbol de sintaxis basado en la entrada y posteriormente mostrarlo visualmente. Permite construir automáticamente lenguajes personalizados mediante descripciones gramaticales (analizador) y intérprete (traductor).

ANTLR se divide en dos etapas:

1. Etapa de análisis léxico - el programa llamado lexer, es el responsable de agrupar tokens en clase de token o tipo de token.
2. Etapa de análisis - en base al método léxico, se construye un árbol sintáctico.



(Entendiendo Antlr - Programador Clic, 2020)

Los principales escenarios de aplicación que ANTLR brinda son:

- Personalización de un idioma de dominio específico (DSL) la cual se usa para definir la gramática de alto nivel de la operación a realizar, lo cuál hace que la gramática sea más comprensible para humanos.
- Análisis de texto, lo cual permite analizar JSON, HTML, XML, EDIFACT o formatos de mensajes personalizados. También ayuda a definir qué hacer con la información analizada dentro del archivo de estructura.
- Cálculo matemático (utilizado para el proyecto) donde permite realizar suma, resta, multiplicación, división, ecuaciones lineales, operaciones geométricas, cálculo, etc.

ANÁLISIS DE REQUISITOS

4. Requisitos Funcionales.

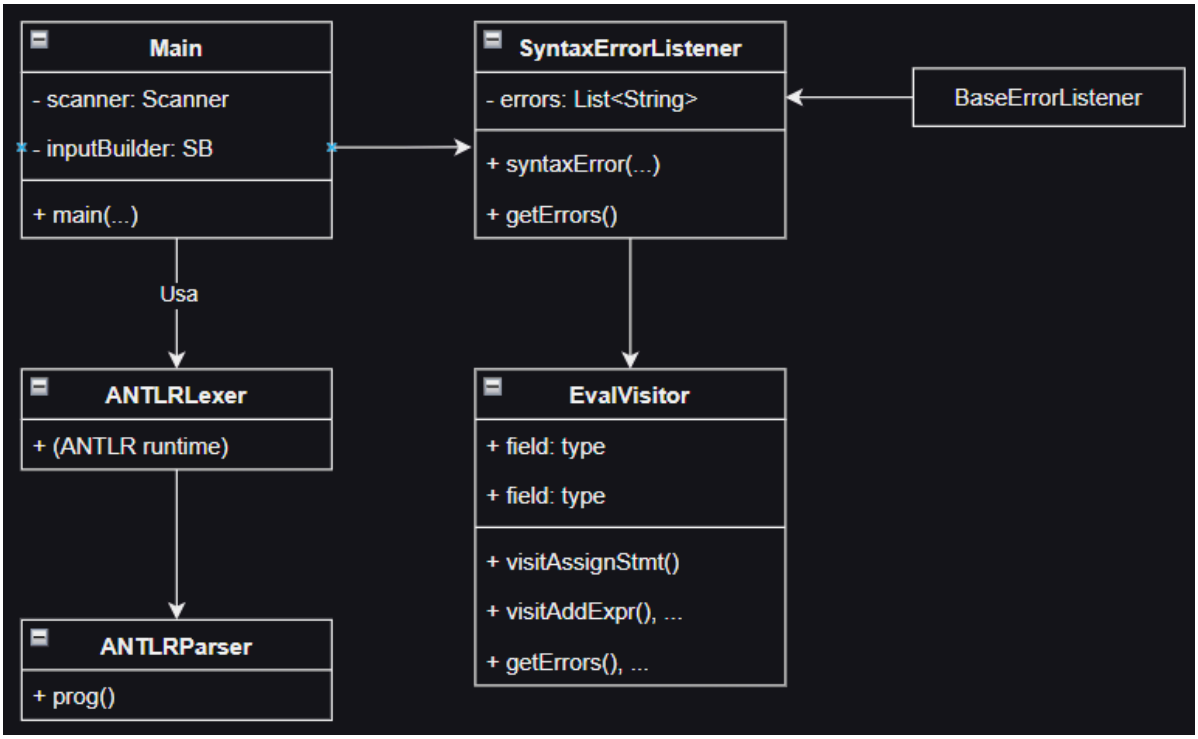
De forma lógica:

- Ingreso del código fuente: el usuario debe poder escribir varias líneas de código hasta indicar finalización (enter doble)
- Análisis léxico: Tokenización del código utilizando ANTLR.
- Análisis sintáctico: Validación de la gramática definida con el uso de () y ==.
- Manejo de errores:
  - Sintaxis inválida: error con línea, columna y mensaje.
  - División por cero: reporte específico.
  - Uso de variable no definida: reporte específico.
- Evaluación de expresiones: cálculo recursivo de expresiones algebraicas.

De forma visual:

- Ingreso de código fuente: el usuario ingresa o escribe varias líneas de código hasta indicar con el botón de analizar, el análisis del mismo.
- Visualización de resultados: dentro de la línea de resultados el programa mostrará el resultado de la operación analizada.
- Visualización de errores: dentro de la línea de errores el programa mostrará tanto los errores léxicos como los sintácticos.

5. Diagrama de Clases Principales.



6. Diseño de Pantalla.

7. Algoritmos.

Algoritmo 1: Visitor Recursivo,
Recorrido del algoritmo sintáctico (AST) para evaluar expresiones.

Algoritmo 2: Tokenización ANTLR.
División de la entrada en tokens según la gramática.

Algoritmo 3: Detección de errores.

- Método syntaxError, almacenamiento de errores de sintaxis.
- Validación en visitDivExpr y visitId, manejo de división por cero y variables no definidas.

Algoritmo 4: Evaluación Matemática.

- Uso de llamadas recursivas visit(expr): manejo de división por cero y variables no definidas.
- Operación de potencia con Math.pow.

8. Casos de Prueba.

Caso	Entrada	Salida Esperada
Asignación y suma	x == 2+3	x = 5.0
Operaciones mixtas	a == (2*3)-4/2	a = 4.0
Potencia y variables	b == 2^3; c == b+1	b = 8.0c = 9.0
División por cero	d == 5/0	Error División por cero en línea 1
Variable no definida	e == f+1	Error Variable no definida 'f'...
Agrupación con corchetes/braces	Según grupo: [2+3]*{4-1}	Resultado acorde y sin errores

DISEÑO Y ARQUITECTURA DEL CÓDIGO

La gramática definida en el archivo Expr.g4 establece las reglas léxicas y sintácticas necesarias para interpretar expresiones algebraicas con soporte de asignación, operaciones aritméticas y agrupación con distintos símbolos. Sus componentes principales son:

Reglas Léxicas (Tokens):

```
ID      : [a-zA-Z]+ ;             // Identificadores: variables con letras
NUM     : [0-9]+ ('.' [0-9]+)? ;  // Números enteros o decimales
PLUS    : '+' ;
MINUS   : '-' ;
MUL     : '*' ;
DIV     : '/' ;
POW     : '^' ;
EQ      : '==' ;
LPAREN  : '(' ;
RPAREN  : ')' ;
LBRACK  : '[' ;
RBRACK  : ']' ;
LBRACE  : '{' ;
RBRACE  : '}' ;
SEMI    : ';' ;
NEWLINE : [\r\n]+ ;               // Fin de línea
WS      : [ \t]+ -> skip ;        // Espacios en blanco ignorados
```

Estas reglas definen los componentes básicos del lenguaje, como operadores, paréntesis, números, variables y espacios en blanco.

Reglas Sintácticas:

```
prog: stat+ ;
stat: expr NEWLINE          # exprStmt
   | ID EQ expr (SEMI)?     # assignStmt ;
expr: expr POW expr         # powExpr
   | expr op=(MUL|DIV) expr # mulDivExpr
   | expr op=(PLUS|MINUS) expr # addSubExpr
   | group                  # groupedExpr
   | ID                     # idExpr
   | NUM                    # numExpr ;
group: LPAREN expr RPAREN
    | LBRACK expr RBRACK
    | LBRACE expr RBRACE ;
```

prog: Punto de entrada del programa. Consiste en una o más instrucciones.
stat: Instrucciones válidas, estas pueden ser:

- Evaluación de una expresión (exprStmt).
- Asignación de una variable (assignStmt) con el símbolo == y una expresión a la derecha.

expr: Reglas recursivas para evaluar expresiones aritméticas según la precedencia de operadores:

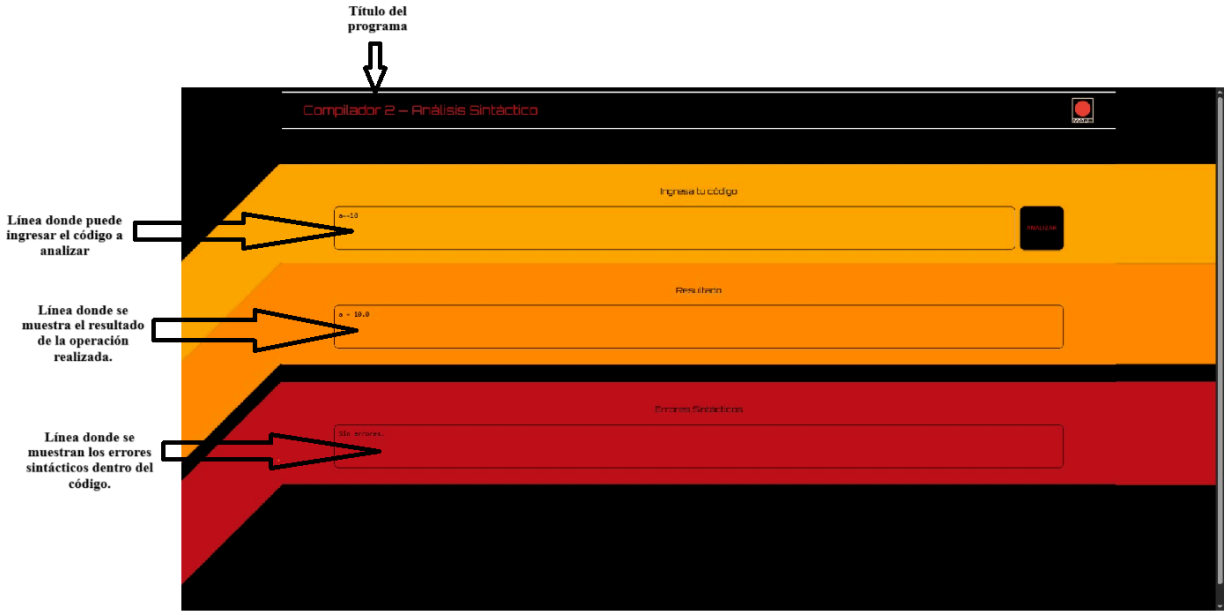
- Potencia (^) tiene la mayor prioridad.
- Multiplicación y división (*, /) siguen.
- Suma y resta (+, -) tienen la menor prioridad.
- Se permite el uso de paréntesis () como agrupadores.

MANUAL DE USUARIO

Estimado usuario, gracias por tomarse el tiempo en leer el siguiente documento, esto facilitará el proceso y la experiencia que tendrá con nuestro programa. Los requisitos que solicitamos para que el programa se ejecute de forma correcta son:

- Gramática funcional.

Al ingresar se le mostrará una ventana con las siguientes características donde debe ingresar el código que desea analizar y ejecutar:



Inicialmente ingresa la o las líneas de código que desea analizar, posteriormente presiona el botón de analizar ubicado a la derecha de la línea de ingreso. Posteriormente el programa analizará el código y por último mostrará en la línea de resultado, el resultado de la operación realizada y de contener algún error sintáctico o léxico se verá reflejado en la línea de errores.

CONCLUSIONES

Se determinó que ANTLR es una herramienta muy poderosa para realizar gramáticas y que las estructuras deben seguir una estructura específica para que pueda y entienda cómo construir el árbol de análisis del lenguaje asegurando robustez en el análisis del código fuente.

Se determinó que al definir de forma correcta la gramática, ANTLR puede generar automáticamente analizadores léxicos y sintácticos en varios lenguajes (Java, C#, Python, etc), lo que acelera el desarrollo de compiladores, intérpretes y herramientas de análisis.

Se comprobó que dividir el proyecto en componentes bien definidos como EvalVisitor, SyntaxErrorListener y Main promueve una arquitectura limpia que facilita la lectura, mantenimiento y extensión del programa.

La implementación de validaciones como la detección de división por cero o uso de variables no definidas mejora la experiencia del usuario, al evitar resultados inesperados y proporcionar mensajes de error claros y útiles.

Este intérprete puede escalar fácilmente hacia un lenguaje más completo, permitiendo agregar estructuras de control, funciones personalizadas o tipos de datos adicionales gracias a la base sólida proporcionada por ANTLR y la estructura modular del código.

REFERENCIAS

Entendiendo antlr - programador clic. (2020). Programmerclick.com. https://programmerclick.com/article/95881081580/