Laboratorio 8: Analizador Sintáctico Descendente

Diego Alejandro Rendón Suaza
Milton Alejandro Cuervo Ramírez

Materia: Teoría de Lenguajes
Profesor: Rubén Ángel Correa
Universidad de Antioquia
Programa: Ingeniería de Sistemas
2025-1

10 de junio de 2025

Resumen

This report details the design and development of a recursive descent parser for a basic arithmetic calculator. The main objective was to apply theoretical concepts of syntactic analysis to create a functional program capable of interpreting a string of characters, validating its grammatical structure, and if correct, calculating the numerical result. The implemented program can handle additions, subtractions, multiplications, divisions, and the use of parentheses, respecting standard operator precedence and hierarchy.

Índice

1. Introducción

El presente informe detalla el proceso de diseño y desarrollo de un analizador sintáctico descendente para una calculadora de operaciones aritméticas básicas. El objetivo principal es aplicar los conceptos teóricos de análisis sintáctico para crear un programa funcional capaz de interpretar una cadena de caracteres, validar su estructura gramatical y, en caso de ser correcta, calcular el resultado numérico correspondiente.

El programa implementado es capaz de manejar sumas, restas, multiplicaciones, divisiones y el uso de paréntesis para agrupar operaciones, respetando la jerarquía y precedencia estándar de los operadores.

2. Objetivos

2.1. Objetivo General

Elaborar un analizador sintáctico funcional que responda a la sintaxis de una calculadora simple, aplicando la técnica de análisis descendente.

2.2. Objetivos Específicos

- Implementar un analizador léxico (Lexer) para convertir la cadena de entrada en una secuencia de tokens.
- Desarrollar un analizador sintáctico (Parser) utilizando el método de descenso recursivo.
- Asegurar que el parser respete la precedencia de operadores.
- Proveer un sistema de reporte de errores de sintaxis claro y específico.
- Calcular y presentar el resultado final de las expresiones válidas.

—

3. Cuerpo de la Práctica

Esta sección detalla los aspectos técnicos y metodológicos de la implementación del analizador sintáctico descendente. Se cubrirán los componentes principales, las decisiones de diseño, los algoritmos utilizados y cualquier consideración relevante durante el desarrollo.

3.1. Marco Teórico

El análisis sintáctico es la segunda fase del proceso de compilación. Su función es verificar si la secuencia de tokens generada por el lexer se ajusta a las reglas gramaticales que definen el lenguaje.

3.1.1. Análisis por Descenso Recursivo

Es una de las formas más sencillas de implementar un analizador sintáctico. Consiste en tener un conjunto de funciones, una por cada no-terminal de la gramática. La ejecución del analizador comienza llamando a la función del símbolo inicial. Cada función es responsable de procesar la parte de la entrada que corresponde a su no-terminal.

3.1.2. Precedencia de Operadores

Para manejar correctamente expresiones como $10 + 2 \times 5$, el analizador debe saber que la multiplicación tiene mayor precedencia que la suma. En nuestro diseño, esto se logra separando las reglas gramaticales en diferentes niveles de funciones (‘expr‘, ‘term‘, ‘factor‘), donde cada nivel corresponde a una precedencia distinta.

3.2. Diseño e Implementación

El proyecto fue desarrollado en **Java** utilizando el IDE **IntelliJ IDEA**. La arquitectura se dividió en los siguientes componentes principales:

- **Clase `Token`:** Representa una unidad léxica con un tipo y un valor.
- **Clase `Lexer`:** Responsable del análisis léxico. Convierte la entrada en una lista de `Tokens`.
- **Clase `Parser`:** Es el núcleo del proyecto. Implementa el analizador por descenso recursivo para validar la sintaxis y calcular el resultado.
- **Clase `Main`:** Contiene el bucle principal que interactúa con el usuario.

A continuación se muestra un fragmento clave del `Parser` que gestiona la precedencia:

```
// Regla: expr ::= term ((PLUS | MINUS) term)*
private double expr() throws Exception {
    double result = term();
    while (currentToken.type == Token.TokenType.PLUS ||
           currentToken.type == Token.TokenType.MINUS) {
        // ... L gica de suma/resta
    }
    return result;
}

// Regla: term ::= factor ((MUL | DIV) factor)*
private double term() throws Exception {
    double result = factor();
    while (currentToken.type == Token.TokenType.MUL ||
           currentToken.type == Token.TokenType.DIV) {
        // ... L gica de multiplicaci n/divisi n
    }
    return result;
}
```

Listing 1: Funciones del Parser para manejar precedencia

3.3. Pruebas y Resultados

Se realizaron diversas pruebas para validar el correcto funcionamiento del analizador.

3.3.1. Casos de Éxito

Expresión de Entrada	Resultado Esperado	Resultado Obtenido
`10 + 2 * 6`	22	22.0
`(10 + 2) * 6`	72	72.0
`100 / (10 * 2) - 3`	2	2.0

Cuadro 1: Resultados de expresiones válidas.

3.3.2. Casos de Error

Expresión de Entrada	Mensaje de Error Obtenido
`5 * / 2`	Error de Sintaxis: Se esperaba un número o paréntesis
`10 + (3 * 2`	Error de Sintaxis: Se esperaba RPAREN pero se encontró EOF

Cuadro 2: Resultados de expresiones inválidas.

—

4. Conclusiones

El desarrollo de este proyecto permitió la aplicación práctica de los conceptos de análisis sintáctico descendente. Se cumplieron todos los objetivos, logrando un programa capaz de interpretar expresiones aritméticas, respetar la precedencia de operadores, calcular resultados y reportar errores de forma efectiva.

El método de descenso recursivo demostró ser una técnica intuitiva y potente para gramáticas simples. Como trabajo futuro, el analizador podría extenderse para soportar operadores unarios, variables y funciones matemáticas.