26 de febrero del 2025

Centro Universitario de Ciencias Exactas e Ingeniería

TRADUCTORES DE LENGUAJES ll

Practica 1

JULIO ESTEBAN VALDES LOPEZ

Juan Pablo Hernández Orozco

219294285

Objetivo:

Elaborar un analizador sintáctico descendente con la gramática vista en clase.

Introducción:

En el ámbito del procesamiento de lenguajes formales y el desarrollo de compiladores, el análisis sintáctico juega un papel fundamental. Un analizador sintáctico predictivo es un tipo de analizador recursivo descendente que, basándose en una gramática libre de contexto sin recursión por la izquierda, verifica si una secuencia de tokens cumple con la sintaxis esperada.

Este proyecto implementa un analizador sintáctico predictivo para expresiones aritméticas utilizando Python. La solución está dividida en tres componentes principales:

- Análisis léxico (Lexer): Se encarga de transformar una cadena de entrada en una lista de tokens, identificando números enteros, operadores matemáticos (+, -, *, /) y paréntesis.
- Análisis sintáctico (Parser): Utiliza la técnica de recursive descent parsing para validar la estructura de la expresión de acuerdo con una gramática bien definida.
- Interfaz gráfica (GUI) con Tkinter: Proporciona una forma interactiva para que el usuario ingrese una expresión matemática y reciba retroalimentación visual sobre su validez.

El objetivo de este proyecto es detectar errores de sintaxis en expresiones aritméticas y proporcionar retroalimentación clara al usuario. Además, esta implementación permite reforzar conceptos clave en el diseño de compiladores, como la tokenización, las reglas gramaticales y el análisis sintáctico recursivo.

Desarrollo:

1. Estructura del Analizador Sintáctico Predictivo
    a. El programa se divide en tres módulos principales:
        i. Lexer (Análisis Léxico): Convierte la expresión ingresada por el usuario en una lista de tokens.
        ii. Parser (Análisis Sintáctico): Verifica que la secuencia de tokens cumpla con la gramática de expresiones aritméticas.
        iii. Interfaz Gráfica (Tkinter): Permite la interacción con el usuario de manera visual.
    b. Análisis Léxico (Lexer)
        i. El lexer se encarga de escanear la cadena de entrada y convertirla en una lista de tokens. Para esto, se utilizan expresiones regulares que identifican los siguientes elementos:
            1. Números enteros (INT)
            2. Operadores aritméticos (PLUS, MINUS, MUL, DIV)
            3. Paréntesis (LPAREN, RPAREN)
            4. Espacios en blanco (SKIP, que se ignoran)
            5. Cualquier otro carácter no reconocido (MISMATCH, que genera un error)
            6. Cada token se representa mediante la clase Token, la cual almacena su tipo y valor (en el caso de los números). Finalmente, se agrega un token especial '$' para marcar el fin de entrada.

2. Diseño y Enfoque de la Solución

2.1 Tokenización (Lexer)

Para poder analizar la expresión, primero se debe transformar la cadena de entrada en una secuencia de tokens. Se utiliza el módulo re (expresiones regulares) para definir patrones y clasificar cada símbolo:

INT: Reconoce números enteros.

PLUS, MINUS, MUL, DIV: Reconocen los operadores +, -, * y /.

LPAREN y RPAREN: Reconocen los paréntesis abiertos y cerrados.

SKIP: Ignora espacios en blanco.

MISMATCH: Captura cualquier carácter no permitido y lanza un error.

El resultado de la tokenización es una lista de objetos Token, cada uno con un tipo (y valor en caso de números). Se agrega un token especial ($) para marcar el final de la entrada.

2.2 Análisis Sintáctico (Parser)

Se implementa un parser basado en la técnica de recursive descent, en la que cada regla de la gramática se traduce a un método en la clase Parser. La estructura es la siguiente:

E(): Llama a T() y luego a E'().

E'(): Si el token actual es PLUS o MINUS, consume el operador, llama a T() y vuelve a llamar a E(); en caso contrario, produce la producción vacía (ε).

T(): Llama a F() y luego a T'().

T'(): Si el token actual es MUL o DIV, consume el operador, llama a F() y se repite; en caso contrario, produce ε.

F(): Si el token actual es un paréntesis abierto, consume el paréntesis, llama a E() y luego consume el paréntesis cerrado. Si es un número (INT), simplemente lo consume. En otro caso, lanza un error.

El método eat() se encarga de verificar y consumir el token esperado. Si el token actual no coincide con lo esperado, se lanza un error de sintaxis.

2.3 Interfaz Gráfica con Tkinter

Para una interacción amigable con el usuario, se utiliza Tkinter. La interfaz consta de:

Un Entry para que el usuario ingrese la expresión.

Un Button que, al ser presionado, invoca la función parse_input().

Un Label para mostrar el resultado de la validación (expresión válida o mensaje de error).

La función parse_input() obtiene la cadena ingresada, la envía al lexer, luego al parser y finalmente actualiza el Label con el resultado.

El código se organiza de la siguiente manera:

Definición de la clase Token:

Permite almacenar y representar cada token con su tipo y, en el caso de números, su valor.

Función lexer(text):

Utiliza expresiones regulares para dividir la entrada en tokens. Se recorre la cadena, se identifican los diferentes tipos de tokens y se generan objetos Token. Al final se añade un token especial '$' para marcar el fin de la entrada.

Clase Parser:

Contiene métodos para cada no terminal de la gramática (E, E', T, T', F). Utiliza la técnica de recursive descent para consumir tokens y verificar que la expresión se ajusta a la gramática.

El método advance() actualiza el token actual.

El método eat() consume el token si coincide con lo esperado o lanza un error de sintaxis.

Los métodos correspondientes a las reglas gramaticales implementan la lógica de análisis.

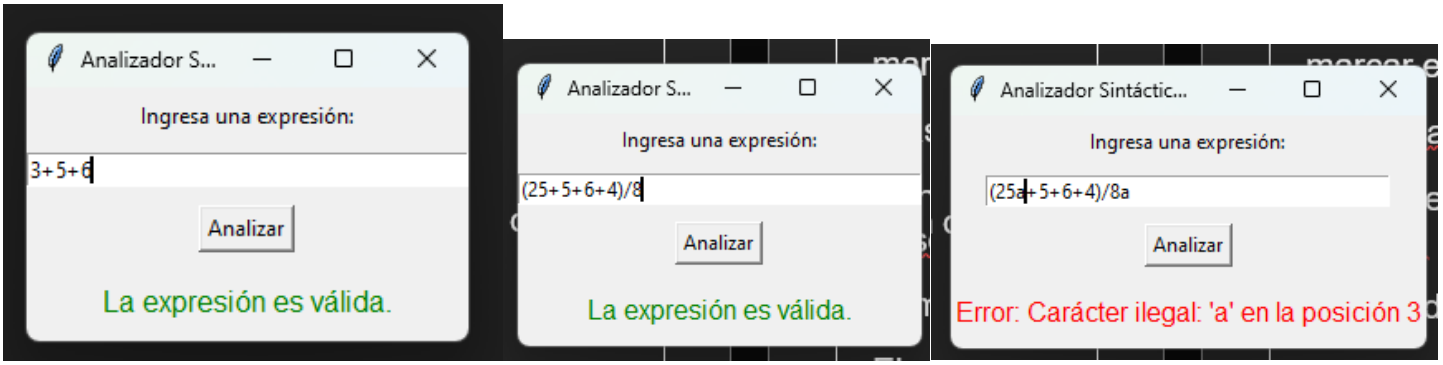
Función parse_input():

Es llamada al presionar el botón de la interfaz. Lee la entrada, invoca al lexer y parser, y actualiza el resultado en la interfaz.

Configuración de la interfaz con Tkinter:

Se crea la ventana principal, se configuran los widgets (Entry, Button, Label) y se inicia el loop principal de la aplicación.

https://github.com/ELJuanP/TRADUCTORES-DE-LENGUAJES-II/blob/main/Practica%201/main.py



Conclusión:

Esta solución integra conceptos fundamentales de análisis léxico y sintáctico para la validación de expresiones aritméticas mediante un analizador predictivo en Python. La combinación de un lexer basado en expresiones regulares, un parser recursivo y una interfaz gráfica con Tkinter permite:

- Transformar la entrada del usuario en una secuencia de tokens.
- Analizar la sintaxis de la expresión de acuerdo con una gramática definida.
- Proporcionar retroalimentación inmediata al usuario a través de una interfaz visual.

El enfoque modular y la separación de responsabilidades (tokenización, análisis y presentación) facilitan la comprensión, el mantenimiento y la posible extensión de la aplicación para soportar gramáticas más complejas o funcionalidades adicionales.