19 de marzo del 2025

Centro Universitario de Ciencias Exactas e Ingeniería

TRADUCTORES DE LENGUAJES ll

Practica 2

JULIO ESTEBAN VALDES LOPEZ

Juan Pablo Hernández Orozco

219294285

Objetivo:

Implementar un analizador sintáctico descendente con la gramática que reconoce expresiones aritméticas con 4 operaciones y parentesis.

Si la cadena pertenece al lenguaje se debe mostrar la misma expresión en notación postfija

Introducción:

En esta práctica se desarrolló una aplicación en Python que convierte expresiones aritméticas en notación infija a notación postfija. Se implementó un analizador descendente recursivo a través de la clase Parser para procesar y transformar las expresiones, y se integró una interfaz gráfica utilizando la librería Tkinter para facilitar la interacción del usuario.

El análisis sintáctico es una de las etapas fundamentales en el diseño de compiladores, ya que se encarga de verificar que la secuencia de tokens generada en el análisis léxico cumpla con la estructura gramatical del lenguaje. En esta práctica se utiliza un analizador sintáctico recursivo descendente basado en una gramática simple, la cual se expresa de la siguiente forma:

$E \rightarrow T \{ (+ \mid -) T \}$

$T \rightarrow F \{ (* \mid /) F \}$

$F \rightarrow (E) \mid \text{número}$

Esta gramática, presentada en la página 217 del libro *Compiladores: Principios, Técnicas y Herramientas*, es ideal para ilustrar los conceptos de análisis sintáctico descendente y de traducción orientada a la sintaxis. A lo largo del reporte se detalla cómo se aplican estos conceptos en la práctica, desde la tokenización y el análisis de la estructura sintáctica, hasta la presentación de resultados a través de una interfaz gráfica.

Desarrollo:

Analizador Descendente Recursivo:

La clase Parser se encarga de recorrer la cadena de entrada y reconocer los distintos componentes de la gramática aritmética.

- El método number() captura números compuestos por varios dígitos.

- factor() identifica números o sub-expresiones encerradas entre paréntesis, garantizando el correcto manejo de la prioridad de las operaciones.
- Los métodos term() y expr() se encargan, respectivamente, de procesar operaciones de multiplicación/división y suma/resta, concatenando los operandos y operadores en el orden adecuado para obtener la notación postfija.
  Este enfoque permite que, en lugar de detener la ejecución ante un error, se genere una estructura que facilita la gestión de fallos de manera controlada.

Interfaz Gráfica con Tkinter:
Se creó una ventana principal en la que el usuario puede ingresar una expresión aritmética en un campo de texto.

- Un botón vinculado a la función convert_expression() desencadena el proceso de análisis y conversión.
- El resultado en notación postfija se muestra en una etiqueta, proporcionando retroalimentación inmediata al usuario.
  Esta integración de la lógica de análisis con una interfaz visual hace la herramienta accesible y fácil de usar.

La práctica se organiza en tres componentes principales:

1. Análisis Sintáctico (Parser):

Se implementa utilizando la técnica de *recursive descent parsing*. Cada regla de la gramática se traduce en un método dentro de la clase `Parser`.

E → T { (+ | −) T }

T → F { (* | /) F }

F → (E) | número

Durante el análisis, se generan acciones semánticas que producen la expresión en notación postfija. Esto se realiza concatenando los operandos y colocando el operador al final, de manera similar a los esquemas de traducción explicados en el libro.

2. Implementación en Python:

Módulo Parser:

La clase `Parser` se encarga de recorrer la cadena de entrada, saltar espacios en blanco, reconocer números y paréntesis, y validar la estructura sintáctica conforme a la gramática definida.

Se incluyen métodos como `advance()`, `skip_whitespace()`, `number()`, `factor()`, `term()` y `expr()`, que permiten el análisis recursivo descendente.

Generación de Notación Postfija:

Durante el análisis se generan subexpresiones en notación postfija, lo que permite obtener una representación alternativa de la expresión original y sirve como base para futuros procesos de generación de código o evaluación.

3. Interfaz Gráfica (GUI) con Tkinter:

Se implementa una GUI sencilla que permite al usuario ingresar una expresión aritmética.

Un widget *Entry* recoge la expresión de entrada.

Un botón invoca la función que procesa la expresión.

Un *Label* muestra el resultado, ya sea la notación postfija generada o un mensaje de error en caso de sintaxis incorrecta.

3.2 Flujo de Ejecución

1. Entrada y Tokenización:

Al ingresar la expresión, el programa salta los espacios en blanco y reconoce números o paréntesis directamente, integrando de manera simple la funcionalidad de un lexer dentro del mismo módulo del parser.

2. Análisis Recursivo:

Se inicia el análisis mediante la función `expr()`, que llama a `term()` y, a su vez, a `factor()`.

Si se encuentra un paréntesis abierto, se llama recursivamente a `expr()` para procesar la subexpresión y luego se espera el paréntesis cerrado.

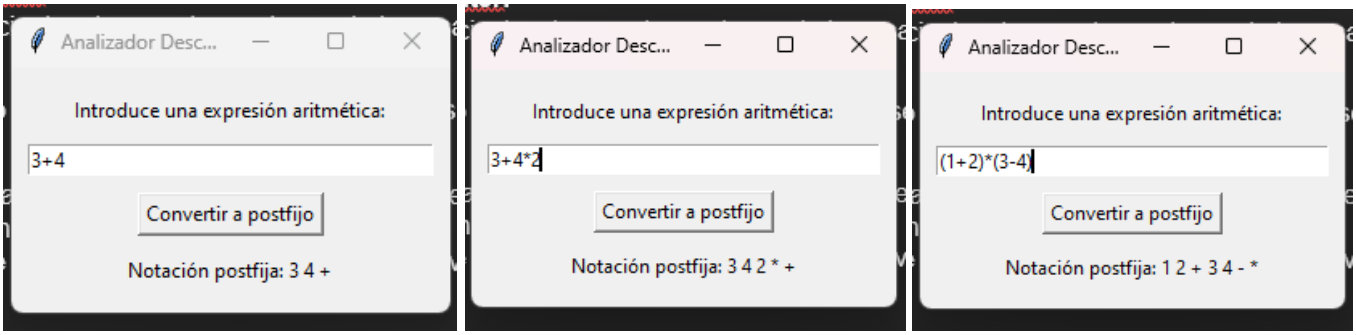
En el caso de números, se captura el lexema y se retorna su valor.

3. Generación de Salida:

Cada vez que se procesa un operador, se concatenan los resultados parciales para formar la notación postfija. Al finalizar, si no hay caracteres pendientes en la entrada, se muestra la notación postfija; de lo contrario, se notifica un error.

4. Interacción con el Usuario:

La función `convert_expression()` se encarga de leer la entrada, invocar el parser y actualizar la interfaz con el resultado.



https://github.com/ELJuanP/TRADUCTORES-DE-LENGUAJES-II/blob/main/Practica%202/main.py

Conclusión:

Esta práctica integra conceptos fundamentales del análisis sintáctico y de la traducción orientada a la sintaxis. La implementación del analizador sintáctico descendente en Python, basada en la gramática simple presentada en la página 217 del libro *Compiladores: Principios, Técnicas y Herramientas*, permite:

- Comprender y aplicar la técnica de *recursive descent parsing* para validar expresiones aritméticas.

- Generar una representación en notación postfija durante el proceso de análisis, lo cual es un ejemplo clásico de traducción semántica en compiladores.

- Desarrollar una solución interactiva mediante la integración de una interfaz gráfica con Tkinter, facilitando la experimentación y retroalimentación inmediata para el usuario.

El enfoque modular y la claridad en la separación de responsabilidades (análisis sintáctico y presentación) no solo refuerzan los conceptos teóricos aprendidos, sino que también sientan las bases para la extensión del proyecto a gramáticas más complejas o para la integración con otras fases de un compilador.