Procesadores de Lenguaje. Práctica 1

Juan Luis del Valle - 100454201
María Robledano - 100474933

Procesadores del Lenguaje
Marzo 2025
Curso 2024/2025

1. Introducción

El objetivo de esta práctica consiste en desarrollar un analizador léxico y un analizador sintáctico para interpretar expresiones matemáticas en notación polaca utilizando la librería PLY de python. El analizador léxico es responsable de dividir la entrada en tokens, mientras que el analizador sintáctico interpreta y evalúa las expresiones matemáticas.

Los principales objetivos de la práctica incluyen:

- Crear e implementar un analizador léxico que reconozca distintos tipos de entradas, como números, operadores y funciones matemáticas.
- Desarrollar un analizador sintáctico capaz de evaluar expresiones en notación polaca.
- Implementar una serie de pruebas para verificar el correcto funcionamiento del sistema.

2. Analizador Léxico

El analizador léxico (`lexer.py`) se ha implementado utilizando la librería `ply.lex`. Su función principal es reconocer y categorizar los diferentes tokens presentes en la entrada. Se han definido los siguientes tipos de tokens:

- **Números:*** enteros, reales, binarios y hexadecimales.*
- **Operadores:*** suma (+), resta (-), multiplicación (*), división (/).*
- **Funciones matemáticas:*** `sin`, `cos`, `log`, `exp`.*
- **Valores especiales:*** `inf`, `nan`.*
- **Memoria:*** la variable especial `MEMORY` para almacenar valores.*
- **Comentarios:*** de una línea (`#`) y multilínea (`''' ... '''`), que son ignorados por el lexer.*

Se han definido expresiones regulares para identificar cada tipo de token en el formato t_*token*(t) , y se han implementado manejadores para convertir las cadenas a valores numéricos en los casos necesarios, por ejemplo:

- Los valores hexadecimales en formato *0x[A-F0-0]* o binarios en formato *0b[01]* pasarlos a enteros
- Los valores reales, el infinito, o el nan pasarlos a flotantes
- Darle un operando a las expresiones aritméticas definidas en los tokens
- Darle valor negativo al token negativo
- Declarar los t_ignore y los t_error

3. Analizador Sintáctico

El analizador sintáctico (`parser.py`) se ha implementado utilizando `ply.yacc`. Su objetivo es interpretar y evaluar expresiones matemáticas en notación polaca, procesando los tokens generados por el analizador léxico y aplicando reglas gramaticales definidas.

Las reglas principales implementadas incluyen:

- **Operaciones matemáticas binarias:** suma, resta, multiplicación y división.
- **Funciones unitarias:** `exp`, `log`, `sin`, `cos`.
- **Manejo de valores especiales:** `inf`, `nan`.
- **Implementación de memoria (`MEMORY`)** permitiendo almacenar y recuperar valores.
- **Expresiones con operadores negativos:** reconocimiento del operador `neg` para invertir el signo de un número.

Para evitar errores, se ha definido una jerarquía de operadores y se han considerado casos especiales como la división por cero o el intento de calcular logaritmos de valores negativos. Además, el parser permite evaluar expresiones almacenadas en memoria, garantizando una ejecución correcta de las asignaciones y recuperaciones de valores.

4. Integración y Ejecución

El programa principal (`main.py`) coordina la ejecución del analizador léxico y sintáctico. Consiste en:

1. Leer el archivo `entrada.txt`.
2. Eliminar los comentarios de la entrada.
3. Procesar las expresiones una por una, evaluándose con el parser.
4. Mostrar resultados en la terminal.

Ejemplo de ejecución:

```
Unset
Entrada:

+ 3 5

log 10

MEMORY = 7

* MEMORY 2


Salida:

Resultado '+ 3 5' [Línea 1]: 8

Resultado 'log 10' [Línea 2]: 2.302

Resultado 'MEMORY = 7' [Línea 3]: 7

Resultado '* MEMORY 2' [Línea 4]: 14
```

¿Es necesario el uso de paréntesis en notación polaca?

No, en notación polaca no es necesario el uso de paréntesis, ya que el orden de evaluación se define por la ubicación de los operadores y operandos.

¿Se necesitan las mismas reglas de precedencia que en notación infija?¿Cómo serían para el caso de la notación postfija o polaca inversa?

No. En notación polaca, la jerarquía de operadores está implícita en la estructura de la expresión, por lo que no es necesario definir reglas de precedencia ya que van implícitas. En la notación postfija (o polaca inversa), los operadores aparecen después de los operandos, lo que también elimina la necesidad de paréntesis o reglas de precedencia.

¿Cuál es la notación más sencilla de implementar con una gramática?¿Y la más compleja?

La notación más sencilla de implementar con una gramática es la notación postfija (polaca inversa). Esto se debe a que su estructura permite evaluar expresiones de manera secuencial sin necesidad de definir reglas de precedencia ni manejar paréntesis. Por otro lado, la notación más compleja de implementar es la notación infija, ya que requiere reglas de precedencia de operadores y asociatividad para determinar el orden correcto de evaluación., ademas de la necesidad de gestionar el uso de paréntesis.

¿Por qué los paréntesis son necesarios en notación infija pero no en polaca?

En notación infija, los paréntesis son necesarios para evitar ambigüedades en el orden de evaluación de las operaciones. En notación polaca, el orden es explícito y no requiere paréntesis.

5. Conclusión

La práctica ha permitido comprender e implementar un analizador léxico y sintáctico en Python utilizando PLY, abordando distintos aspectos clave, como el reconocimiento de tokens, la evaluación de expresiones matemáticas y la implementación de memoria. También nos ha ayudado a comprender mejor el entorno de la asignatura, aún siendo a niveles sencillos.