



Tecnológico Nacional de México

Instituto Tecnológico de Pachuca

Tema:

Implementación del analizador sintáctico al léxico 5.1

Ingeniería en Sistemas Computacionales

7to Semestre Grupo: B

Materia: Lenguajes y Autómatas I

Profesor: Baume Lazcano Rodolfo

Equipo:

Martínez González José Pablo

Morales Ordoñez Yesenia

Lara Lopez Marco Antonio

Muñoz Castillo Ariana

SEMESTRE: Febrero – Junio 2024

30 – Mayo – 2024

INTRODUCCIÓN

El lenguaje de programación utilizado fue realizado en Python su función es procesar componentes básicos de una expresión matemática. El analizador utiliza la biblioteca "Ply" para la construcción de compiladores en este caso utilizando el **lex** para el léxico, y el **yacc** para el sintáctico.

DESCRIPCIÓN DEL LENGUAJE

El subconjunto de lenguaje analizado se compone de expresiones aritméticas básicas que incluyen números enteros, los operadores suma (+), resta (-), multiplicación (*), división (/). Las expresiones se pueden formar utilizando paréntesis para agrupar sub expresiones.

PROPÓSITO DEL ANALIZADOR SINTÁCTICO

Es ser el encargado de verificar la gramática utilizada en las entradas para que esta sea correcta de acuerdo a las reglas ya establecidas en nuestro lenguaje de programación para ello se tomaría en cuenta la serie de tokens que generamos en lo léxico ya que de no ser así se arrojaría un mensaje de error en la sintaxis.

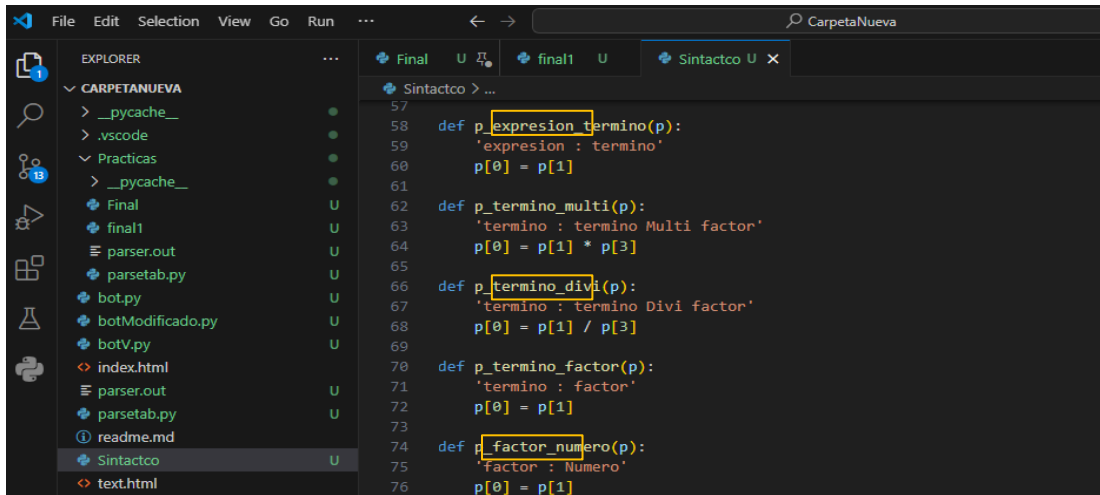
SÍMBOLOS TERMINALES

- | | |
|-------------------------|--|
| 1. Numero | Solo entero |
| 2. Suma | Símbolo característico de suma + |
| 3. Resta | Símbolo característico de resta - |
| 4. Multi | Símbolo característico de multiplicación * |
| 5. Divi | Símbolo característico de división / |
| 6. Delimitadores | Son (, [, { |

```
0
7  # Expresiones regulares para tokens
8  t_Suma = r'\+'
9  t_Resta = r'\-'
10 t_Multi = r'\*'
11 t_Divi = r'\/'
12 t_Igual = r'\='
13 t_Par = r'\('
14 t_Corchete = r'\['
15 t_Llave = r'\{'
```

SÍMBOLOS NO TERMINALES

- **expresion**: representa el total de entrada (toda la operación a evaluar)
- **termino**: representa una parte esencial de la operación (valor numérico)
- **factor**: representa el complemento requerido para ser evaluado (signo o determinantes)



DETERMINANTES

Los declarados como determinantes dentro de nuestro código son:

- Paréntesis sencillo abierto '('
- Corchete sencillo abierto '['
- Llave sencilla abierto '{'

PRODUCCIÓN Y REGLAS

Dentro de estas producciones y reglas determinamos de qué forma deberían establecerse el orden de las entras ya que si este orden no se lleva a cabo podría recaer a un error y estos errores se inclinan a lo que es la procedencia y la asociatividad ya que son punto clave que como analizador sintáctico identifica.

- **expresion** : termino Suma termino
- **expresion** : termino Resta termino
- **expresion**: conjunto de terminos
- **termino** : termino Multi factor
- **termino** : termino Divi factor
- **termino** : factor
- **factor** : Número

SÍMBOLOS INICIALES

El símbolo inicial de la gramática es 'expresión'

PROCEDENCIA Y ASOCIATIVIDAD

Los operadores determinan el orden en que se evalúan los operadores en una expresión. Los operadores con mayor precedencia se evalúan antes que los operadores con menor precedencia.

Multiplicación y división tienen una mayor precedencia que los operadores aditivos (suma y resta).

Procedencia

Operador de multiplicación ($*$) tiene precedencia alta

Operador de división ($/$) tiene precedencia alta

Operador de suma ($+$) tiene precedencia baja

Operador de resta ($-$) tiene precedencia baja

Asociatividad

Operador de multiplicación ($*$) asociativo por la izquierda

Operador de división ($/$) asociativo por la izquierda

Operador de suma ($+$) asociativo por la izquierda

Operador de resta ($-$) asociativo por la izquierda

COMENTARIOS Y ANOTACIONES

1. # Definición de tokens
2. # Expresiones regulares para tokens
3. # Expresión regular que reconoce números enteros
4. # Ignorar espacios en blanco y/o saltos de línea
5. # Manejar saltos de línea y actualizar el número de línea
6. # Manejar errores de análisis léxico
7. # Construir el analizador léxico
8. # Precedencia y asociatividad
9. # Reglas de producción para la gramática
10. # Construir el analizador sintáctico
11. # Prueba del analizador sintáctico con la entrada
12. # Función personalizada para imprimir los tokens
13. # Imprimir tokens
14. # Parsear cada línea individualmente