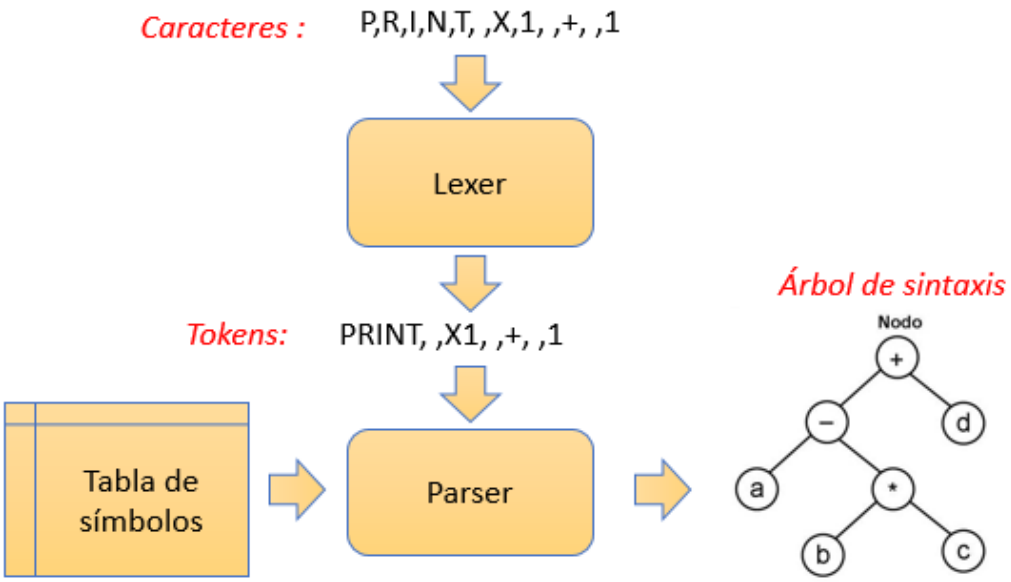
Alumno: Matuz Gómez Braulio Alanni

Materia: Seminario de Traductores de lenguajes 2

sección: D02 Código estudiante: 219293637

2024A

Título de la Tarea: Mini Generador Sintactico



Introducción:

Dentro de esta nueva adición que es el analizador sintáctico se estará descubriendo como contextualmente se debe tomar en cuenta en imaginación y estructura el cómo trabaja el analizador sintáctico y en el cómo tiene que interpretarse cada palabra o ingreso hacia el compilador, representados con algunas adjunciones como lo son las siguientes "< >" claro está que solo es como una representación documentada para saber cómo trabajan ciertas expresiones.

Investigación:

La función ***sintactic*** está programada de forma que cuando se detecte un token en el análisis léxico se manda a llamar dicha función y se le pasan ciertos parámetros.

j: es el id que tiene cada token por eso el switch está atenido a dicha variable.
k: es la linea de código donde se detectó el token.
linea: es el valor de cadena que tiene la linea que se está analizando.

El análisis del token lo realizamos gracias a las expresiones regulares.

Las ***regex*** (en inglés, regular expressions) son las unidades de descripción de los lenguajes regulares, que se incluyen en los denominados lenguajes formales. Son un instrumento clave de la informática teórica, la cual, entre otras cosas, establece las bases para el desarrollo y la ejecución de programas informáticos, así como para la construcción del compilador necesario para ello. Es por esto que las expresiones regulares, también denominadas regex y basadas en reglas sintácticas claramente definidas, se utilizan principalmente en el ámbito del desarrollo de software.

Ejemplos del código y la salida que se obtuvo:

primera parte del codigo mostrando nuevamente la parte lexica donde se tienen pequeñas modificaciones.

```python
import re

class AnalizadorLexico:
    def __init__(self, codigo_fuente):
        self.codigo_fuente = codigo_fuente
        self.tokens = []
        self.reservadas = {'if': 19, 'while': 20, 'return': 21, 'else': 22, 'int': 4, 'float': 4, 'void': 4}
        self.patron_identificador = re.compile(r'[a-zA-Z][a-zA-Z0-9]*')
        self.patron_entero = re.compile(r'\d+')
        self.patron_real = re.compile(r'\d+\.\d+')
        self.patron_operador = re.compile(r'[+\-*/=!<>]|&&|\|\|')
        self.patron_parentesis = re.compile(r'[()]')
        self.patron_llave = re.compile(r'[{}]')
        self.patron_punto_y_coma = re.compile(r';')
        self.patron_coma = re.compile(r',')
        self.patron_igual = re.compile(r'=')
        self.posicion = 0

    def analizar(self):
        while self.posicion < len(self.codigo_fuente):
            self.tokenizar()

    def tokenizar(self):
```

ahora como parte nueva que es el analizador sintáctico se muestra el uso de la representación de la salida del mismo léxico para la interpretación del sintáctico.

```python
class AnalizadorSintactico:
    def __init__(self, tokens):
        self.tokens = tokens
        self.posicion = 0

    def analizar(self):
        try:
            self.expresion()
            print("Análisis sintáctico exitoso.")
        except Exception as e:
            print(f"Error sintáctico: {e}")

    def expresion(self):
        self.termino()
        while self.posicion < len(self.tokens) and self.tokens[self.posicion][1] in ('+', '-'):
            self.match(('operador', '+') if self.tokens[self.posicion][1] == '+' else ('operador', '-'))
            self.termino()
```

y como salidas se muestra lo siguiente mostrando como minimo una funcionalidad que en un futuro puede ser cambiada

```
identificador           0
operador                =
entero          1
punto_y_coma            ;
identificador           4
identificador           0
operador                =
real            2
punto_y_coma            ;
identificador           19
parentesis              (
identificador           0
operador                >
entero          1
operador                &&
identificador           0
operador                <
real            2
parentesis              )
llave           {
identificador           21
identificador           0
operador                +
identificador           0
punto_y_coma            ;
llave           }
identificador           22
identificador           21
identificador           0
operador                *
identificador           0
punto_y_coma            ;
llave           }
llave           }
fin             23
```

```
PS C:\Users\USER\Downloads\Micro Analizador Lexico> &
ador Sintactico.py"
Error sintáctico: Error en el factor
PS C:\Users\USER\Downloads\Micro Analizador Lexico>
```