Sobre el funcionamiento del Analizador Léxico

El programa de análisis léxico que hemos desarrollado para un lenguaje de programación simplificado funciona siguiendo varios pasos clave para transformar el código fuente en una serie de tokens, que son las unidades básicas de significado dentro del lenguaje. Aquí te proporciono un resumen de cómo funciona este programa:

Definición de Tokens: El primer paso en el análisis léxico es definir los tokens que representan las unidades básicas del lenguaje, como palabras reservadas (e.g., if, else, while, print), identificadores, literales numéricos, operadores aritméticos (+, -, *, /), y símbolos de puntuación (;, (,), {, }). Esto se logra mediante expresiones regulares y una lista de palabras reservadas.

Análisis del Código Fuente: El analizador léxico, creado con la ayuda de PLY (una herramienta de análisis léxico y sintáctico para Python), lee el código fuente desde un archivo y lo procesa carácter por carácter. Utiliza las definiciones de tokens para clasificar segmentos del código fuente en tokens correspondientes.

Generación de Tokens: A medida que el código fuente es leído, el analizador léxico genera una secuencia de tokens. Cada token tiene un tipo que corresponde a su clasificación (por ejemplo, NUMBER, ID, IF) y, en algunos casos, un valor asociado (como el valor numérico para un token NUMBER o el nombre para un token ID).

Manejo de Errores Léxicos: Si el analizador encuentra caracteres o secuencias de caracteres que no puede clasificar dentro de las reglas definidas, se generan errores léxicos. El programa maneja estos errores informando al usuario sobre la presencia de caracteres ilegales y continúa el análisis ignorándolos.

Salida del Analizador: La salida final del analizador léxico es una lista de todos los tokens identificados en el código fuente. Esta lista es crucial para el siguiente paso en el proceso de traducción, el análisis sintáctico, donde la estructura gramatical del programa será analizada.

Este programa es esencialmente la primera fase en la construcción de un compilador o intérprete, permitiendo transformar el texto del código fuente en una forma que pueda ser más fácilmente analizada y procesada en pasos posteriores del proceso de traducción del lenguaje de programación.

Ejemplo de salida

LexToken(ID,'var',2,1)

LexToken(ID,'x',2,5)

LexToken(EQUALS,'=',2,7)

LexToken(NUMBER,5,2,9)

LexToken(SEMICOLON,';',2,10)

LexToken(ID,'var',3,12)

LexToken(ID,'y',3,16)

LexToken(EQUALS,'=',3,18)

LexToken(NUMBER,10,3,20)

LexToken(SEMICOLON,';',3,22)

LexToken(ID,'var',4,24)

LexToken(ID,'z',4,28)

LexToken(EQUALS,'=',4,30)

LexToken(ID,'x',4,32)

LexToken(PLUS,'+',4,34)

LexToken(ID,'y',4,36)

LexToken(SEMICOLON,';',4,37)

LexToken(PRINT,'print',6,40)

LexToken(LPAREN,'(',6,45)

LexToken(ID,'z',6,46)

LexToken(RPAREN,')',6,47)

LexToken(SEMICOLON,';',6,48)

LexToken(IF,'if',8,51)

LexToken(LPAREN,'(',8,54)

LexToken(ID,'z',8,55)

Carácter ilegal '>'

LexToken(NUMBER,10,8,59)

LexToken(RPAREN,')',8,61)

LexToken(LBRACE,'{',8,63)

LexToken(PRINT,'print',9,69)

LexToken(LPAREN,'(',9,74)

LexToken(ID,'z',9,75)

LexToken(RPAREN,')',9,76)

LexToken(SEMICOLON,';',9,77)

LexToken(RBRACE,'}',10,79)

LexToken(ELSE,'else',10,81)

LexToken(LBRACE,'{',10,86)

LexToken(PRINT,'print',11,92)

LexToken(LPAREN,'(',11,97)

LexToken(ID,'x',11,98)

LexToken(RPAREN,')',11,99)

LexToken(SEMICOLON,';',11,100)

LexToken(RBRACE,'}',12,102)

LexToken(WHILE,'while',14,105)

LexToken(LPAREN,'(',14,111)

LexToken(ID,'x',14,112)

Carácter ilegal '<'

LexToken(ID,'z',14,116)

LexToken(RPAREN,')',14,117)

LexToken(LBRACE,'{',14,119)

LexToken(PRINT,'print',15,125)

LexToken(LPAREN,'(',15,130)

LexToken(ID,'x',15,131)

LexToken(RPAREN,')',15,132)

LexToken(SEMICOLON,';',15,133)

LexToken(ID,'x',16,139)

LexToken(EQUALS,'=',16,141)

LexToken(ID,'x',16,143)

LexToken(PLUS,'+',16,145)

LexToken(NUMBER,1,16,147)

LexToken(SEMICOLON,';',16,148)

LexToken(RBRACE,'}',17,150)