Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

División de Tecnologías para la Integración Ciber-Humana

Departamento de Ciencias Computacionales

Práctica 1 Analizador léxico

Seminario de Solución de Problemas de Traductores de Lenguajes 2

Profesor: Luis Felipe Muñoz Mendoza

Sección: D09

Hernandez Rodriguez Diego Andres.

Juan Antonio Pérez Juárez.

Índice

Marco Teórico.....2
Desarrollo.....3
Referencias.....16

Marco Teórico

El análisis léxico es la primera fase de un compilador o intérprete, cuya función principal es leer el código fuente y transformarlo en una secuencia de tokens. Un token es la unidad mínima de significado dentro de un lenguaje de programación, como palabras clave, identificadores, operadores y símbolos especiales.

Esta práctica tiene como objetivo el desarrollo de un analizador léxico en Python, capaz de identificar distintos tokens en una cadena de entrada y clasificarlos en categorías predefinidas.

Desarrollo

El análisis léxico es el proceso mediante el cual una cadena de caracteres se agrupa en tokens que representan estructuras significativas del lenguaje. Para esto, el analizador utiliza expresiones regulares o algoritmos de búsqueda para identificar los tokens en la entrada.
El analizador léxico tiene las siguientes responsabilidades:

Leer el código fuente y dividirlo en componentes básicos.

- Clasificar cada componente en una categoría (token).
- Detectar errores léxicos en caso de encontrar caracteres no válidos.
- Generar una lista de tokens para su uso en etapas posteriores del procesamiento del código.

Un token es una **unidad mínima de significado** en el código fuente. En este analizador léxico, se han definido las siguientes categorías de tokens:

int, float, char, void, string	Tipo de dato (0)
Identificadores (nombres de variables y funciones)	Identificador (1)
Números enteros y decimales, constantes como # y pi	Constante (2)
;	Punto y coma (3)
,	Coma (4)
(	Paréntesis izquierdo (5)
)	Paréntesis derecho (6)
{	Llave izquierda (7)
}	Llave derecha (8)
=	Operador de asignación (9)
if	Condicional (9)
while	Bucle (10)
return	Retorno de función (11)
else	Condicional alternativo (12)
for	Bucle (13)
+, -	Operadores de adición (14)
*, /, <<, >>	Operadores de multiplicación (15)
&&, `	
<, >, >=, <=, ==, !=	Operadores relacionales (17)
$	Fin de entrada (18)
"..."	Cadenas de texto (19)

Todos y cada una de las instrucciones ingresadas serán evaluadas y clasificadas en cada una de las categorías para ser identificadas como un token. De esta manera se pueden identificar errores de manera más eficiente, porque si alguna parte no puede ser clasificada dentro de las categorías, esa es la parte del código (en esta caso de la instrucción) que presenta un error en su escritura.

Código fuente:

```
import re

# Definición de los tokens y sus categorías
tokens = [
    (r'\b(int|float|char|void|string)\b', 0),  # Tipo de dato
    (r'\b(if|while|return|else|for)\b', lambda m: {'if': 9, 'while':
10, 'return': 11, 'else': 12, 'for': 13}[m.group()]),
    (r'\b[a-zA-Z_][a-zA-Z0-9_]*\b', 1),  # Identificador
    (r'\b\d+(\.\d+)?\b', 2),  # Constante (números enteros y decimales)
    (r';', 3), (r',', 4), (r'\(', 5), (r'\)', 6), (r'\{', 7), (r'\}',
8), (r'=', 9),
    (r'\+|-', 14),  # Operadores de adición
    (r'\*|/|<<|>>', 15),  # Operadores de multiplicación
```

```
    (r'&&|\|\|', 16),  # Operadores lógicos
    (r'<|>|>=|<=|==|!=', 17),  # Operadores relacionales
    (r'\$', 18),  # Fin de entrada
    (r'".*?"', 19)  # Cadenas de texto
]

def lexer(input_string):
    token_counts = {i: 0 for i in range(20)}
    token_list = []
    pos = 0
    while pos < len(input_string):
        match = None
        for pattern, category in tokens:
            regex = re.compile(pattern)
            match = regex.match(input_string, pos)
            if match:
                token = match.group(0)
                category = category(match) if callable(category) else
category
                token_counts[category] += 1
                token_list.append((token, category))
                pos = match.end()
                break
        if not match:
            print(f"Error léxico en: {input_string[pos]}")
            pos += 1

    return token_list, token_counts

if __name__ == "__main__":
    user_input = input("Ingrese el código a analizar: ")
    tokens_encontrados, conteo_tokens = lexer(user_input)

    print("\nTokens encontrados:")
    for token, categoria in tokens_encontrados:
        print(f"{token}: {categoria}")

    print("\nResumen de categorías:")
    for categoria, cantidad in conteo_tokens.items():
        print(f"Categoría {categoria}: {cantidad}")
```

Este codigo lo que hace es analizar una cadena ingresada y comparar parte por parte hasta hallar alguna coincidencia con alguna de las 19 categorias que tiene para los tokens, en caso de no encontrar ninguna en el espacio revisado, lo manda como error léxico.

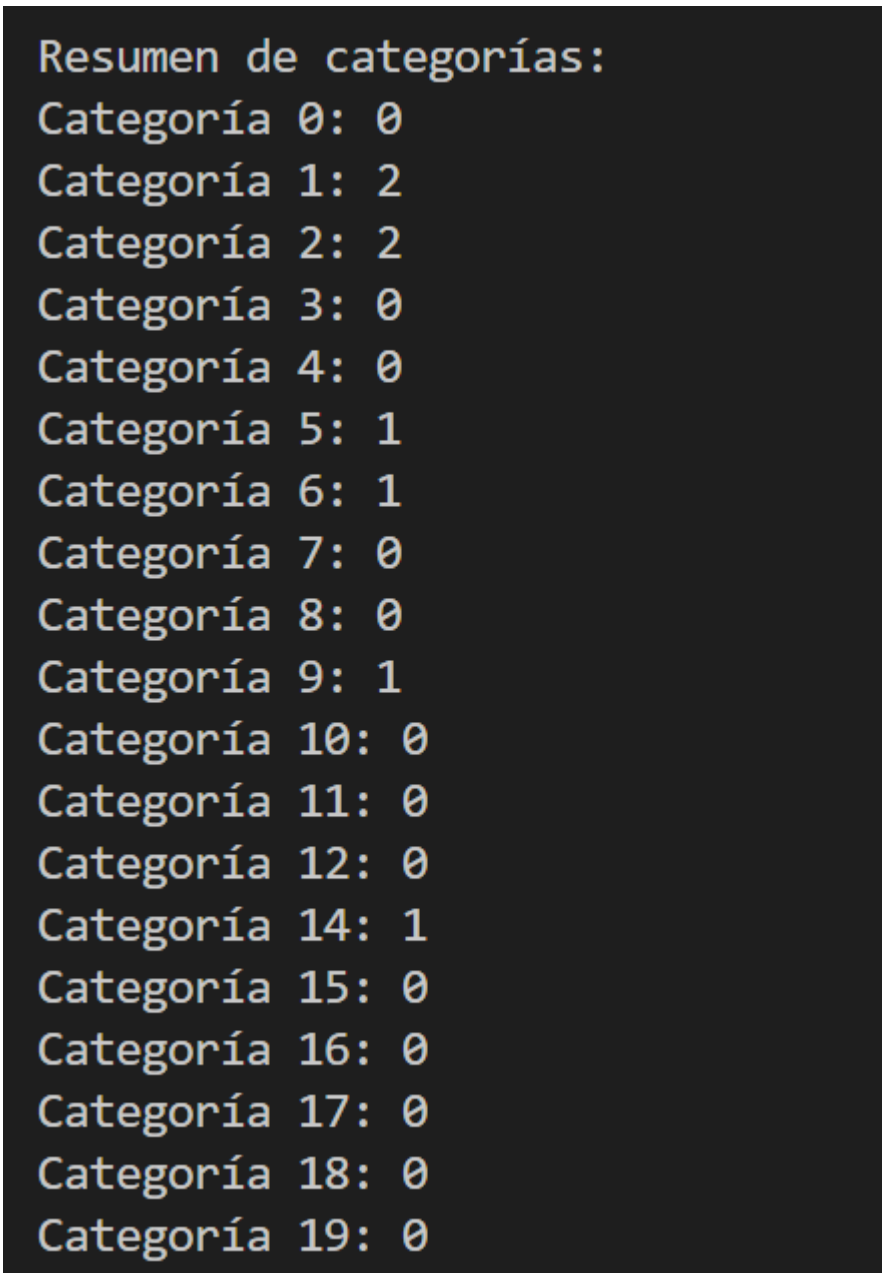
Algunos caracteres son fáciles de encontrar, por ejemplo, si se ingresa parentesis (ya sea de apertura o de cierre) se va a detectar este tipo de caracteres, si se ingresa algun tipo de dato, lo va a analizar y coincidir con alguno de los ya existentes.
Como tal, este código también podría recibir sentencias de c++ porque también biene añadido entre las opciones una forma de reconocer el punto y coma característico del final de las sentencias de c++, pero no viene implementado de la libreira de uso estandar los comandos de cout y cin caracteristicos del del lenguaje de c++.

Captura del código

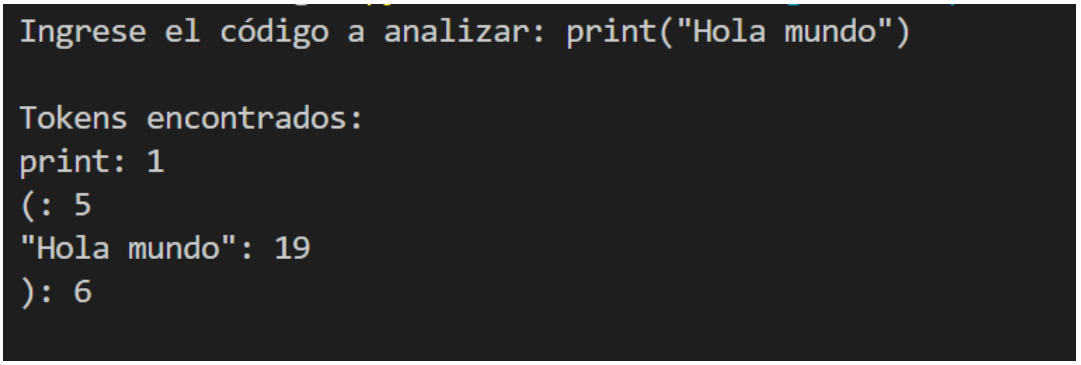
```
Ingrese el código a analizar: print(num=5+5)

Tokens encontrados:
print: 1
(: 5
num: 1
=: 9
5: 2
+: 14
5: 2
): 6
```

Esta parte de aquí lo que hace es que se ingresa la cadena de caracteres “print(num=5+5)” y en esta implementación no verifica que la expresión este bien formada, solo reconoce el contenido ingresado y va clasificando cada parte en el toque que le corresponde.



También menciona cada una de las categorías que fueron llamadas en esta simple instrucción.



En este caso, se ingresa una cadena de caracteres que corresponde a la cadena ” print(“Hola mundo”) “ La cual regresa el contendio que encontró, en este caso, tanto los paréntesis de cierre como los de apertura, una cadena de caracteres y una instrucción de tipo print.