Un dibujo de una persona

El contenido generado por IA puede ser incorrecto.

**Universidad Nacional Autónoma de México**

**Ingeniería en Computación**

**Compiladores**

**Analizador Léxico**

**Alumnos:**

**Jiménez Elizalde Josue - 320334489**

**Medina Guzmán Santiago - 320067354**

**Tavera Castillo David Emmanuel - 320054831**

**Tenorio Martinez Jesus Alejandro – 320218666**

**Grupo: 5**

**Semestre: 2025-2**

**México, CDMX. Marzo 2025**

1. **Introducción**

En este trabajo, construimos e implementamos un analizador léxico con ayuda del lenguaje de programación python y con la ayuda de los temas vistos en la clase teórica.

Desarrollamos el analizador léxico, de modo que lee una cadena de entrada, y de esta va leyendo cada simbolo uno por uno, de esta forma clasifica cada lexema en un tipo especifico, ya sea una palabra reservada, una constante, un identificador o literal etc. Al final de la ejecución, muestra la cantidad de tokens que hay, así como también la clasificación de todos los lexemas.

1. **Marco Teórico**

El analizador léxico, tambien conocido como analizador lexicográfico ( o tambien conocido como scanner o lexer en ingles), es la primera fase de un compilador, consistente en un programa que recibe como entrada el codigo fuente de otro y produce como salida tokens y símbolos. Estos tokens obtenidos de la salida se utilizan para una etapa posterior del proceso de traducción, siendo la entrada para el analizador sintactico (en ingles parser)

Los lenguajes de programacion incuyen unas reglas basadas en expresiones regulares que indican el conjunto de posibles secuencias de caracteres que definen al token o lexema.

El token léxico o simplemente token es una cadena con un significado asignado y, por lo tanto, identificado. Está estructurado como un par que consta de un “nombre de token” y un “valor de token” opcional. El nombre del token es una categoría de unidad léxica. Los nombres de token comunes son: identificador, los nombres que decide el programador; palabra clave, nombres únicos del lenguaje de programación; puntuación, caracteres de puntuación; operador, símbolos que operan sobre argumentos y que producen resultados; literales, ej. Lógicos, numéricos, textuales o de referencia

Ejemplos de cada uno.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Finalmente el analizador léxico producirá algo parecido a la siguiente secuencia de tokens

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Las gramáticas que se utilizan en los analizadores léxicos estan basados en expresiones regulares. Estas expresiones regulares definen la secuencias de caracteres que pueden formar un token.

*Recursion por la izquierda.*

Ocurre cuando una regla en una gramática libre de contexto tiene una producción que se llama a sí misma al inicio.



Se puede corregir de la siguiente forma:

Diagrama

El contenido generado por IA puede ser incorrecto.

De esta forma la recursión se convierta en una forma iterativa.

*Ambigüedad.*

La ambiguedad ocurre cuando una cadena de entrada puede ser derivada mas de una forma.

Por ejemplo:

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

En la gramática anterior la entrada: a + b +c se puede derivar como (a + b) \* c o como a + (b \* c). + y \*, no tendrían ninguna prescedencia, esto se puede corregir de la siguiente manera:

Tabla

El contenido generado por IA puede ser incorrecto.

\* se resuelve primero porque Term → Term \* Factor está más abajo en la jerarquía.

1. **Desarrollo.**

Lo que hicimos fue un codigo en Python que utiliza la biblioteca re que “es un módulo que permite trabajar con expresiones regulares (regex). Las expresiones regulares son una herramienta para buscar, extraer y manipular patrones de texto”

En la primera parte del codigo definimos

Texto

El contenido generado por IA puede ser incorrecto.

**Cada línea define un tipo de token** con su expresión regular correspondiente.

Se detectan:

* **Palabras clave** (def, import, if, while, return).
* **Literales de cadena** ("texto", incluyendo f-strings).
* **Constantes numéricas** (123, 3.14).
* **Identificadores** (nombre\_de\_variable).
* **Operadores** (+, -, \*, ==, != etc.).
* **Signos de puntuación** ((), [], {}).
* **Espacios en blanco** (solo para ignorarlos después).

Lo que hace la función lexer es ir contando cada token cuando lo va encontrando, re.finditer() busca todas las coincidencias de la regex en el código fuente. Ignora espacios en blanco. Clasifica cada token según su tipo (keywords, identifiers, etc.).

Texto

El contenido generado por IA puede ser incorrecto.

*Buenas prácticas de ingeniería de software*

Aplicamos testing a este programa usando otro programa, “test\_lexer”, que usa el módulo unittest, diseñado para verificar el correcto funcionamiento del analizador léxico (lexer) que escribio antes.

class TestLexer(unittest.TestCase): Hereda de unittest.TestCase, lo que significa que es una clase de pruebas unitarias. Contiene pruebas para verificar si lexer() clasifica correctamente los tokens

def test\_variables\_y\_bucle(self): Prueba que el lexer detecte variables y estructuras de control (while).

Texto

El contenido generado por IA puede ser incorrecto.

Se usa Code, como codigo de prueba, se verifican los resultados usando assert y si todas las afirmaciones se cumplen, la prueba pasa.

def test\_import\_y\_funcion(self): Prueba la detección de import, def y return en funciones. Ejecuta el lexer con el Code y se verifica con assert, si todas las verificaciones pasan, la prueba es exitosa.

Texto

El contenido generado por IA puede ser incorrecto.

Nuestro codigo tambien cumple con el principio de responsabilidad simple, lo que nos permite identificar y corregir errores en clases con una única responsabilidad, asi como reducir la probabilidad de errores en cascada, ya que un cambio en una clase no afecta a otras clases

1. **Resultados.**

Al ejecutar nuestro programa, con la entrada:

Texto

El contenido generado por IA puede ser incorrecto.

La salida es:

Texto

El contenido generado por IA puede ser incorrecto.

Lo que era la salida esperada

1. **Conclusiones**

A lo largo de la implementación del analizador léxico, aprendimos varios puntos importantes sobre el funcionamiento de esta fase en la compilación, como identificar y clasificar partes del código en tokens, también definimos expresiones regulares. Aplicamos SRP, el código inicial mezclaba muchas responsabilidades en la función lexer(). Lo mejoramos separando la definición de reglas, la generación de expresiones y el análisis de tokens. Usamos unittests para probar nuestro codigo y asegurarnos de que clasificaba los tokens de manera correcta.

En conclusión estamos listos para pasar al parser.

1. **Referencias**

* *A. V. Aho, R. Sethi, y J. D. Ullman, Compiladores: principios, técnicas y herramientas. Pearson Educación, 1990.*
* *I. R. Martínez y L. Olivé, Compiladores. Epistemología evolucionista, 1977.*
* *A. V. Aho, M. S. Lam, y J. D. Ullman, Compilers: Principles, Techniques & Tools. Pearson Education, 2007.*