

UNIVERSIDAD AUTÓNOMA DE CHIAPAS



FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

ING. EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

6 "M"

COMPILADORES

SUBCOMPETENCIA I.- ANÁLISIS LÉXICO

ACT. 1.3.1 PRÁCTICA. UNIDAD 1. REALIZA UN ANALIZADOR LÉXICO
EN PYTHON CONTEO VARIABLES RESERVADA

ALUMNO: MARCO ANTONIO ZÚÑIGA MORALES – A211121

DOCENTE: DR. LUIS GUTIÉRREZ ALFARO

TUXTLA GUTIÉRREZ, CHIAPAS
SÁBADO, 26 DE AGOSTO DE 2023

ACT. 1.3.1 PRÁCTICA. UNIDAD 1. REALIZA UN ANALIZADOR LÉXICO EN PYTHON CONTEO VARIABLES RESERVADA

```
#Librerias
import tkinter as tk
import re
from tkinter import ttk

#Define una clase Lexer para el análisis léxico
class Lexer:
    def __init__(self):
        #Lista de palabras reservadas, operadores y delimitadores
        self.RESERVADA = ['for', 'do', 'while', 'if', 'else', 'public', 'static', 'void', 'int', 'main']
        self.OPERADOR = ['=', '+', '-', '*', '/']
        self.DELIMITADOR = ['(', ')', '{', '}', ';']

        #Expresiones regulares para patrones de tokens
        self.tokens_regex = {
            'RESERVADA': '|'.join(r'\b' + re.escape(keyword) + r'\b' for keyword in self.RESERVADA),
            'OPERADOR': '|'.join(map(re.escape, self.OPERADOR)),
            'DELIMITADOR': '|'.join(map(re.escape, self.DELIMITADOR)),
            'NUMERO': r'\d+(\.\d+)?', #Patrón para números enteros y decimales
            'IDENTIFICADOR': r'[A-Za-z_]+' #Patrón para identificadores
        }

        #Compila las expresiones regulares en un patrón combinado
        self.token_patterns = re.compile('|'.join(f'(?P<{t}>{self.tokens_regex[t]})' for t in self.tokens_regex))

    #Tokeniza el texto de entrada
    def tokenize(self, text):
        tokens = [] #Lista para almacenar los tokens
        lines = text.split('\n') #Divide el texto de entrada en líneas
        NumeroLinea = 1 #Inicializa el número de línea
        #Itera a través de cada línea del texto
        for line in lines:
            line_has_tokens = False #Bandera para verificar si la línea tiene tokens
            #Busca los patrones de tokens en la línea actual
            for match in self.token_patterns.finditer(line):
                line_has_tokens = True
                #Itera a través de los grupos coincidentes de la expresión regular
                for token_type, token_value in match.groupdict().items():
                    #Verifica el tipo de token y su longitud
                    if token_type == 'IDENTIFICADOR' and token_value and len(token_value) > 1:
                        tokens.append((NumeroLinea, 'ERROR LÉXICO', token_value)) #Agrega un token de error léxico
                    elif token_type == 'IDENTIFICADOR' and token_value and len(token_value) == 1:
                        tokens.append((NumeroLinea, 'IDENTIFICADOR', token_value)) #Agrega un token identificador
                    elif token_value:
                        tokens.append((NumeroLinea, token_type, token_value)) #Agrega un token con su tipo y valor
            #Si la línea tiene tokens, incrementa el número de línea
            if line_has_tokens:
                NumeroLinea += 1
        return tokens #Devuelve la lista de tokens resultante

    #Realiza el análisis de los tokens y genera una cadena de resultados
    def analyze(self, text):
        tokens = self.tokenize(text) #Tokeniza el texto de entrada
        result = "Token\t\tLexema\t\tLínea\n" #Crea una cadena para almacenar los resultados formateados
        for line_number, token_type, token_value in tokens: #Itera a través de la lista de tokens generados
            result += f"{token_type}\t\t{token_value}\t\t{line_number}\n" #Agrega una línea formateada al resultado
        return result #Devuelve la cadena de resultados
```

```

#Define una clase LexerApp para la interfaz gráfica de usuario
class LexerApp:
    def __init__(self): #Define una clase llamada LexerApp para la interfaz grafica de usuario
        self.window = tk.Tk() #Crea una ventana de la clase
        self.window.title("Analizador Léxico") #Establece el título de la ventana

        #Crea una etiqueta para el título de la aplicación
        self.text_label = tk.Label(text="----- ANALIZADOR LÉXICO -----", height=2, width=45, font=("Arial", 15, 'bold'), fg="#FFD700",
        bg="#F0F0F0")
        self.text_label.pack(pady=5)

        #Crea un cuadro de texto para la entrada de texto
        self.text_input = tk.Text(self.window, height=8, width=55, font=("Arial", 12))
        self.text_input.pack(pady=5)

        #Crea un marco para los botones
        self.button_frame = tk.Frame(self.window)
        self.button_frame.pack()

        #Crea un botón para realizar el análisis léxico del texto de entrada
        self.analyze_button = tk.Button(self.button_frame, text="Analizar", command=self.analyze_text, bg="#ADD8E6", font=("Arial",
14, 'bold'))
        self.analyze_button.grid(row=0, column=0, padx=30, pady=10)

        #Crea un botón para limpiar el cuadro de texto
        self.clean_button = tk.Button(self.button_frame, text="Limpiar", command=self.clean_text, bg="#FFC300", font=("Arial", 14,
'bold'))
        self.clean_button.grid(row=0, column=1, padx=30, pady=10)

        #Crea un botón para salir del programa
        self.exit_button = tk.Button(self.button_frame, text="Salir", command=self.exit_app, bg="#FF69B4", font=("Arial", 14, 'bold'))
        self.exit_button.grid(row=0, column=2, padx=30, pady=10)

        #Crea un widget Treeview para mostrar los resultados de análisis
        self.treeview = ttk.Treeview(self.window, columns=("Token", "Lexema", "Linea", show="headings"))
        self.treeview.heading("Token", text="Token")
        self.treeview.heading("Lexema", text="Lexema")
        self.treeview.heading("Linea", text="Linea")
        self.treeview.pack()

        # Configura la alineación de las columnas para centrar el contenido
        self.treeview.column("Token", anchor="center")
        self.treeview.column("Lexema", anchor="center")
        self.treeview.column("Linea", anchor="center")

        self.count_tree = ttk.Treeview(self.window, columns=("Elemento", "Cantidad", show="headings"))
        self.count_tree.heading("Elemento", text="Elemento")
        self.count_tree.heading("Cantidad", text="Cantidad")
        self.count_tree.pack(pady=10)

        self.count_tree.column("Elemento", anchor="center")
        self.count_tree.column("Cantidad", anchor="center")

    def analyze_text(self):
        lexer = Lexer() #Crea una instancia de la clase Lexer
        text = self.text_input.get("1.0", "end").strip() # Obtiene el texto ingresado y elimina espacios en blanco
        if not text: # Verifica si el texto está vacío
            return # Si el texto está vacío, no realiza el análisis ni muestra los resultados

        result = lexer.tokenize(text)

        # Limpia las entradas existentes en el Treeview
        self.treeview.delete(*self.treeview.get_children())

        num_reserved_words = 0
        num_operators = 0
        num_delimiters = 0
        num_numbers = 0
        num_identifiers = 0
        num_lexical_errors = 0
        num_open_parentheses = 0
        num_close_parentheses = 0
        num_open_braces = 0
        num_close_braces = 0
        num_semicolons = 0

        for line_number, token_type, token_value in result:
            self.treeview.insert("", "end", values=(token_type, token_value, line_number))

            if token_type == "RESERVADA":
                num_reserved_words += 1
            elif token_type == "OPERADOR":
                num_operators += 1
            elif token_type == "DELIMITADOR":
                num_delimiters += 1
                if token_value == '(':
                    num_open_parentheses += 1
                elif token_value == ')':
                    num_close_parentheses += 1
                elif token_value == '{':
                    num_open_braces += 1
                elif token_value == '}':
                    num_close_braces += 1
                elif token_value == ';':
                    num_semicolons += 1
            elif token_type == "NUMERO":
                num_numbers += 1
            elif token_type == "IDENTIFICADOR":
                num_identifiers += 1
            elif token_type == "ERROR LÉXICO":
                num_lexical_errors += 1

        # Inserta las cuantificaciones en el Treeview de conteo
        self.count_tree.delete(*self.count_tree.get_children()) # Limpia el contenido existente

        # Inserta las cuantificaciones en la tabla
        quantities = [
            ("Palabras Reservadas", num_reserved_words),
            ("Operadores", num_operators),
            ("Delimitadores", num_delimiters),
            ("Números", num_numbers),
            ("Errores Léxicos", num_lexical_errors),
            ("Identificadores", num_identifiers),
            ("Paréntesis de Apertura", num_open_parentheses),
            ("Paréntesis de Cierre", num_close_parentheses),
            ("Llaves de Apertura", num_open_braces),
            ("Llaves de Cierre", num_close_braces),
            ("Punto y Coma", num_semicolons)
        ]

        for element, quantity in quantities:
            self.count_tree.insert("", "end", values=(element, quantity))


    def clean_text(self):
        self.text_input.delete("1.0", "end") #Borra el contenido del cuadro de texto
        self.treeview.delete(*self.treeview.get_children()) #Limpia el contenido del ttk
        self.count_tree.delete(*self.count_tree.get_children()) # Limpia el contenido del Treeview de conteo

    def exit_app(self): #Define un método para salir del programa
        self.window.destroy()

    def run(self): #Define un método para ejecutar la aplicación de la interfaz grafica
        self.window.mainloop() #Inicia el bucle de la interfaz grafica

app = LexerApp() #Crea una instancia de la clase Lexer app
app.run() #Ejecuta la aplicación de la interfaz grafica llamando al método "run"

```



Analizador léxico

ANALIZADOR LÉXICO

```

public static void main ()
{
    int n = 23.23;
}

(( ))

publica maina statica voida

```

Analizar

Limpiar

Salir

Token	Lexema	Línea
RESERVADA	public	1
RESERVADA	static	1
RESERVADA	void	1
RESERVADA	main	1
DELIMITADOR	(1
DELIMITADOR)	1
DELIMITADOR	{	2
RESERVADA	int	3
IDENTIFICADOR	n	3
OPERADOR	=	3

Elemento	Cantidad
Palabras Reservadas	5
Operadores	1
Delimitadores	9
Números	1
Errores Léxicos	4
Identificadores	1
Paréntesis de Apertura	3
Paréntesis de Cierre	3
Llaves de Apertura	1
Llaves de Cierre	1