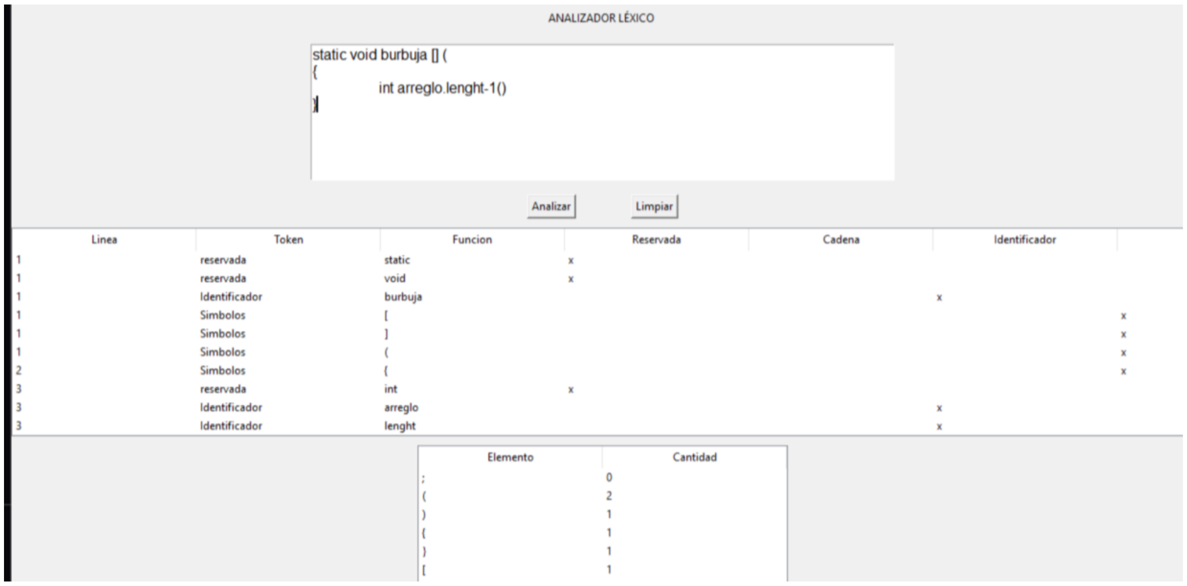
UNIVERSIDAD AUTONOMA DE CHIAPAS

Facultad de contaduría y administración

Materia: Compiladores

Docente: ***Luis Gutiérrez Alfaro***


ANALIZADOR LÉXICO

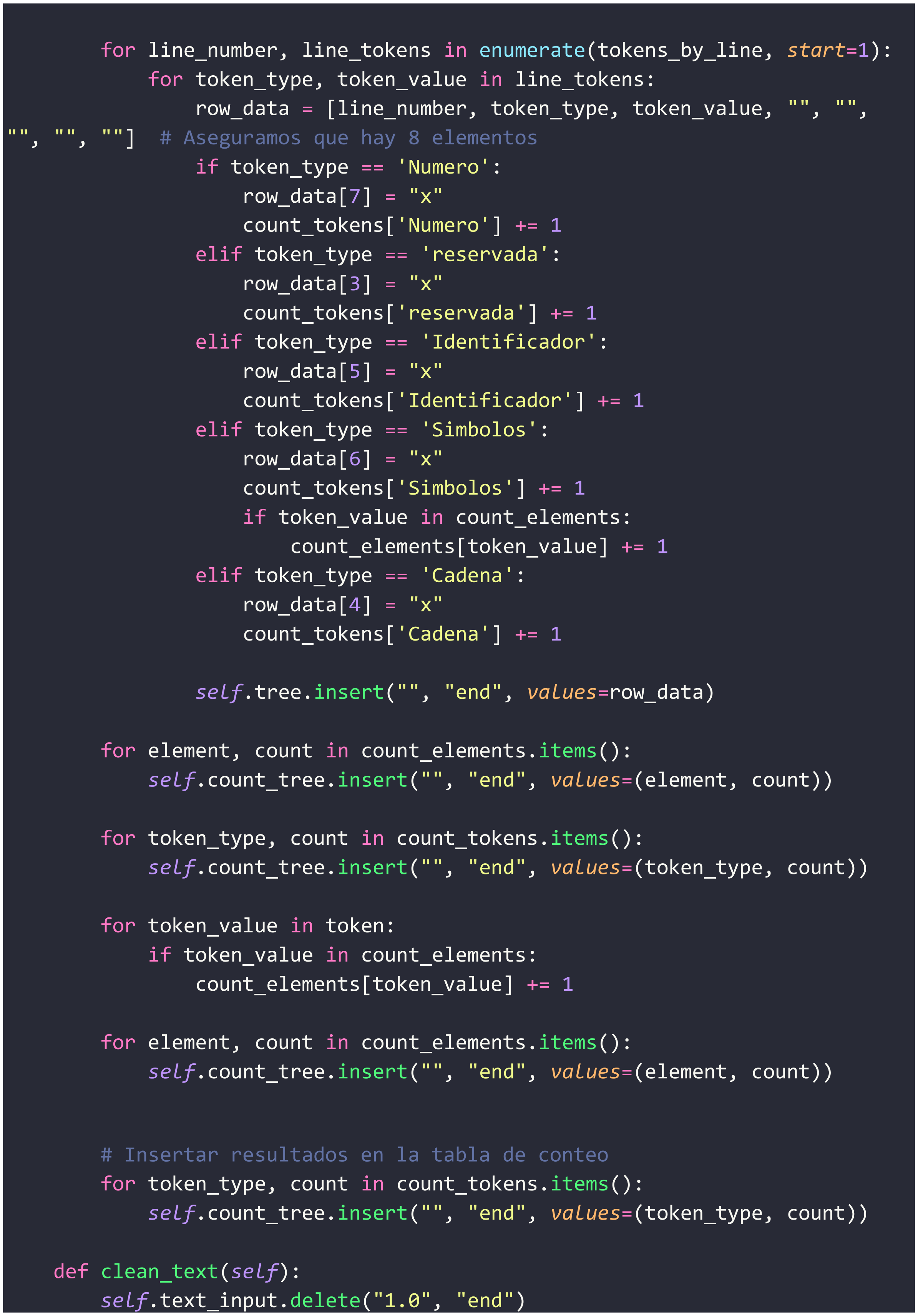
```
static void burbuja [] (
{
		int arreglo.lenght-1()
}
```

Linea	Token	Funcion	Reservada	Cadena	Identificador	
1	reservada	static	x			
1	reservada	void	x			
1	Identificador	burbuja			x	
1	Simbolos	[				x
1	Simbolos	]				x
1	Simbolos	(				x
2	Simbolos	{				x
3	reservada	int	x			
3	Identificador	arreglo			x	
3	Identificador	lenght			x	

Elemento	Cantidad
;	0
(	2
)	1
{	1
}	1
[	1

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

class Lexer:
    def __init__(self):
        self.reservada_keywords = ['if', 'else', 'while', 'for', 'int',
'float', 'Cadena', 'print', 'programa', 'read', 'terminar',
'imprimir','public','static','void','main']
        self.Simboloss = ['+', '-', '*', '/', '=', '==', '!=', '<', '>',
'<=', '>=','[',']','(', ')', '{', '}', ';', ',', '"', "'"]
        self.token_patterns = [
            ('Cadena', r'"(?:[^"\\]|\\.)*"'),
            ('VARIABLE', r'\$\w+'),
            ('Numero', r'^\-?[0-9]+(\.[0-9]+)?$ |[0-9]+|-?[0-9]+'),
            ('reservada', '|'.join(r'\b' + re.escape(keyword) + r'\b' for
keyword in self.reservada_keywords)),
            ('Identificador', r'[A-Za-z_][A-Za-z0-9_]*'),
            ('Simbolos', '|'.join(map(re.escape, self.Simboloss))),
            ('SPACE', r'\s+'),
        ]
        self.token_regex = '|'.join(f'(?P<{name}>{pattern})' for name,
pattern in self.token_patterns)
        self.token_pattern = re.compile(self.token_regex)

    def tokenize(self, text):
        tokens = []
        position = 0
        while position < len(text):
            match = self.token_pattern.match(text, position)
            if match:
                token_type = match.lastgroup
                if token_type != 'SPACE':
                    token_value = match.group(token_type)
                    tokens.append((token_type, token_value))
                position = match.end()
            else:
                position += 1
        return tokens

class LexerApp:
    def __init__(self):
        self.windows = tk.Tk() #Crea una ventana de la clase
        self.windows.title("Analizador léxico") #Establece el titulo de la
ventana

        #Crea una etiqueta para el titulo de la aplicacion
        self.text_label = tk.Label(text="ANALIZADOR LÉXICO ", height=2,
width=50,)
        self.text_label.pack(pady=5)

        #Crea un cuadro de texto para la entrada de texto
        self.text_input = tk.Text(self.windows, height=8, width=70,
font=("Arial", 12))
        self.text_input.pack(pady=5)

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

        #crea un boton para realizar el analisis lexico del texto de entrada
        self.analyze_button = tk.Button(self.button_frame, text="Analizar",
command=self.analyze_text)
        self.analyze_button.grid(row=0, column=0, padx=30, pady=10)

        #crea un boton para limpiar el cuadro de texto
        self.clean_button = tk.Button(self.button_frame, text="Limpiar",
command=self.clean_text)
        self.clean_button.grid(row=0, column=1, padx=30, pady=10)

        self.tree = ttk.Treeview(self.windows, columns=("Linea", "Token",
"Funcion", "Reservada", "Cadena", "Identificador", "Símbolo",
"Numero"),show="headings")
```

```python
        self.tree.heading("Linea", text="Linea")
        self.tree.heading("Token", text="Token")
        self.tree.heading("Funcion", text="Funcion")
        self.tree.heading("Reservada", text="Reservada")
        self.tree.heading("Cadena", text="Cadena")
        self.tree.heading("Identificador", text="Identificador")
        self.tree.heading("Símbolo", text="Símbolo")
        self.tree.heading("Numero", text="Numero")
        self.tree.pack()

        self.count_tree = ttk.Treeview(self.windows, 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)

    def analyze_text(self):
        lexer = Lexer()
        text = self.text_input.get("1.0", "end")
        lines = text.split('\n')
        tokens_by_line = [lexer.tokenize(line) for line in lines]

        self.tree.delete(*self.tree.get_children())
        self.count_tree.delete(*self.count_tree.get_children())

        # Diccionario para contar los tokens
        count_tokens = {
            'Cadena': 0,
            'reservada': 0,
            'Numero': 0,
            'Identificador': 0,
            'Simbolos': 0
        }


         # Count elements
        count_elements = {
            ';': 0,
            '(': 0,
            ')': 0,
            '{': 0,
            '}': 0,
            '[': 0,
            ']': 0,
        }
```


```
        for line_number, line_tokens in enumerate(tokens_by_line, start=1):
            for token_type, token_value in line_tokens:
                row_data = [line_number, token_type, token_value, "", "",
"", "", ""]  # Aseguramos que hay 8 elementos
                if token_type == 'Numero':
                    row_data[7] = "x"
                    count_tokens['Numero'] += 1
                elif token_type == 'reservada':
                    row_data[3] = "x"
                    count_tokens['reservada'] += 1
                elif token_type == 'Identificador':
                    row_data[5] = "x"
                    count_tokens['Identificador'] += 1
                elif token_type == 'Simbolos':
                    row_data[6] = "x"
                    count_tokens['Simbolos'] += 1
                    if token_value in count_elements:
                        count_elements[token_value] += 1
                elif token_type == 'Cadena':
                    row_data[4] = "x"
                    count_tokens['Cadena'] += 1

                self.tree.insert("", "end", values=row_data)

        for element, count in count_elements.items():
            self.count_tree.insert("", "end", values=(element, count))

        for token_type, count in count_tokens.items():
            self.count_tree.insert("", "end", values=(token_type, count))

        for token_value in token:
            if token_value in count_elements:
                count_elements[token_value] += 1

        for element, count in count_elements.items():
            self.count_tree.insert("", "end", values=(element, count))


        # Insertar resultados en la tabla de conteo
        for token_type, count in count_tokens.items():
            self.count_tree.insert("", "end", values=(token_type, count))

    def clean_text(self):
        self.text_input.delete("1.0", "end")
```

```python
        self.tree.delete(*self.tree.get_children())
        self.count_tree.delete(*self.count_tree.get_children())

    def run(self):
        self.windows.mainloop()

app = LexerApp()
app.run()
```