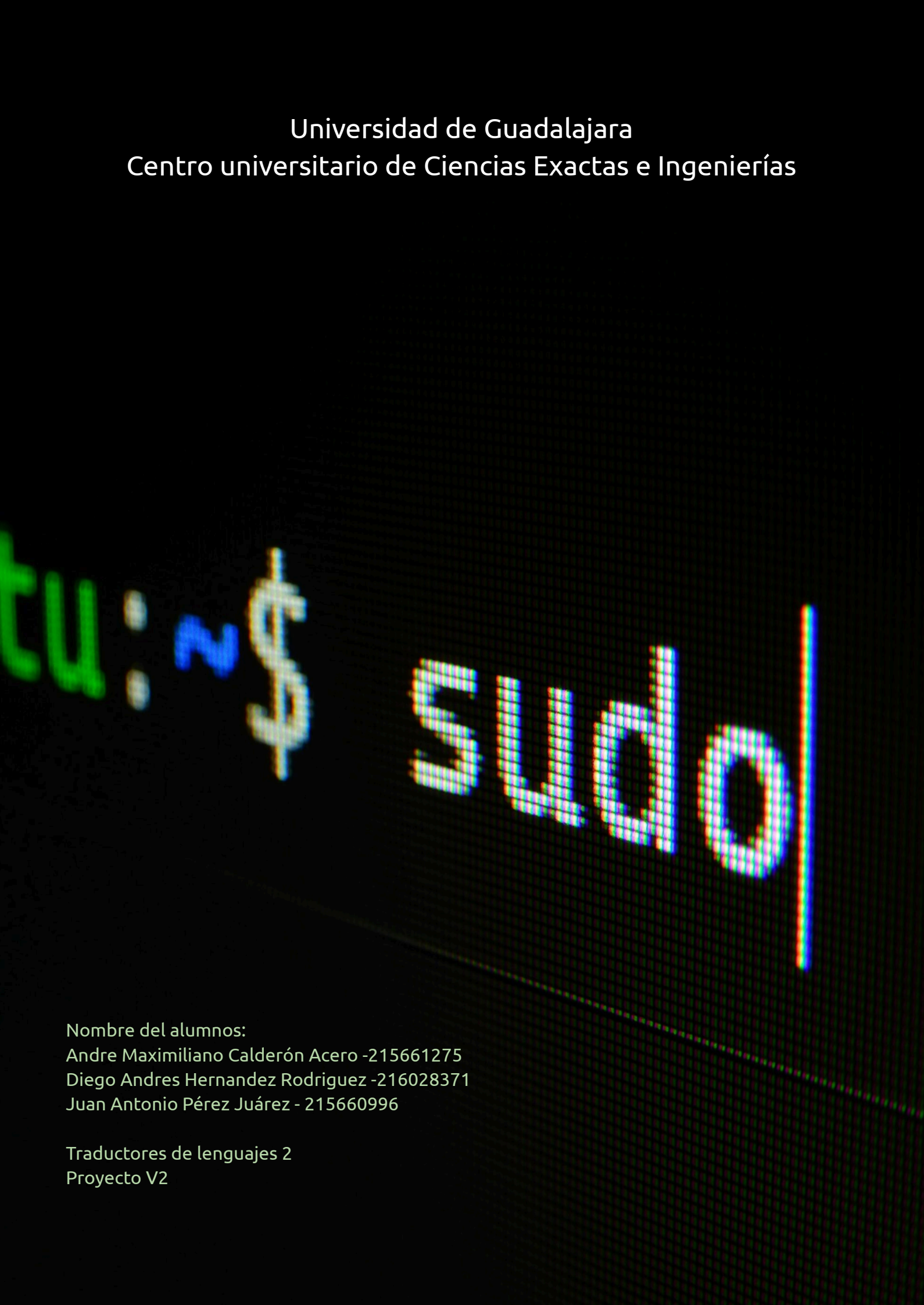
Universidad de Guadalajara
Centro universitario de Ciencias Exactas e Ingenierías



Nombre del alumnos:
Andre Maximiliano Calderón Acero -215661275
Diego Andres Hernandez Rodriguez -216028371
Juan Antonio Pérez Juárez - 215660996

Traductores de lenguajes 2
Proyecto V2

Actividad

Reutilizar la parte uno para generar como salida el código de 3 direcciones.

Marco Teórico

Es importante recalcar que para esta práctica nos sirve el hecho de tener la teoría de la actividad pasada, por que sigue siendo relevante para nuestro proyecto, así que no está de más el revisarlo nuevamente.

Un analizador sintáctico descendente predictivo es un tipo de parser que procesa la entrada desde el símbolo inicial de la gramática hacia las hojas, utilizando un enfoque predictivo basado en gramáticas LL(1), lo que evita el retroceso mediante el uso de una tabla de análisis que determina de manera única la producción a aplicar en cada paso. Este analizador trabaja en conjunto con un analizador léxico, el cual se encarga de leer el flujo de caracteres de entrada y agruparlos en tokens como identificadores (definidos por patrones como [a-zA-Z_][a-zA-Z0-9_]*) y números de más de un dígito (enteros o decimales, reconocidos mediante expresiones como [0-9]+(\.[0-9]+)?).

La gramática debe estar diseñada para ser no ambigua y sin recursión por la izquierda, permitiendo así una derivación eficiente. El proceso de análisis sintáctico se basa en comparar los tokens generados por el lexer con las reglas gramaticales almacenadas en la tabla predictiva, expandiendo los no terminales según corresponda y construyendo el árbol de derivación de manera sistemática. Este enfoque es eficiente en tiempo lineal, aunque presenta limitaciones al requerir gramáticas LL(1), lo que puede exigir modificaciones en la definición original del lenguaje. Su aplicación es común en el desarrollo de intérpretes y compiladores para lenguajes con estructuras sintácticas bien definidas y relativamente simples.

1) E → E1 + T	E.nodo = new Nodo('+', E1.nodo, T.nodo)
2) E → E1 - T	E.nodo = new Nodo('-', E1.nodo, T.nodo)
3) E → T	E.nodo = T.nodo
4) T → T1 * F	T.nodo = new Nodo('*', T1.nodo, F.nodo)
5) T → T1 / F	T.nodo = new Nodo('/', T1.nodo, F.nodo)
6) T → F	T.nodo = F.nodo
7) F → (E)	F.nodo = E.nodo
8) F → id	F.nodo = new Hoja(id, id.entrada)
9) F → num	F.nodo = new Hoja(num, num.val)

Ahora agregamos algo que no habíamos tratado, el código de 3 direcciones.

En ciencias de la computación, el código de tres direcciones (en inglés: three address code, normalmente abreviado a TAC o 3AC) es un lenguaje intermedio usado por compiladores

optimizadores para ayudar en las transformaciones de mejora de código. Cada instrucción TAC tiene a lo sumo tres operandos y es típicamente una combinación de asignación y operador binario. Por ejemplo, t1 := t2 + t3. El nombre proviene del uso de tres operandos en estas declaraciones aunque instrucciones con menos operandos pueden existir.

Ya que el código de tres direcciones es usado como un lenguaje intermedio en los compiladores, los operandos normalmente no contendrán direcciones de memoria o registros concretos, sino que direcciones simbólicas que serán convertidas en direcciones reales durante la asignación de registros. Es también común que los nombres de los operandos sean numerados secuencialmente ya que el código de tres direcciones es típicamente generado por el compilador.

Código en python de la práctica

Python

```python
#ANALIZADOR
import tkinter as tk
from tkinter import filedialog
from tkinter import ttk
from lexer import Lexer

class LexicalAnalyzerGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Analizador")
        self.root.configure(bg='#f0f0f0')  # Color de fondo suave

        # Configuración de estilos
        style = ttk.Style()
        style.configure("Treeview",
                        background="#ffffff",
                        fieldbackground="#ffffff",
                        foreground="#333333")
        style.configure("Treeview.Heading",
                        background="#e1e1e1",
                        font=('Arial', 9, 'bold'))
        style.configure("TButton",
                        padding=5,
                        background="#4a90e2")

        # Frame principal con padding
        main_frame = ttk.Frame(self.root, padding="10")
```

```
        main_frame.pack(fill=tk.BOTH, expand=True)

        # Área de texto con borde y fondo
        self.text_area = tk.Text(
            main_frame,
            wrap="word",
            width=50,
            height=10,
            font=('Consolas', 10),
            bg='white',
            relief="solid",
            borderwidth=1
        )
        self.text_area.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)
        self.text_area.insert(tk.END, "a = 7\nb = 2\nsuma = a+b\nresta =
ab\nprint('suma:',suma, '\nResta:',resta )\n\n")

        # Frame para botones
        button_frame = ttk.Frame(main_frame)
        button_frame.pack(pady=5)

        # Botones estilizados
        self.read_button = ttk.Button(
            button_frame,
            text="Leer archivo",
            command=self.read_file,
            style="TButton"
        )
        self.read_button.pack(side=tk.LEFT, padx=5)

        self.analyze_button = ttk.Button(
            button_frame,
            text="Analizar léxicamente",
            command=self.analyze_lexically,
            style="TButton"
        )
        self.analyze_button.pack(side=tk.LEFT, padx=5)

        self.create_table()

        # Frame para la tabla semántica
        self.semantic_frame = ttk.Frame(main_frame)
        self.semantic_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)
```

```
        # Tabla semántica
        self.semantic_tree = ttk.Treeview(
            self.semantic_frame,
            columns=("Lexma", "Token", "Coincidencia"),
            show="headings",
            height=10
        )

        # Configuración de columnas
        for col in ("Lexma", "Token", "Coincidencia"):
            self.semantic_tree.heading(col, text=col)
            self.semantic_tree.column(col, width=150)

        self.semantic_tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Scrollbar para tabla semántica
        self.semantic_scrollbar = ttk.Scrollbar(
            self.semantic_frame,
            orient="vertical",
            command=self.semantic_tree.yview
        )
        self.semantic_scrollbar.pack(side=tk.RIGHT, fill="y")

self.semantic_tree.configure(yscrollcommand=self.semantic_scrollbar.set)

    def create_table(self):
        # Frame para la primera tabla
        self.table_frame = ttk.Frame(self.root)
        self.table_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

        # Primera tabla
        self.tree = ttk.Treeview(
            self.table_frame,
            columns=("Lexma", "Token"),
            show="headings",
            height=10
        )

        # Configuración de columnas
        for col in ("Lexma", "Token"):
            self.tree.heading(col, text=col)
            self.tree.column(col, width=150)
```

```
        self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Scrollbar para primera tabla
        self.scrollbar = ttk.Scrollbar(
            self.table_frame,
            orient="vertical",
            command=self.tree.yview
        )
        self.scrollbar.pack(side=tk.RIGHT, fill="y")
        self.tree.configure(yscrollcommand=self.scrollbar.set)

    def update_table(self, tokens):
        for token in tokens:
            lexma, token_name = token[1], token[0]
            self.tree.insert("", "end", values=(lexma, token_name))

    def read_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text files",
"*.txt")])
        if file_path:
            with open(file_path, 'r') as file:
                content = file.read()
                self.text_area.delete("1.0", tk.END)
                self.text_area.insert(tk.END, content)

    def analyze_lexically(self):
        input_string = self.text_area.get("1.0", tk.END)
        lexer = Lexer(input_string)
        tokens = lexer.lex()

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

        for token in tokens:
            lexma, token_name = token[1], token[0]
            self.tree.insert("", "end", values=(lexma, token_name))

        self.compare_with_file(tokens)

    def compare_with_file(self, tokens_input):
        file_path = filedialog.askopenfilename(filetypes=[("Text files",
"*.txt")])
        if file_path:
```

```
            with open(file_path, 'r') as file:
                content = file.read()
                lexer = Lexer(content)
                tokens_file = lexer.lex()

                self.semantic_tree.delete(*self.semantic_tree.get_children())

                for token_input, token_file in zip(tokens_input, tokens_file):
                    lexma_input, token_name_input = token_input[1],
token_input[0]
                    lexma_file, token_name_file = token_file[1], token_file[0]
                    coincidence = "✓" if lexma_input == lexma_file and
token_name_input == token_name_file else "✗"

                    # Insertar con tags para colorear
                    tag = "valid" if coincidence == "✓" else "invalid"
                    self.semantic_tree.insert("", "end",
                                              values=(lexma_input,
token_name_input, coincidence),
                                              tags=(tag,))

                # Configurar colores para las filas
                self.semantic_tree.tag_configure("valid", background="#e6ffe6")
                self.semantic_tree.tag_configure("invalid",
background="#ffe6e6")

if __name__ == "__main__":
    root = tk.Tk()
    app = LexicalAnalyzerGUI(root)
    root.mainloop()
```

Python

```
#LEXER
import re
from tokens import tokens # Asegúrate de que se importa correctamente el diccionario de
tokens

class Lexer:
    def __init__(self, input_string):
        self.input_string = input_string
        self.pos = 0
        self.token_list = []
```

```
    def lex(self):
        while self.pos < len(self.input_string):
            match = None
            for token, pattern in tokens.items():
                regex = re.compile(pattern)
                match = regex.match(self.input_string, self.pos)
                if match:
                    value = match.group(0)
                    if token != 'SALTO LINEA':
                        self.token_list.append((token, value))
                    self.pos = match.end()
                    break

            if not match:
                print("Error: Caracter no reconocido -", self.input_string[self.pos])
                self.pos += 1
        return self.token_list

class CodigoTresDirecciones:
    def __init__(self, tokens):
        self.tokens = tokens
        self.temp_count = 1
        self.code = []

    def generate_code(self):
        for token, value in self.tokens:
            if token == 'Palabra reservada':
                self.code.append(f"t{self.temp_count} = {value}")
                self.temp_count += 1
            elif token == 'ASIGNACION':
                self.code.append(f"{value} = t{self.temp_count - 1}")
            elif token == 'ENTERO' or token == 'REAL' or token == 'Comillas':
                self.code.append(f"t{self.temp_count} = {value}")
                self.temp_count += 1
            elif token == 'SUMA' or token == 'RESTA' or token == 'MULTIPLICACION' or
token == 'DIVISION':
                self.code.append(f"t{self.temp_count} = t{self.temp_count - 2}
{value}t{self.temp_count - 1}")
                self.temp_count += 1
            elif token == 'OPERADOR_RELACIONAL' or token == 'AND_LOGICO' or token ==
'OR_LOGICO' or token == 'NOT_LOGICO':
                self.code.append(f"t{self.temp_count} = t{self.temp_count - 2}
{value}t{self.temp_count - 1}")
                self.temp_count += 1
            elif token == 'PARENTESIS_IZQUIERDO' or token == 'PARENTESIS_DERECHO' or
token == 'LLAVE_IZQUIERDA' or token == 'LLAVE_DERECHA':
                pass
```

```
            elif token == 'PUNTO_Y_COMA':
                self.code.append("") # Separador de instrucciones
            elif token == 'SALTO LINEA':
                pass # No se necesita generar código para el salto de línea
        return self.code

    def write_code_to_file(self, filename):
        with open(filename, 'w') as f:
            for line in self.code:
                f.write(line + '\n')

input_string = """
 begin
 entero a;
 a:=1;
 if(a<10)
 while(a<10)
 a:=a+1;
 endwhile;
 else
 while(a>0)
 a:=a-1;
 endwhile;
 end;
 end
 """
lexer = Lexer(input_string)
token_list = lexer.lex()
codigo_tres_direcciones = CodigoTresDirecciones(token_list)
codigo_tres_direcciones_result = codigo_tres_direcciones.generate_code()
codigo_tres_direcciones.write_code_to_file('Codigo_tres_direcciones.txt')
```

Python

```
#TOKENS
 'Palabra reservada': r'[a-zA-Z][a-zA-Z0-9]*',
 'ENTERO': r'\d+',
 'REAL': r'\d+\.\d+',
 'SUMA': r'\+',
 'RESTA': r'\-',
 'SALTO LINEA': r'\n',
 'MULTIPLICACION': r'\*',
 'DIVISION': r'\/',
 'ASIGNACION': r'=',
 'OPERADOR_RELACIONAL': r'<|>|<=|>=|!=|==',
```

```
 'AND_LOGICO': r'&&',
 'OR_LOGICO': r'\|\|',
 'NOT_LOGICO': r'!',
 'PARENTESIS_IZQUIERDO': r'\(',
 'PARENTESIS_DERECHO': r'\)',
 'LLAVE_IZQUIERDA': r'{',
 'LLAVE_DERECHA': r'}',
 'PUNTO_Y_COMA': r';',
 'Comillas': r'"',
 'COMA': r',',
}
```

Lógica del programa

Se crea una clase que encapsula toda la funcionalidad del analizador.
El método __init__ configura la ventana principal y todos sus componentes.

```python
class LexicalAnalyzerGUI:
    def __init__(self, root):
        # Configuración inicial
```

Se establece el título de la ventana.
Se configuran los colores de fondo y estilos visuales para mejorar la apariencia
Se establecen los estilos de los widgets ttk (tablas, botones, etc.)

```python
self.root.title("Analizador")
self.root.configure(bg='#f0f0f0')
style = ttk.Style()
# Configuraciones de estilo...
```

Se crea un área de texto donde el usuario puede escribir o ver el código a analizar
Se configura propiedades como fuente, tamaño y color
Se inserta un texto de ejemplo predeterminado

```python
self.text_area = tk.Text(...)
self.text_area.pack(...)
self.text_area.insert(...)
```

Se crea un marco para contener los botones
Se añade dos botones principales: uno para leer archivos y otro para analizar

Conectar cada botón con su función correspondiente

```python
button_frame = ttk.Frame(main_frame)
self.read_button = ttk.Button(...)
self.analyze_button = ttk.Button(...)
```

Se implementa un método para crear la primera tabla (treeview)
Esta tabla mostrará los tokens encontrados en el análisis léxico
Se configura columnas para mostrar lexema y token
Se añade scrollbar para navegar por resultados extensos

```python
def create_table(self):
    # Configuración de la tabla...
```

Se crea una segunda tabla para mostrar comparaciones
Esta tabla incluye columnas adicionales para indicar coincidencias
Se configura esta tabla con su propio scrollbar

```python
self.semantic_tree = ttk.Treeview(...)
# Configuración de la tabla semántica...
```

Se implementa la funcionalidad para abrir un diálogo de selección de archivos
Se lee el contenido del archivo seleccionado
Actualiza el área de texto con el contenido leído

```python
def read_file(self):
    file_path = filedialog.askopenfilename(...)
    # Lectura del archivo...
```

Se obtiene el texto del área de texto
Crea una instancia del analizador léxico
Procesa el texto para obtener tokens
Actualiza la tabla con los resultados del análisis

```python
def analyze_lexically(self):
    input_string = self.text_area.get("1.0", tk.END)
    lexer = Lexer(input_string)
    tokens = lexer.lex()
    # Actualización de la tabla...
```

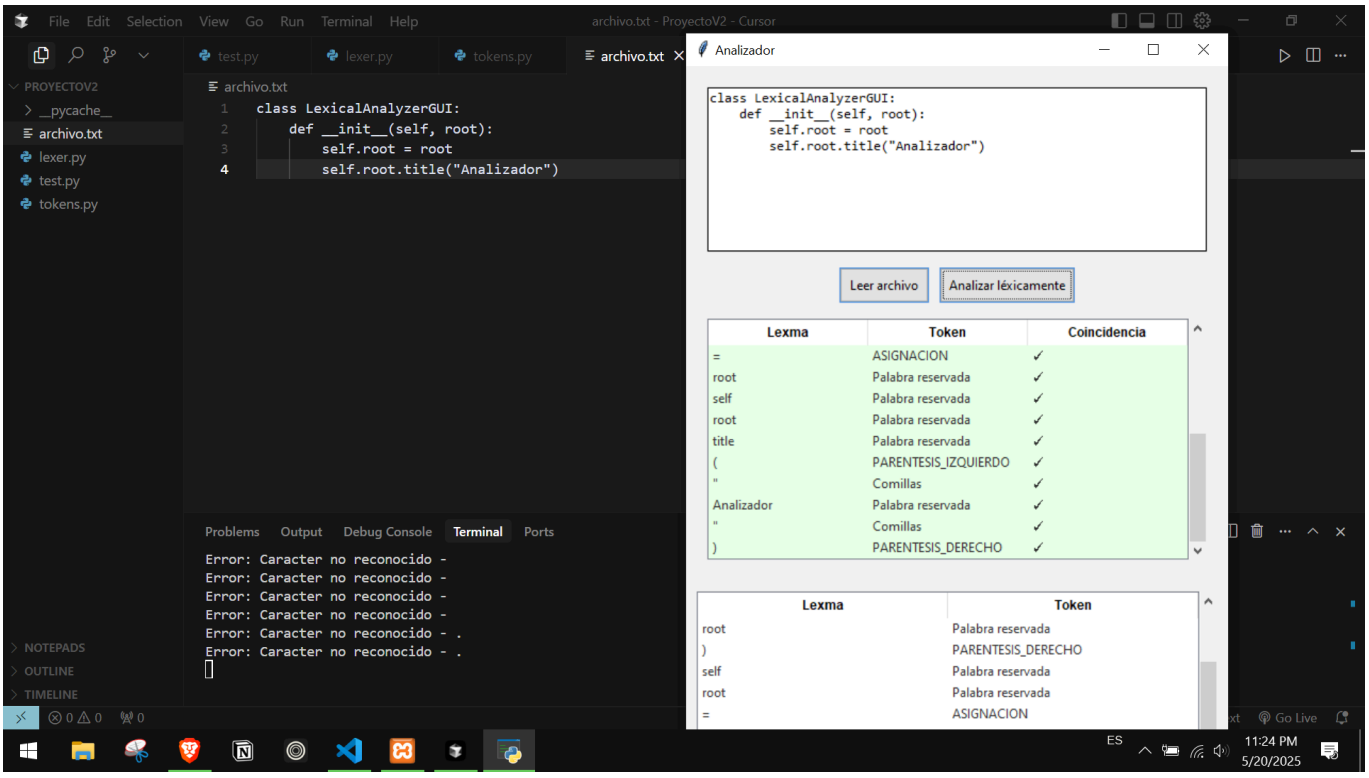
Permite seleccionar un segundo archivo para comparar
Realiza el análisis léxico de este archivo
Compara los tokens de ambos análisis
Muestra los resultados en la tabla semántica con indicadores visuales

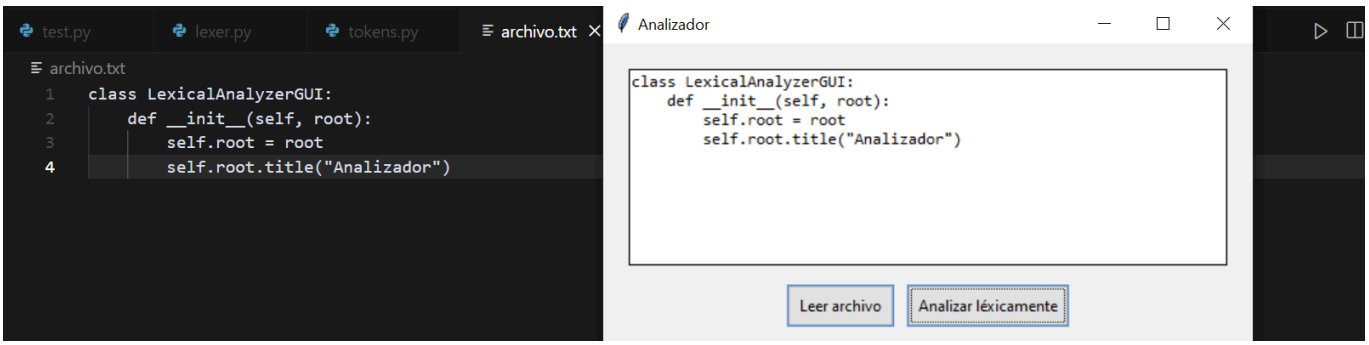
```python
def compare_with_file(self, tokens_input):
    # Selección de archivo...
    # Análisis del archivo seleccionado...
    # Comparación de tokens...
```

Verifica si el script se está ejecutando directamente
Se crea la ventana principal de tkinter
Se Inicializa la aplicación
Inicia el bucle principal de eventos de la interfaz gráfica

```python
if __name__ == "__main__":
    root = tk.Tk()
    app = LexicalAnalyzerGUI(root)
    root.mainloop()
```

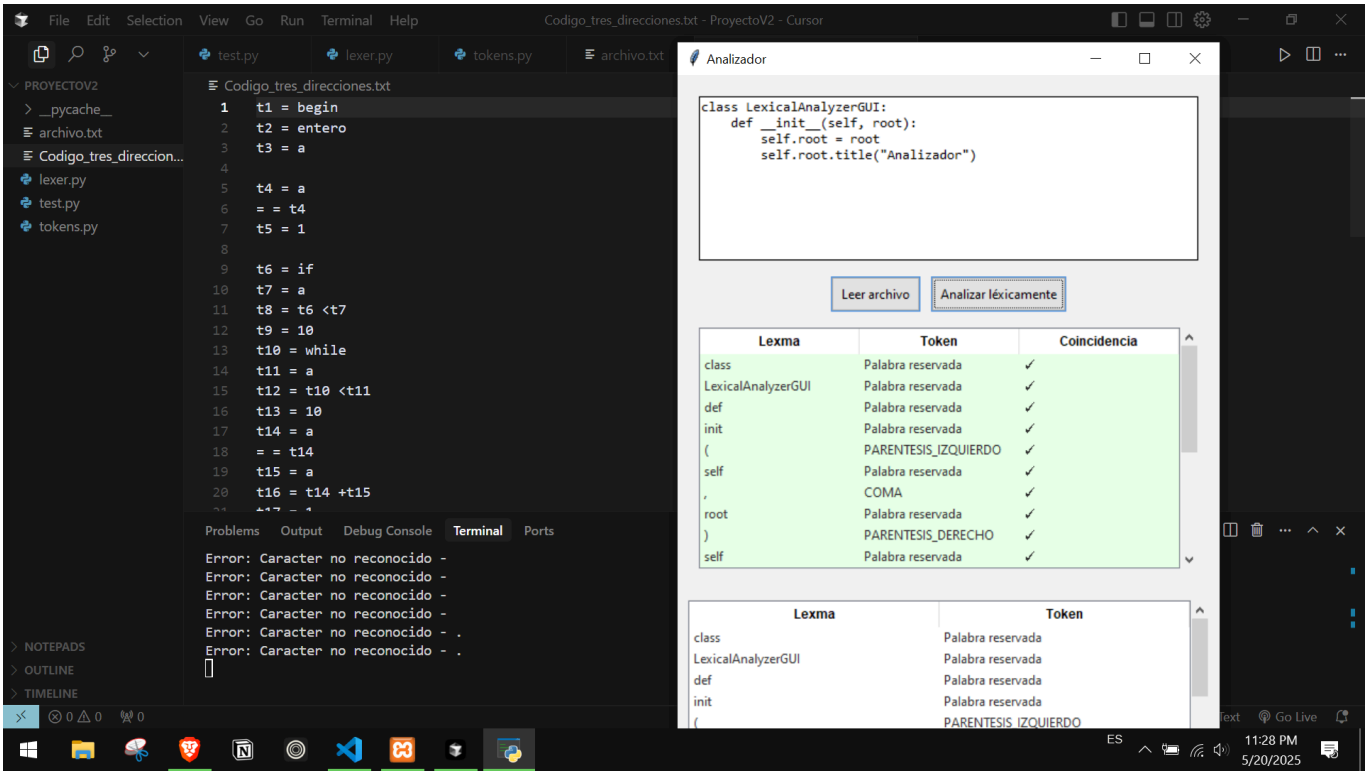
Capturas de pantalla





Lexma	Token	Coincidencia
=	ASIGNACION	✓
root	Palabra reservada	✓
self	Palabra reservada	✓
root	Palabra reservada	✓
title	Palabra reservada	✓
(	PARENTESIS_IZQUIERDO	✓
"	Comillas	✓
Analizador	Palabra reservada	✓
"	Comillas	✓
)	PARENTESIS_DERECHO	✓

Lexma	Token
root	Palabra reservada
)	PARENTESIS_DERECHO
self	Palabra reservada
root	Palabra reservada
=	ASIGNACION
root	Palabra reservada



Conclusión:

Diego Andres Hernández Rodríguez:
El desarrollo de un analizador léxico que transforma cadenas en tokens y genera código de tres direcciones no solo es un logro técnico, sino también un reflejo de cómo la humanidad busca dar sentido al caos. En esencia, este proceso imita nuestra capacidad de abstraer y organizar el mundo en unidades comprensibles, tal como lo hacemos con el lenguaje y el pensamiento.

Este proyecto no solo es una herramienta para el procesamiento de lenguaje, sino una metáfora del progreso humano. Así como el analizador léxico descompone y reconstruye, nosotros descomponemos problemas complejos y los transformamos en soluciones elegantes. Este ciclo perpetuo de análisis y síntesis es, en sí mismo, un testimonio de nuestra creatividad y nuestra búsqueda constante de trascender los límites del entendimiento.

Juan Antonio Pérez Juárez:
Uffffff, afortunadamente ya tenía muchos elementos de la práctica anterior, lo que más me quebré la cabeza por el diseño en tkinter, que afortunadamente hace 6 meses terminé un curso de python donde se enfocaba mucho en tkinter, así que a pesar de que fue difícil hacer las conexiones con las funciones como parser(), fue medianamente sencillo y con ayuda de la IA para los colores, me ayudó a no demorar mucho.
Considero que este tipo de prácticas, a corto plazo no les encuentro valor, pero supongo que adquieren valor con el paso o desarrollo de la carrera, por que a pesar de que conozco mejor el cómo funcionan los compiladores.
No me veo capaz de poder hacer un compilador desde 0 o como dicen en ingles From scratch to Compiler, quiza algunas cositas pero incluso el ASM se me complica muchísimo así que no creo dedicarme a esta rama de la programación.

Pero el tener los conocimientos me hace estar por encima de la media de los programadores y eso es una herramienta invaluable.

Para esta práctica, como lo reciclamos y solo hicimos que generase código de 3 direcciones, fue relativamente sencillo, y que funciona medianamente bien.
Y funciona, así que no le voy a mover.

Andre Maximiliano Calderon Acero

Este proyecto muestra cómo algo tan complicado como un programa puede dividirse en partes más simples para que sea más fácil de manejar. Por ejemplo, si pensamos en un problema como armar un mueble, primero identificamos las piezas (los tokens), luego seguimos las instrucciones (el código) y al final construimos algo funcional. Así, este analizador no solo organiza el caos, sino que también ayuda a construir soluciones claras y efectivas.

Agradecimientos:

Queremos expresar nuestro más sincero agradecimiento al Maestro Julio Esteban Valdéz López, Su manera relajada, amable y comprensible de enseñar hizo que esta materia, que puede ser compleja, se sintiera más accesible y llevadera. Además, su carácter afable y carismático creó un ambiente ameno en el aula.

No podemos dejar de mencionar el impacto que tuvo su famoso chiste de "aritmética venezolana", que no solo nos sacó una sonrisa, sino que también dio lugar al nombre de nuestro grupo de WhatsApp, un detalle que siempre recordaremos con cariño.

Gracias, maestro, por su paciencia, dedicación y por compartir con nosotros no solo sus conocimientos, sino también su buen humor y su humanidad. ¡Le deseamos lo mejor y esperamos que siga inspirando a más generaciones de estudiantes!

Bibliografía:

colaboradores de Wikipedia. (2020, April 16). Código de tres direcciones. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/C%C3%B3digo_de_tres_direcciones