

**UNIVERSIDAD DE SAN CARLOS DE GUATEMALA**  
**FACULTAD DE INGENIERÍA**  
**ESCUELA DE CIENCIAS Y SISTEMAS**  
**LENGUAJES FORMALES Y DE PROGRAMACIÓN**  
**SEGUNDO SEMESTRE 2023**  
**Inga. Vivian Damaris Campos González**  
**Aux. Enrique Alejandro Pinula Quiñonez**



## **Proyecto2-LFP-BizData-Manual\_Técnico**

**Nombre:** Frander Ovelto Carreto Gómez

**Carné:** 201901371

**Sección:** A-

**Fecha:** 26/10/2023

# Introducción

En un entorno tecnológico en constante evolución, la documentación técnica desempeña un papel fundamental para facilitar la comprensión, implementación y mantenimiento de sistemas y soluciones informáticas. Este manual técnico se erige como una guía esencial para aquellos profesionales y desarrolladores que buscan una referencia exhaustiva y detallada sobre el funcionamiento, configuración y uso de una aplicación o sistema específico. Con un enfoque preciso en la resolución de problemas y la optimización de procesos, este manual se convierte en una herramienta esencial en la búsqueda de la excelencia técnica.

Este manual técnico no solo se erige como una guía comprensible y precisa, sino también como un recurso de consulta constante. A medida que las tecnologías avanzan y los sistemas se actualizan, la documentación se convierte en una herramienta estratégica para mantenerse actualizado y abordar los desafíos técnicos con confianza. A lo largo de estas páginas, encontrarás la información necesaria para desbloquear el potencial de una aplicación o sistema, junto con las mejores prácticas que permiten alcanzar la excelencia en el ámbito técnico. Estamos seguros de que este manual técnico será una valiosa adición a tu repertorio de recursos de referencia técnica.

## Analizador Léxico:

El analizador léxico es la parte del código que identifica y clasifica los tokens en el código fuente. Los tokens son elementos como números, identificadores, operadores, etc. Este código utiliza una lista de patrones de expresiones regulares para identificar tokens en el código fuente. Los patrones se almacenan en la lista **patrones**, y cada patrón tiene un nombre asociado que se utiliza para clasificar el token. La función **analizar\_texto(texto)** realiza el análisis léxico y genera una lista de tokens.

```
def analizar_texto(texto):
    global tokens
    tokens = []

    lineas = texto.split('\n')
    fila = 1

    for linea in lineas:
        columna = 1
        for patron in patrones:
            expresion = re.compile(patron['patron'])
            matches = expresion.finditer(linea)

            for match in matches:
                lexema = match.group()
                nombre = patron['nombre']
                tokens.append({'nombre': nombre, 'lexema': lexema, 'fila': fila, 'columna': columna})
                columna += len(lexema)

            fila += 1

    # Generar un informe de tokens en formato HTML
    reporte_tokens = "<html><head><title>Reporte de Tokens</title></head><body><h1>Reporte de Tokens</h1>"
    reporte_tokens += "<table border='1'><tr><th>Token</th><th>Lexema</th><th>Fila</th><th>Columna</th></tr>"


    for token in tokens:
        reporte_tokens += f"<tr><td>{token['nombre']}</td><td>{html.escape(token['lexema'])}</td><td>"
        reporte_tokens += f"{token['fila']}</td><td>{token['columna']}</td></tr>"

    reporte_tokens += "</table></body></html>"

    return reporte_tokens
```

## Analizador Sintáctico:

Aunque se menciona la construcción de un árbol de derivación en la función **construir\_arbol(tokens)**, en el código proporcionado, se construye un árbol de derivación fijo y simple, que consta de un nodo raíz y tres nodos hijos que representan una expresión matemática. No se implementa un análisis sintáctico completo en este código. El árbol se grafica utilizando la biblioteca Graphviz en la función **graficar\_arbol(arbol)**.



```

def graficar_arbol(arbol):
    dot = Digraph(comment='Árbol de
Derivación')
    dot.add_node((arbol, None))

    while stack:
        nodo, padre = stack.pop()
        nombre_nodo = nodo.nombre
        if nodo.valor:
            nombre_nodo += f" ({nodo.valor})"
        dot.node(nombre_nodo)

        if padre:
            dot.edge(padre, nombre_nodo)

        for hijo in nodo.hijos:
            stack.append((hijo, nombre_nodo))

    return dot

```

### Generación de Informes:

El código proporciona la capacidad de generar informes de tokens y errores. La función **generar\_reporte\_errores()** identifica errores en el análisis léxico, como identificadores no válidos, y genera un informe en formato HTML. El informe se muestra en una ventana separada y se puede guardar como un archivo HTML en el directorio actual.

```

def generar_reporte_errores():
    global tokens
    errores = [token for token in tokens if token['nombre'] == 'IDENTIFICADOR' and not
    token['lexema'].isidentifier()]

    if not errores:
        reporte_errores = "No se encontraron errores."
    else:
        reporte_errores = "<html><head><title>Reporte de Errores</title></head><body><h1>Reporte de Errores</h1>"
        reporte_errores += "<table border='1'><tr><th>Token</th><th>Nombre</th></tr>"
        for error in errores:
            reporte_errores += f"<tr><td>{html.escape(error['lexema'])}</td><td>{error['nombre']}</td></tr>"
        reporte_errores += "</table></body></html>"

    try:
        archivo_html = os.path.join(os.getcwd(), "reporte_errores.html")
        with open(archivo_html, "w", encoding="utf-8") as f:
            f.write(reporte_errores)
    except Exception as e:
        print("Error al escribir el archivo:", e)

    ventana_reporte_errores = tk.Toplevel(ventana)
    ventana_reporte_errores.title("Reporte de Errores")

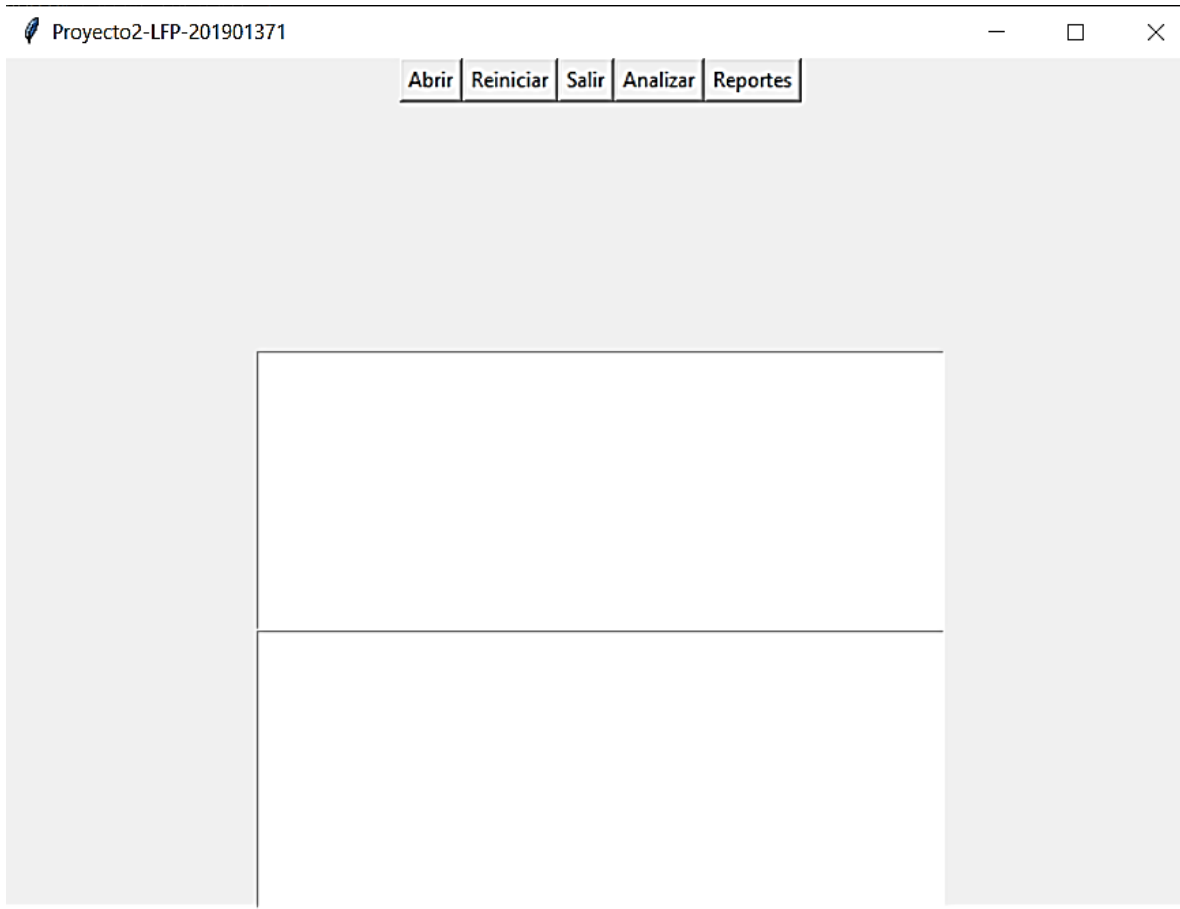
    reporte_frame = tk.Frame(ventana_reporte_errores)
    reporte_frame.pack()

    reporte_text = tk.Text(reporte_frame, height=20, width=60)
    reporte_text.tag_configure("center", justify="center")
    reporte_text.insert(tk.END, reporte_errores)
    reporte_text.pack()

```

## Interfaz Gráfica:

La interfaz gráfica se crea utilizando la biblioteca tkinter. Se define una ventana principal (**ventana**) con botones para abrir un archivo, reiniciar el programa, analizar el archivo, generar informes y salir. Dos cajas de texto (**texto1** y **texto2**) se utilizan para mostrar el contenido del archivo y los informes generados.



### **Gramática:**

El código no proporciona una gramática formal, ya que se enfoca principalmente en el análisis léxico y algunos aspectos básicos del análisis sintáctico para la generación del árbol de derivación. Para implementar un análisis sintáctico más completo, deberías definir una gramática independiente del contexto que describa la estructura del lenguaje fuente que deseas analizar y utilizarla en la construcción del árbol de derivación.

Ten en cuenta que este código es un punto de partida para un analizador léxico y sintáctico simple. Para un análisis sintáctico más avanzado y una gramática más completa, deberás expandir y ajustar el código. Además, ten en cuenta que el código proporcionado tiene limitaciones y no maneja todas las posibles situaciones léxicas o sintácticas.