Universidad Rafael Landívar
Facultad de ingeniería
Ingeniería en informática y sistemas
Lenguajes formales y autómatas, sección 02
Docente: Ing. Vivian Damaris Campos González

MANUAL TÉCNICO
Proyecto 02

Estudiantes:
González Pérez, Pablo Javier - 1211624
Miranda Abrego, Andrea Sofía - 1065824

Guatemala, 02 de noviembre del 2025

I. INTRODUCCIÓN

El proyecto tiene el objetivo de crear un analizador de operaciones aritméticas que permita realizar: suma, resta, multiplicación, división, potencia, mod y operaciones complejas (combinación de dos o más operaciones mencionadas anteriormente). Su funcionamiento se basa en hacer un análisis léxico en base a un archivo de entrada con un formato definido y en generar tres archivos siendo uno de los resultados, otro para los errores reportados y el tercero mostrando un diagrama de las operaciones realizadas.

Todo se muestra con una interfaz visual que posee un área de texto para escribir/modificar texto y siete botones con funcionalidades referentes a abrir un archivo, guardar, guardar cómo, realizar el análisis, ver los datos de los desarrolladores, ver el manual de usuario y el manual técnico. De la misma manera, se muestran los resultados en consola referentes al archivo entrada.txt. Todo el proyecto fue realizado en Python usando la librería de tkinter para el área visual y os para el acceso a procesos del sistema como los archivos.

Para explicar el desarrollo general del proyecto se realizó un manual técnico que contiene cuatro secciones además de la introducción. La primera sección es el análisis y diseño que muestra las entradas, salidas, procesos, restricciones, estructuras usadas, errores y módulos. La segunda muestra la composición de la interfaz visual. La tercera el funcionamiento del analizador léxico y la cuarta los diagramas definidos referentes al funcionamiento general el programa.

II. ANÁLISIS Y DISEÑO

2.1. ENTRADAS

- **Archivo de entrada:** para que el análisis de datos funcione, se lee el archivo de entrada seleccionado. Este archivo debe estar en un formato específico mencionado en las instrucciones del proyecto.
- **Entrada.txt:** para la consola, se usa el archivo de “entrada.txt” que lee el programa inicialmente y muestra los resultados en la consola. No está conectado de forma tan directa a la interfaz visual, es una vista previa.
- **Área de texto:** al ejecutar el programa, se presenta un área de texto en el centro que permite escribir directamente o modificar un archivo ya cargado.
- **Botones:** al ejecutar el programa, hay 7 botones que servirán para abrir un archivo de entrada, guardar, guardar cómo, analizar, ver el manual técnico, el de usuario y ayuda (datos de desarrolladores).

2.2. PROCESOS

- **Manejar archivos:** existe la posibilidad de cargar un archivo de entrada, de sobrescribir dicho archivo y de guardar un nuevo archivo basándose en lo escrito en el área de texto.
- **Analizar:** después de cargar el archivo, existe la posibilidad de analizar las operaciones aritméticas dentro de dicho archivo.
- **Ver manuales:** si se desea saber más respecto al programa, se pueden consultar los archivos manual de usuario y manual técnico presionando sus botones correspondientes.
- **DFA:** el dfa maneja y valida el formato numérico que se usará en el proceso de analizar.

2.3. SALIDAS

- **Resultados en consola**: recién se ejecuta el programa, la consola mostrará los resultados de las operaciones del archivo “entrada.txt”.
- **Resultados.html:** al ejecutar el análisis, se reciben los resultados y se crea un archivo llamado “resultados.html”. Si todo es correcto, se mostrarán los resultados, sino, se especificará que en alguna parte hay un error.
- **Errores.html:** al ejecutar el análisis, se genera un reporte de errores aparte de los resultados llamado “errores.html”. Si no hay errores el html aclarará que no hay errores, de lo contrario, se mostrarán en una tabla.
- **DiagramaOperaciones.svg:** al ejecutar el análisis, se creará un gráfico simple que mostrará la jerarquía de las operaciones matemáticas ejecutadas.

.

2.4. RESTRICCIONES

- **Pre-requisitos:** se necesita de un archivo de entrada para que el analizador funcione.

- **Salida de reportes:** las salidas de resultados y errores deben ser en formato HTML.
- **Salida de diagrama de operaciones:** la salida del diagrama debe ser en formato png o svg.
- **Errores:** se deben manejar errores sin detener la ejecución.
- **Interfaz gráfica:** la interfaz gráfica debe tener las mismas funcionalidades que pide la plantilla del archivo del proyecto (Abrir, guardar, guardar cómo, analizar, ayuda, manual de usuario, manual técnico). De la misma manera debe tener un área de texto para escribir o cargar un archivo.
- **Analizar léxico:** este debe seguir la estructura ya definida en la documentación, se debe seguir el dfa y la tabla de transiciones.
- **Librerías externas:** no se pueden usar librerías externas, solo propias de Python para el área visual, no para facilitar el análisis.

2.5. ESTRUCTURAS USADAS

- **Estructuras de datos:** listas, diccionarios

```
errores = []
pila_ops = []   # cada item: {"tipo":..., "nums":[], "tiene_subop":False}
historico = []  # para imprimir en el orden de cierre
```

```
OPERACIONES_PERMITIDAS = [
    "SUMA","RESTA","MULTIPLICACION","DIVISION","POTENCIA","RAIZ","INVERSO","MOD"
]
```

Listas

```
def simbolo(t):
    return {
        "SUMA": "+",
        "RESTA": "-",
        "MULTIPLICACION": "*",
        "DIVISION": "/",
        "POTENCIA": "^",
        "MOD": "%"
    }.get(t, None)
```

Diccionario

- **Estructuras de control:** while, for, if, else, elif, pila, DFA, try, except

```
while i < n:
    i = saltar_espacios(texto, i)
    if i >= n: break

    # <Operacion= OP>
    if texto.startswith("<Operacion", i):
        i += len("<Operacion")
        i = saltar_espacios(texto, i)
        if i >= n or texto[i] != '=':
            errores.append(("Falta '=' en <Operacion= ... >", i)); break
```

```
def dfa_numero(cadena):
    """DFA: [+-]?[0-9]+ ('.' [0-9]+)?  -> True/False"""
    i, n = 0, len(cadena)
```

```
else:
    if not pila_ops:
        errores.append(("Número fuera de una <Operacion>", i))
```

```
simb = simbolo(tipo)
if simb and len(nums) >= 2:
    print(f"{nums[0]}{simb}{nums[1]} = {round(res,2)}\n")
elif tipo == "RAIZ" and len(nums) >= 1:
    print(f"√({nums[0]}) = {round(res,2)}\n")
```

```
pila_ops.append({"tipo": op, "nums": [], "tiene_subop": False})
continue
```

```
for op in historico:
    tipo, nums, res, es_compleja = op["tipo"], op["nums"], op["res"], op["compleja"]
```

```
try:
    nombre = "Manual de usuario_proyecto2.pdf"
    if os.path.exists(nombre):
        os.startfile(nombre)
```

```
except Exception as e:
    messagebox.showerror("Error", f"No se pudo abrir el manual de usuario:{str(e)}"
```

- **Estructuras de clase: clase** interfazGrafica

```
#interfaz grafica
class interfazGrafica:
#ventana inicial, iniciación de los botones y área de texto
    def __init__(self):
        self.ventana = tk.Tk()
        self.ventana.title("Analizador de Operaciones Aritméticas")
        self.ventana.geometry("620x600")
        self.ventana.config(bg="#C4CDD2")

        self.archivoActual = None
        self.botones()
        self.Texto()
```

2.6. ERRORES

- Lista que almacena errores durante el análisis

```
errores = []
```

- Verificación de que el tipo de operaciones sea válido.

```
op, i = leer_identificador(texto, i)
if op not in OPERACIONES_PERMITIDAS:
    errores.append((f"Operación desconocida '{op}'", i))
```

- Marca errores léxicos como el formato del número.

```
numtxt = contenido.strip()
if not dfa_numero(numtxt):
    errores.append((f"Número mal formado '{numtxt}'", i))
```

- Detecta si alguna operación se dejó sin cerrar

```
if pila_ops:
    errores.append(("Etiqueta(s) sin cerrar: Operacion", n))
```

- Detecta si alguna operación no tiene una apertura correspondiente.

```
if texto.startswith("</Operacion>", i):
    if not pila_ops:
        errores.append(("Cierre </Operacion> inesperado", i))
        i += len("</Operacion>")
        continue
```

- Maneja errores en la interfaz gráfica, se usa un formato en el que se muestra una ventana emergente con el error,

```
except Exception as e:
    messagebox.showerror("Error", f"No se pudo guardar el archivo: {str(e)}")
```

2.7. MÓDULOS

- **tkinter:** se importa tkinter para toda la interfaz gráfica. Dentro de tkinter también se importan “filedialog, messagebox” como componentes para los mensajes/advertencias emergentes.
- **os:** permite interactuar con archivos del sistema, ayuda a saber el nombre de los archivos y a recogerlos para su uso.

III. ANÁLISIS LÉXICO

A continuación, se muestran los fragmentos de código en los que se trabaja el análisis léxico del proyecto.

1. Operaciones permitidas:

```
OPERACIONES_PERMITIDAS = [
    "SUMA","RESTA","MULTIPLICACION","DIVISION","POTENCIA","RAIZ","INVERSO","MOD"
]
```

En esta parte se definen todas las operaciones aritméticas que permite el lenguaje. Se definen por medio de palabras reservadas.

2. Definición de caracteres:

```
def es_digito(ch):
    return '0' <= ch <= '9'

def es_espacio(ch):
    return ch in ' \t\r\n'
```

Luego de definir las palabras reservadas, se definen dos funciones que ayudan con el reconocimiento de caracteres. La función es_digito define si el carácter es un digito del 0 al 9 (así se evita el uso de letras) y la función es_espacio ayuda a saltar espacios.

3. DFA

```
def dfa_numero(cadena):
    """DFA: [+-]?[0-9]+ ('.' [0-9]+)?  -> True/False"""
    i, n = 0, len(cadena)
    if n == 0:
        return False #cadena vacia
    if cadena[i] in "+-":
        i += 1 #siguiente posicion
        if i >= n:
            return False #no tiene numeros
    if i >= n or not es_digito(cadena[i]):
        return False #debe de continuar con un digito
    while i < n and es_digito(cadena[i]):
        i += 1 #consumo todos los digitos
    if i < n and cadena[i] == '.': #decimales
        i += 1 #salta la poscion del decimal
        if i >= n or not es_digito(cadena[i]):
            return False #debe de seguir numeros despues del decimal
        while i < n and es_digito(cadena[i]):
            i += 1 #consume todos los decimales
    return i == n #retorna si la cadena entera es correcta
```

En esta parte se implementa el DFA creado antes de la realización del código. El DFA valida los números y los símbolos permitidos que sirven para las operaciones

aritméticas (+ y – por ejemplo), se verifica si son dígitos numéricos, decimales y se continua si se valida todo.

4. Funciones que ayudan al análisis léxico:

```python
def saltar_espacios(texto, i):
    n = len(texto)
    while i < n and es_espacio(texto[i]): #verifica que sea un espacio en blanco y los salta
        i += 1
    return i

def leer_identificador(texto, i):
    n = len(texto)
    j = i
    while j < n and (texto[j].isalpha() or texto[j] == '_'):
        j += 1
    return texto[i:j], j #retorna el identificador leido y la nueva posicion a continuar

def leer_hasta(texto, i, token):
    j = texto.find(token, i)#manda a buscar en el texto el token a partir de i
    if j == -1: #si no existe el token retorna nada y el final del texto
        return None, len(texto)
    return texto[i:j], j + len(token) #si lo cuencuentra devuelve la posicion despues del token
```

La primera función se encarga de saltar los espacios, la segunda reconoce el tipo de operación leyendo el identificador y la tercera ayuda a buscar tokens dentro del texto.

5. Símbolos de operaciones:

```python
def simbolo(t):
    return {
        "SUMA": "+",
        "RESTA": "-",
        "MULTIPLICACION": "*",
        "DIVISION": "/",
        "POTENCIA": "^",
        "MOD": "%"
    }.get(t, None)
```

En esta función se hace una conversión de las palabras reservadas a sus símbolos matemáticos correspondientes para cuando se necesite realizar las operaciones matemáticas.

6. Procesamiento de tokens y errores

```python
i, n = 0, len(texto)
errores = []
pila_ops = []   # cada item: {"tipo":..., "nums":[], "tiene_subop":False}
historico = []  # para imprimir en el orden de cierre

while i < n:
    i = saltar_espacios(texto, i)
    if i >= n: break

    # <Operacion= OP>
    if texto.startswith("<Operacion", i):
        i += len("<Operacion")
        i = saltar_espacios(texto, i)
        if i >= n or texto[i] != '=':
            errores.append(("Falta '=' en <Operacion= ... >", i)); break
        i += 1
        i = saltar_espacios(texto, i)
        op, i = leer_identificador(texto, i)
        if op not in OPERACIONES_PERMITIDAS:
            errores.append((f"Operación desconocida '{op}'", i))
        i = saltar_espacios(texto, i)
        if i >= n or texto[i] != '>':
            errores.append(("Falta '>' en <Operacion= ... >", i)); break
        i += 1
        pila_ops.append({"tipo": op, "nums": [], "tiene_subop": False})
        continue
```

```
    # <Numero> ... </Numero>
    if texto.startswith("<Numero>", i):
        i += len("<Numero>")
        contenido, i2 = leer_hasta(texto, i, "</Numero>")
        if contenido is None:
            errores.append(("Falta </Numero>", i)); break
        numtxt = contenido.strip()
        if not dfa_numero(numtxt):
            errores.append((f"Número mal formado '{numtxt}'", i))
        else:
            if not pila_ops:
                errores.append(("Número fuera de una <Operacion>", i))
            else:
                pila_ops[-1]["nums"].append(float(numtxt))
        i = i2
        continue

    # cualquier otro carácter: avanzar
    i += 1

if pila_ops:
    errores.append(("Etiqueta(s) sin cerrar: Operacion", n))
```

Dentro del main() se realiza el procesamiento de los tokens iniciando por la lectura y validación de las operaciones según el formato. Una vez se valide lo inicial, se usa el dfa para validar los números y asociarlos a su operación válida correspondiente. Al finalizar, se verifica si hay etiquetas sin cerrar. Este mismo procesamiento se aplica en la interfaz gráfica, dicha función está basada en la del main().

IV. INTERFAZ VISUAL

A continuación, se muestran los fragmentos de código en los que se trabaja la interfaz visual del proyecto.

1. Ventana principal

```
#interfaz grafica
class interfazGrafica:
#ventana inicial, iniciación de los botones y área de texto
    def __init__(self):
        self.ventana = tk.Tk()
        self.ventana.title("Analizador de Operaciones Aritméticas")
        self.ventana.geometry("620x600")
        self.ventana.config(bg="#C4CDD2")

        self.archivoActual = None
        self.botones()
        self.Texto()
```

Se define inicialmente la ventana principal del programa, colocando el título, sus dimensiones y el color. Así mismo, se invocan elementos a usar como la definición inicial de que al ejecutar el programa no hay ningún archivo cargado y las funciones de botones() y texto() para mostrarlos en la ventana principal.

2. Área de texto

```
#área de texto
    def Texto(self):
        self.texto = tk.Text(self.ventana, wrap=tk.WORD, width=60, height=25)
        self.texto.grid(row=1, column=0, columnspan=10, padx=10, pady=10, sticky="nswe")
```

Se define el área de texto donde se puede escribir o modificar un archivo.txt.

3. Botones

```
#botones que ejecutan cada una de las 7 acciones.
    def botones(self):
        self.boton1 = tk.Button(self.ventana, text="Abrir", command=self.abrirArchivo)
        self.boton1.config(bg='#FFFFFF', fg="black",borderwidth=2, relief="raised",width=16)
        self.boton1.grid(row=0, column=0, padx=10, pady=10)

        self.boton2 = tk.Button(self.ventana, text="Guardar", command=self.guardarArchivo)
        self.boton2.config(bg='#FFFFFF', fg="black",borderwidth=2, relief="raised",width=16)
        self.boton2.grid(row=0, column=1, padx=10, pady=10)

        self.boton3 = tk.Button(self.ventana, text="Guardar como", command=self.guardarComoArchivo)
        self.boton3.config(bg='#FFFFFF', fg="black",borderwidth=2, relief="raised",width=16)
        self.boton3.grid(row=0, column=2, padx=10, pady=10)

        self.boton4 = tk.Button(self.ventana, text="Analizar", command=self.analizar)
        self.boton4.config(bg='#FFFFFF', fg="black",borderwidth=2, relief="raised",width=16)
        self.boton4.grid(row=0, column=3, padx=10, pady=10)

        self.boton5 = tk.Button(self.ventana, text="Manual de usuario", command=self.manualUsuario)
        self.boton5.config(bg='#FFFFFF', fg="black",borderwidth=2, relief="raised",width=18)
        self.boton5.grid(row=2, column=2, padx=10, pady=10)

        self.boton6 = tk.Button(self.ventana, text="Manual técnico", command= self.manualTecnico)
        self.boton6.config(bg='#FFFFFF', fg="black",borderwidth=2, relief="raised",width=18)
        self.boton6.grid(row=2, column=1, padx=10, pady=10)

        self.boton7 = tk.Button(self.ventana, text="Ayuda", command=self.ayuda)
        self.boton7.config(bg='#FFFFFF', fg="black",borderwidth=2, relief="raised",width=18)
        self.boton7.grid(row=2, column=0, padx=10, pady=10)
```

Se definen los 7 botones con su diseño y función correspondiente.

4. Abrir archivo

```python
#permite abrir un archivo de texto en formato txt
    def abrirArchivo(self):
        try:
            filepath = filedialog.askopenfilename(
                title="Seleccionar archivo",
                filetypes=[("Archivos de texto", "*.txt"), ("Todos los archivos", "*.*")]
            )

            if filepath:
                with open(filepath, 'r', encoding='utf-8') as file:
                    contenido = file.read()
                self.texto.delete(1.0, tk.END)
                self.texto.insert(1.0, contenido)
                self.archivoActual = filepath

                messagebox.showinfo("Éxito", f"Se ha cargado el archivo: {os.path.basename(filepath)}")

        except Exception as e:
            messagebox.showerror("Error", f"No se pudo abrir el archivo: {str(e)}")
```

Se define la ventana de seleccionar archivo en el formato txt. Si bien acepta formatos html y svg porque pueden ser leídos por el área de texto, la prioridad son los archivos de texto. Aquí se le da uso a la variable archivoActual iniciada en la función que define la ventana.

5. Manual de usuario

```python
#abre el manual de usuario de forma externa
    def manualUsuario(self):
        try:
            nombre = "Manual de usuario_proyecto2.pdf"
            if os.path.exists(nombre):
                os.startfile(nombre)
            else:
                messagebox.showwarning("Error: no se encontró el manual de usuario",
                                       f"Verificar si el archivo se encuentra en la misma carpeta del programa o si es de nombre: {nombre}"
                                       )
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo abrir el manual de usuario:{str(e)}")
```

Se abre el archivo referente al manual de usuario del proyecto, este se abre de forma externa como en un navegador. Si no se encuentra el archivo dentro de la carpeta, dará error.

6. Manual técnico

```python
#abre el manual técnico de forma externa
    def manualTecnico(self):
        try:
            nombre = "Manual técnico_proyecto2.pdf"
            if os.path.exists(nombre):
                os.startfile(nombre)
            else:
                messagebox.showwarning("Error: no se encontró el manual técnico",
                                       f"Verificar si el archivo se encuentra en la misma carpeta del programa o si es de nombre: {nombre}")
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo abrir el manual técnico:{str(e)}")
```

Se abre el archivo referente al manual técnico del proyecto, este se abre de forma externa como en un navegador. Si no se encuentra el archivo dentro de la carpeta, dará error.

7. Guardar cómo

```
#guardar lo que está escrito en el área de texto con un nombre diferente a la entrada
    def guardarComoArchivo(self):
        try:
            contenido = self.texto.get(1.0, tk.END)
            filepath = filedialog.asksaveasfilename(
                title="Guardar como",
                defaultextension=".txt",
                filetypes=[("Archivos de texto txt", "*.txt"), ("Todos los archivos", "*.*")]
            )
            if filepath:
                with open(filepath, "w", encoding="utf-8") as archivo:
                    archivo.write(contenido)
                self.archivoActual = filepath

                messagebox.showinfo("Éxito", f"El archivo se guardó como: {os.path.basename(filepath)}")

        except Exception as e:
            messagebox.showerror("Error", f"No se pudo guardar el archivo: {str(e)}")
```

Se define que se abrirá la ventana del explorador de archivos para guardar, se proporciona la extensión .txt como prioridad y se crea un nuevo archivo. Aquí se le da uso a la variable archivoActual iniciada en la función que define la ventana.

8. Guardar

```
#guardar lo que está escrito en el área de texto sobreescribieno el archivo de entrada
    def guardarArchivo(self):
        try:
            contenido = self.texto.get(1.0, tk.END)
            if not hasattr(self, 'archivoActual') or not self.archivoActual:
                messagebox.showwarning("Advertencia", "No hay un archivo abierto que se pueda guardar\n" "Primero usar guardar cómo")
                return
            with open(self.archivoActual, "w", encoding="utf-8") as archivo:
                archivo.write(contenido)

            messagebox.showinfo("Éxito", f"Archivo guardado: {os.path.basename(self.archivoActual)}")

        except Exception as e:
            messagebox.showerror("Error", f"No se pudo guardar el archivo: {str(e)}")
```

Se verifica si ya existe un archivo cargado. Si no existe se pide que se cargue un archivo primero y si ya se cargó, se sobrescribe el archivo de entrada cargado. Aquí se le da uso a la variable archivoActual iniciada en la función que define la ventana.

9. Ayuda

```
#muestra los datos de los desarrolladores
    def ayuda(self):
        messagebox.showinfo("Datos de los desarrolladores",
                            "Curso: Lenguajes Formales y Automatas\n"
                            "sección: 02\n" "-----------------estudiantes-----------------\n"
                            "González Pérez, Pablo Javier - 1211624\n"
                            "Miranda Abrego, Andrea Sofía - 1065824\n")
```

Se definen los datos de los integrantes del grupo en una ventana emrgente.

10. Ejecutar

```
#ejecuta como tal la ventana, permite que la ventana siga en ejecución
    def ejecutar(self):
        self.ventana.mainloop()

if __name__ == "__main__":
    main()

app = interfazGrafica()
app.ejecutar()
```

Se define el loop en el que se ejecutará el programa.

11. Analizar

```
#sección correspondiente al botón de analizar. Se procesa el texto usando información del main(), se crean los archivos html de resultado y errores junto con el diagrama de operaciones svg creado con xml
    def analizar(self):
        texto = self.texto.get("1.0", tk.END)

        if not texto.strip():
            messagebox.showwarning("Advertencia", "No se ha cargado un archivo en el área de texto")
            return
        try:
            resultado = self.procesarTexto(texto)
            self.crearHTML(resultado)
            self.crearDiagramaSVG(resultado)
            messagebox.showinfo("Analizar completado",
                                "El análisis se completó exitosamente\n"
                                "Archivos generados:\n"
                                "Resultados.html\n"
                                "Errores.html\n"
                                "diagramaOperaciones.svg")

        except Exception as e:
            messagebox.showerror("Error", f"Error al analizar: {str(e)}")

    def procesarTexto(self, texto):
        i, n = 0, len(texto)
        errores = []
        pilaOps = []
        historico = []
        contador_global = 0
```

Conformado por dos funciones (analizar y procesarTexto), se realiza el análisis general de las operaciones aritméticas. En dicha parte se usa el código del main() del analizador léxico para procesar el texto y luego generar los resultados, errores y el diagrama. Si no se ha cargado ningún archivo se avisa al respecto y si sucede un error, aparece un mensaje. (por el espacio no se coloca el código completo).

12. Crear Diagrama SVG

```
def crearDiagramaSVG(self, resultado):
    if "historico" not in resultado or not resultado["historico"]:
        return

    operaciones = resultado["historico"]
    ancho = 800
    alto = len(operaciones) * 100 + 100
    margen = 50

    svgDiagrama = f'''<?xml version="1.0" encoding="UTF-8"?>
    <svg width="{ancho}" height="{alto}" xmlns="http://www.w3.org/2000/svg">
    <rect width="100%" height="100%" fill="white"/>
    <text x="{ancho//2}" y="30" font-family="Arial" font-size="16" font-weight="bold" text-anchor="middle">Diagrama de jerarquía de operaciones</text>'''
    y_actual = margen + 50
```

Sin usar librerías externas como pide los requisitos del proyecto se genera el diagrama en formato .svg usando directamente xml. (por el espacio no se coloca el código completo).

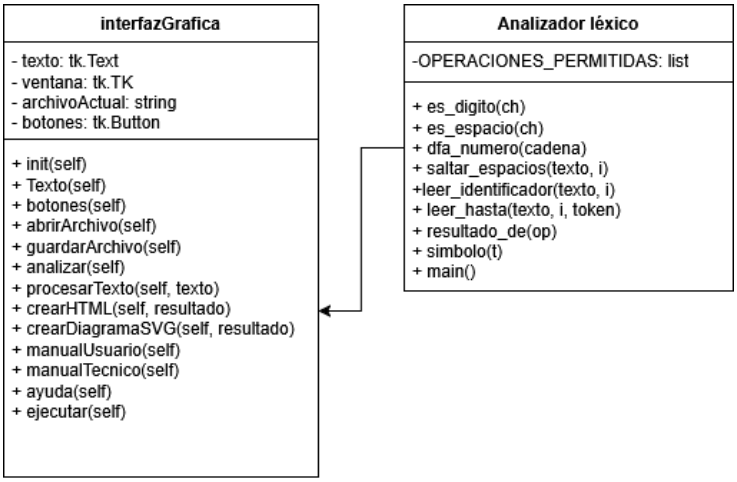
13. Crear HTML

```
def crearHTML(self, resultado):
    with open("Resultados.html", "w", encoding="utf-8") as archivo:
        archivo.write("<html><body>\n")
        archivo.write("<h1>Resultados del Analizador de Operaciones Aritméticas</h1>\n")
```

Se crea el formato de los archivos de resultado y errores en formato html que se generan al ejecutar el análisis en el programa. (por el espacio no se coloca el código completo).

V. DIAGRAMAS

1. Diagrama de clases



2. Diagrama de flujo

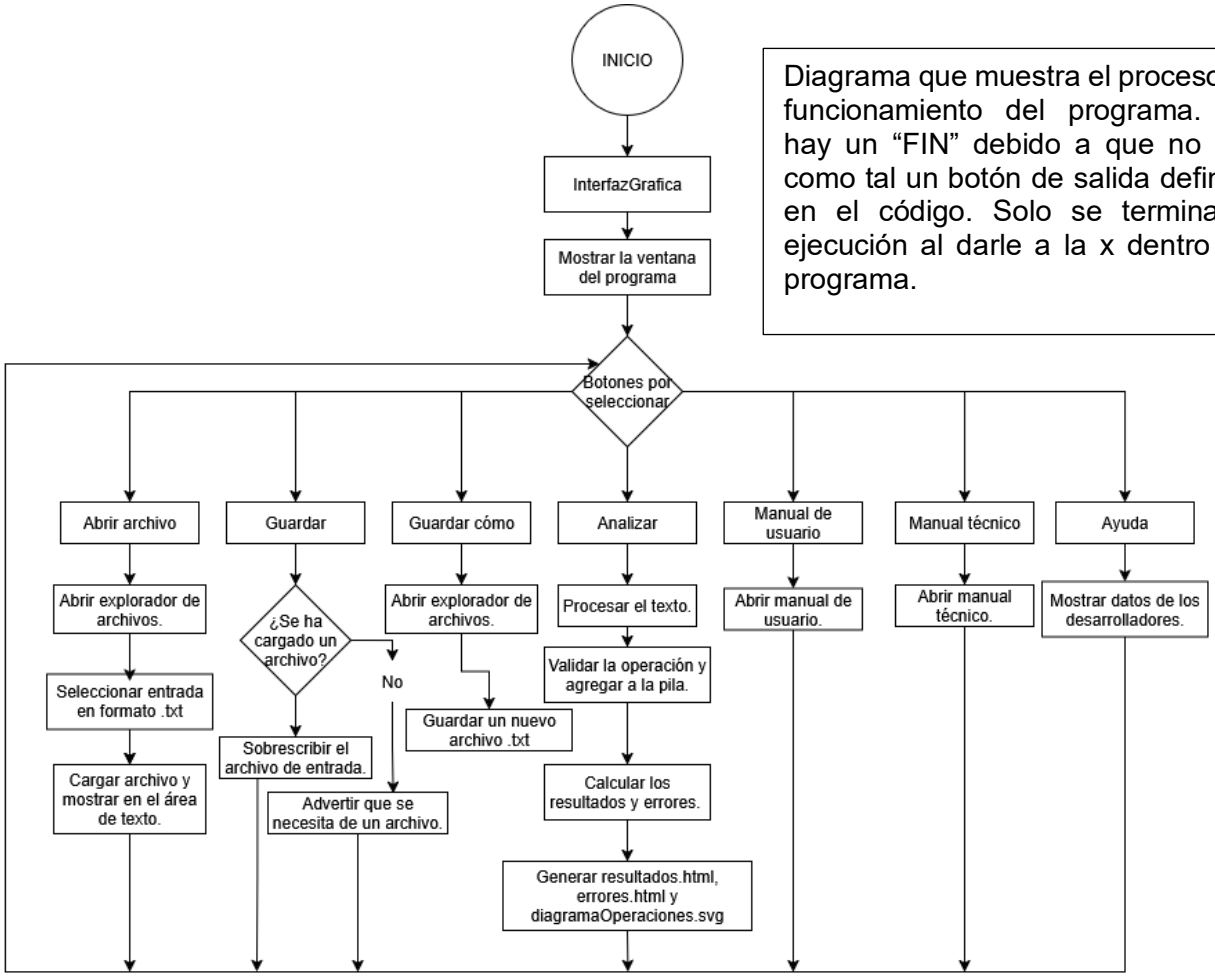


Diagrama que muestra el proceso de funcionamiento del programa. No hay un “FIN” debido a que no hay como tal un botón de salida definido en el código. Solo se termina la ejecución al darle a la x dentro del programa.