PROYECTO 1 LFP

Manual Técnico

USAC – Facultad de ingeniería
Damián Ignacio Peña Afre

CONTENIDO

ENCABEZADO

- Nombre del sistema: Analizador Léxico
- Desarrollador: Damián Ignacio Peña Afre
- Lenguaje utilizado: Python
- Tipo de programa: Desktop
- Cuenta con interfaz gráfica

TECNICAS O PARADIGMAS

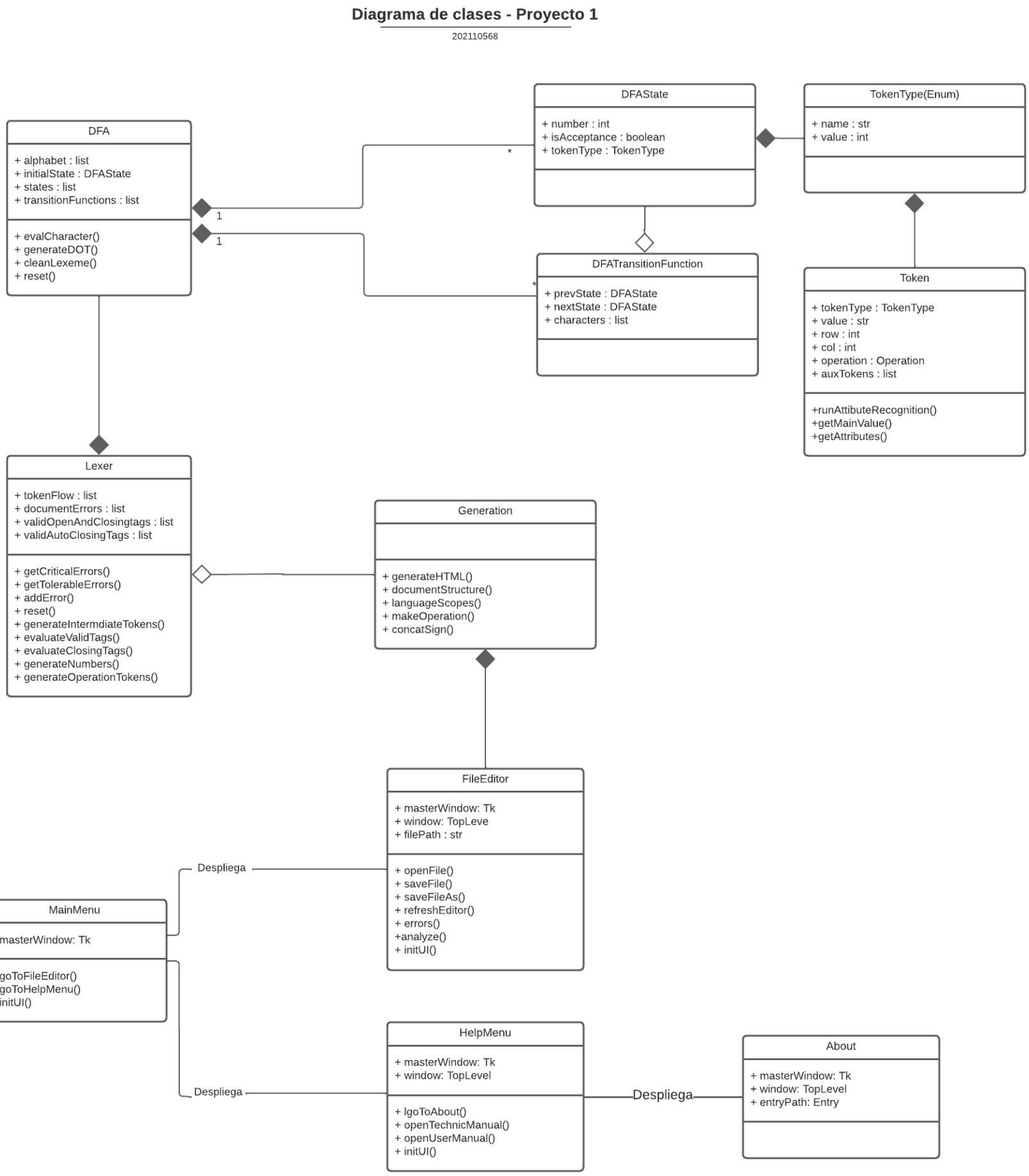
Se utilizo el paradigma de la programación orientada a objetos junto con la separación de los archivos en carpetas en función del tipo de tarea que realizan, de esta manera el proyecto esta segmentado de una forma modular.

Se separó la lógica del aspecto visual, de modo que obtiene una estructura más limpia de código.

CONVENCIONES DE NOMENCLATURA

Se optó por utilizar '*camelcase*' para nombrar los archivos y variables. Tanto el nombre de la clase como el archivo que lo contiene empiezan con letra mayúscula. Los métodos/funciones y archivos que contienen únicamente métodos/funciones empiezan por letra minúscula.

DIAGRAMA DE CLASES



METODOS PRINCIPALES

1. evalCharacter

Método de la clase DFA utilizado para evaluar los caracteres de una cadena, es llamado cada vez que se requiera evaluar un nuevo carácter, internamente guarda el estado actual del autómata y si existe la posibilidad de recorrer hacia otro nodo. Asimismo, determina y retorna un token valido para que posteriormente se inserte al flujo de tokens

```
    def evalCharacter(self, character, isCharacterNext, row, col):

        # * Not in alphabet, not valid character
        if character not in self.alphabet:
            # ? Character
            return Token(TokenType.ERROR, character, row, col)

        # * Looks for next state

        for function in self.transitionFunctions:

            # * Transition function related to actual state
            if function.prevState == self.actualState:

                # * Able to go to next state
                if character in function.characters:
                    self.actualState = function.nextState

                    self.lexeme += character

                    # * There is no character next
                    if not isCharacterNext:
                        # * Eval if new state is acceptance
                        if self.actualState.isAcceptance:
                            # * VALID TOKEN
                            self.validCharacter = True
                            return Token(self.actualState.tokenType, self.cleanLexeme(), row, col)
                        else:
                            # * Incomplete lexema
                            return Token(TokenType.ERROR, self.lexeme, row, col)

                    # * New state set, keeps running
                    return True

        # * No related function

        # * Check if actual state is acceptance
        if self.actualState.isAcceptance:
            # * VALID TOKEN
            self.validCharacter = False
            return Token(self.actualState.tokenType, self.cleanLexeme(), row, col)

        # * No valid lexeme
        self.lexeme += character
```

2. concatSign

Método utilizado para concatenar el signo correspondiente a la operación que se requiere ejecutar.

```python
@ staticmethod
    def concatSign(operationToken: Token, rightNumber: str):
        operation = operationToken.operation.operationType
        rightNumber = str(rightNumber)

        if operation == OperationType.SUMA:
            return f"+{rightNumber}"
        elif operation == OperationType.RESTA:
            return f"-{rightNumber}"
        elif operation == OperationType.MULTIPLICACION:
            return f"*{rightNumber}"
        elif operation == OperationType.DIVISION:
            return f"/{rightNumber}"
        elif operation == OperationType.POTENCIA:
            return f"**{rightNumber}"
        elif operation == OperationType.RAIZ:
            return f"**(1/({rightNumber}))"
        elif operation == OperationType.MOD:
            return f"%{rightNumber}"
        elif operation == OperationType.SENO:
            return f"math.sin({rightNumber})"
        elif operation == OperationType.COSENO:
            return f"math.cos({rightNumber})"
        elif operation == OperationType.TANGENTE:
            return f"math.tan({rightNumber})"
        elif operation == OperationType.INVERSO:
            return f"1/({rightNumber})"
        else:
            # ! ERROR
            print(operation)
            pass
```

3. MakeOperation

Método recursivo que crea la cadena que representará la operación final a ejecutar, el método es llamado cuando detecta una operación dentro de la etiqueta Tipo

```
@staticmethod
    def makeOperation(tokenNumber: int):

        tokens = Lexer.tokenFlow
        operationToken = tokens[tokenNumber]
        operationType = operationToken.operation.operationType

        operationTokenNumberOfClosingTag = getClosingTagNumber(tokenNumber+1)

        # first number
        operation = ""
        n = tokenNumber+2
        firstNumberToken = tokens[tokenNumber+1]

        # If first token is number
        if firstNumberToken.tokenType == TokenType.NUMBER:
            # One number operation
            if operationType == OperationType.SENO or operationType == OperationType.COSENO or
operationType == OperationType.TANGENTE or operationType == OperationType.INVERSO:
                return Generation.concatSign(
                    operationToken, str(firstNumberToken.value))
            else:
                operation = str(firstNumberToken.value)

        # If first token is operation
        elif firstNumberToken.tokenType == TokenType.OPEN_TAG and firstNumberToken.operation:
            if operationType == OperationType.SENO or operationType == OperationType.COSENO or
operationType == OperationType.TANGENTE or operationType == OperationType.INVERSO:
                return "("+Generation.concatSign(operationToken,
str(Generation.makeOperation(tokenNumber+1)))+")"
            else:
                operation = "("+Generation.makeOperation(tokenNumber+1)+")"
            # resets the where start
            n = getClosingTagNumber(tokenNumber+2)+1

        # starts on second number
        while n < operationTokenNumberOfClosingTag:
            nextNumber = tokens[n]
            if nextNumber.tokenType == TokenType.NUMBER:
                operation += Generation.concatSign(
                    operationToken, nextNumber.value)
            elif nextNumber.tokenType == TokenType.OPEN_TAG and nextNumber.operation:
                partialResult = Generation.makeOperation(n)
                operation += Generation.concatSign(
                    operationToken, "("+partialResult+")")
                n = getClosingTagNumber(n+1)+1
                continue
            n += 1

        return operation
```

4. CreateStr

Método utilizado para crear la cadena que representará el archivo html de salida. Son utilizados métodos dentro de la misma clase para estructurar de mejor manera el contenido

```
    def createStr(self):
        self.htmlStr = f"""
<html>
    <head>
        <title>{self.createTitle()}</title>
    </head>
    <body>
        <header>
            <h1 style="{self.createStyles('ESTILOS_TITULO')}">{self.createTitle()}</h1>
            {self.createDescription()}
        </header>
        <main>
{self.createContent()}
        </main>
    </body>
</html>
"""
```

REQUERIMIENTOS

- Contar con un menú de opciones
- Cargar un archivo con extensión CSV o LFP
- Editar el archivo fuente
- Analizar el archivo fuente
- Mostrar los errores del archivo fuente
- Contar con un menú de ayuda

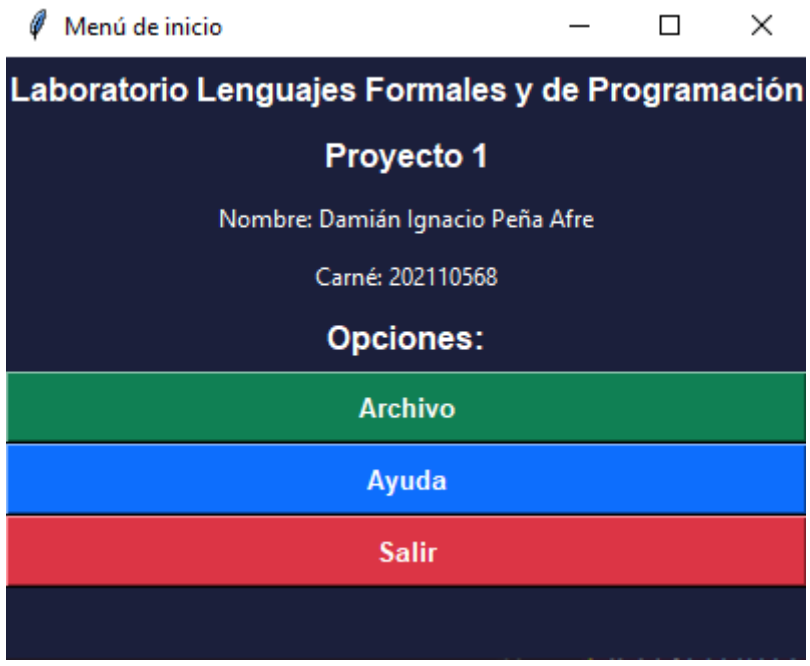
DESCRIPCIÓN

Para la realización del proyecto se utilizaron las siguientes herramientas:

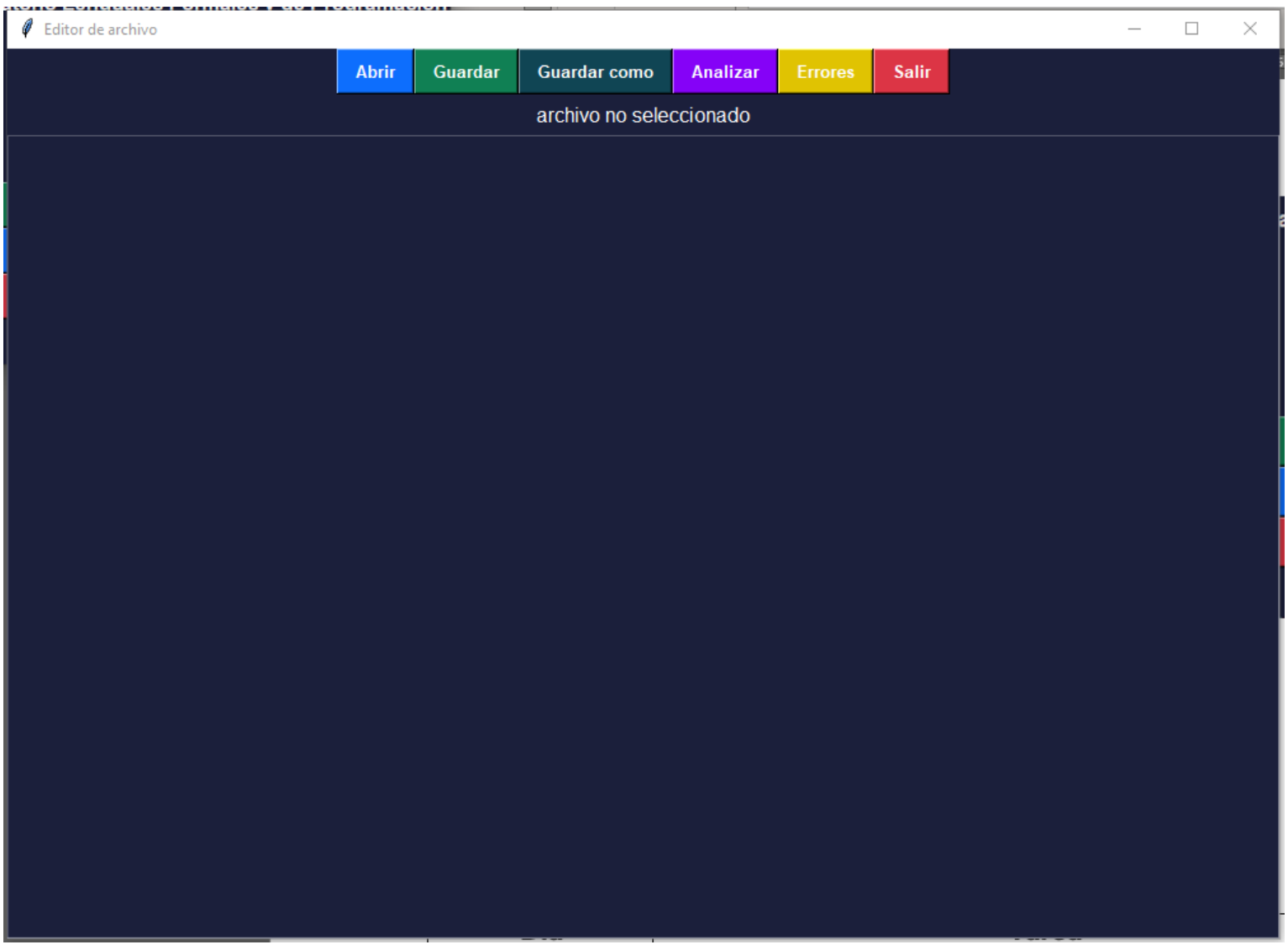
1. Visual Studio Code: Editor de código ligero.
2. Git: Programa para llevar un control de las versiones del proyecto.
3. Github: Repositorio remoto para llevar un control de las versiones del proyecto en la nube.

INTERFACES PRINCIPALES

1. *Menu principal*

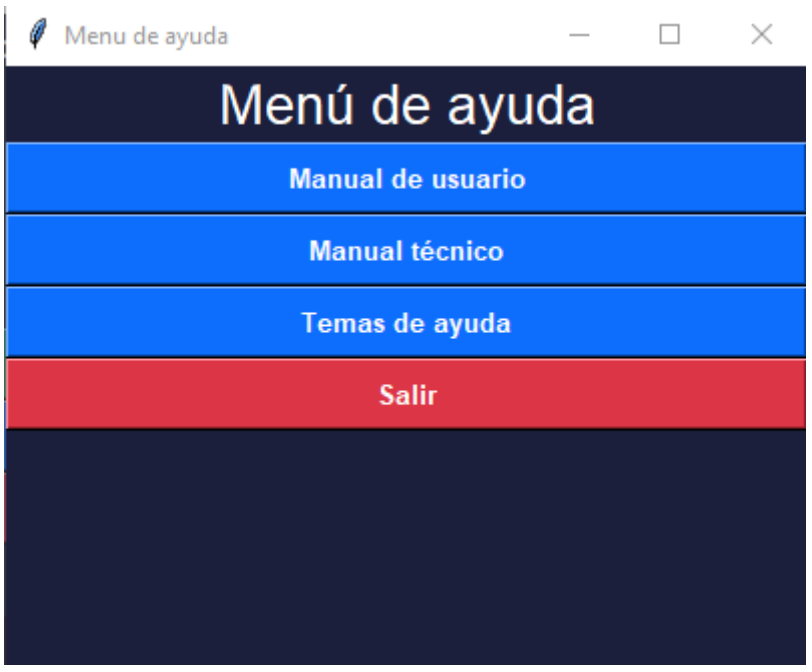


2. *Editor de archivo*



Se toma el contenido dentro del área de texto para sobrescribir o crear un archivo. Es posible también analizar el contenido de dicho archivo con el botón de analizar, en dado caso se presenten errores, es posible también, presionar el botón de errores para generar dicho reporte.

3. Menú de ayuda



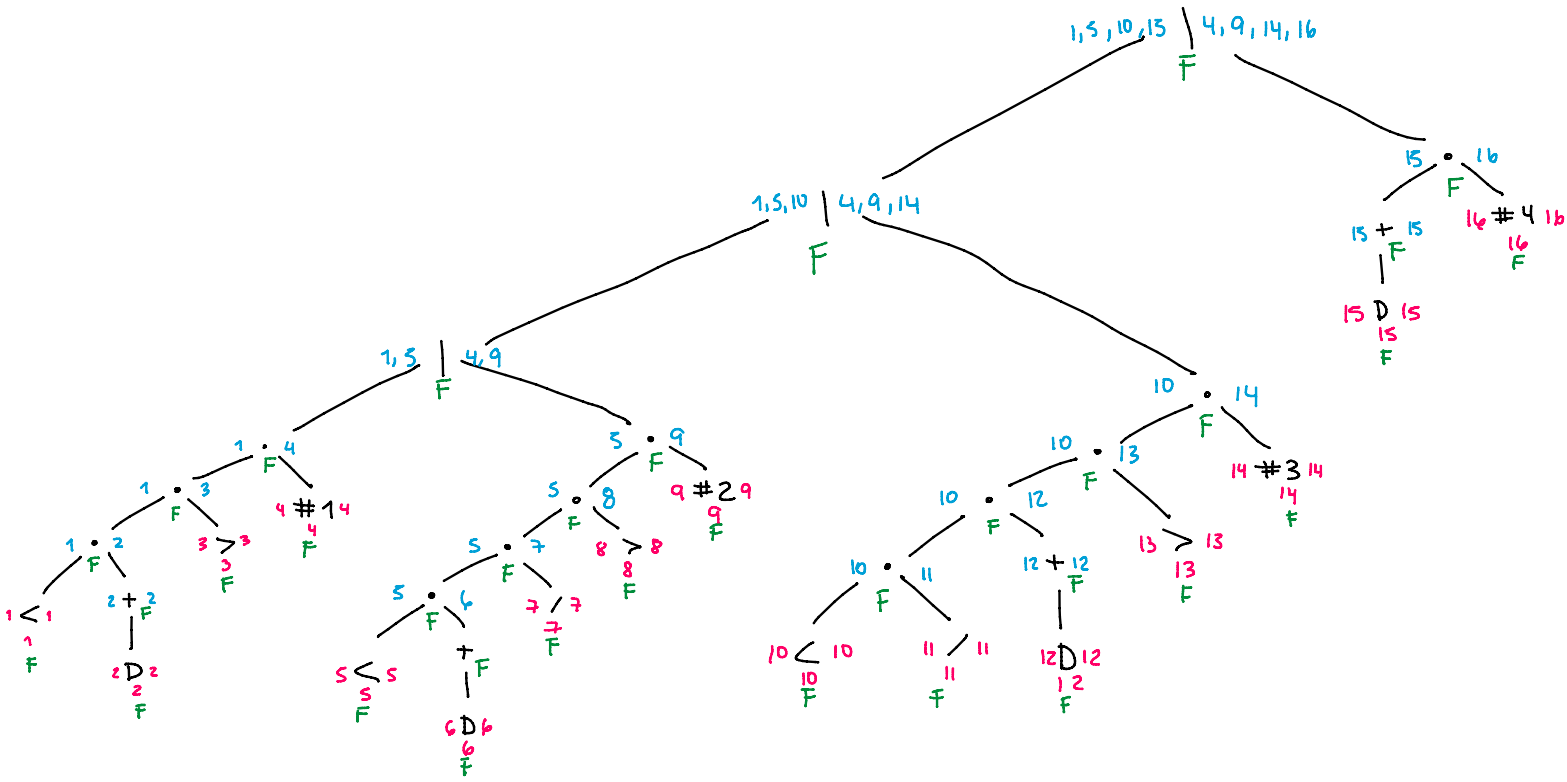
4. Temas de ayuda

PLANIFICACIÓN

Día	Tarea	Tiempo empleado (Horas)
Sabado 27 de agosto	Diseño de autómatas y de funciones claves para la implementación del análisis léxico	3
Domingo 28 de agosto	Programación del DFA	2
Lunes 29 de agosto	Generación de errores y depuración de errores en la lectura	1
Miercoles 14 de septiembre	Generación y validaciones de tokens	3
Jueves 15 de septiembre	Generador de expresiones matemáticas y archivo final	5
Viernes 16 de septiembre	Refactorización y optimización del código	2
Sabado 17 de septiembre	Depuración de bugs	2
Lunes 19 de septiembre	Realización de la documentación	2
Jueves 22 de septiembre	Entrega del proyecto	-

Automata 1

$(<D^{+}>\#1) \mid (<D^{+}/>\#2) \mid (</D^{+}>\#3) \mid (D^{+}\#4)$



# nodo	Simbolo	sigpos
1	<	2
2	D	2,3
3	>	4
4	#1	—
5	<	6
6	D	6,7
7	/	8
8	>	9
9	#2	—
10	<	11
11	/	12
12	D	12,13
13	>	14
14	#3	—
15	D	15,16
16	#4	—

S_0 = 1, 5, 10, 15
< < < D

$T(S_0, <) = sig(1) \cup sig(5) \cup sig(10)$

$= 2, 6, 11 = S_1$
D D /

$T(S_0, D) = sig(15) = 15, 16 = S_2$
D #4

$T(S_1, D) = sig(2) \cup sig(6) = 2, 3, 67 = S_3$
D > D /

$T(S_1, /) = sig(11) = 12 = S_4$
D

$T(S_2, D) = sig(15) = S_2$

$T(S_3, D) = sig(2) \cup sig(6) = S_3$

$T(S_3, >) = sig(3) = 4 = S_5$
#1

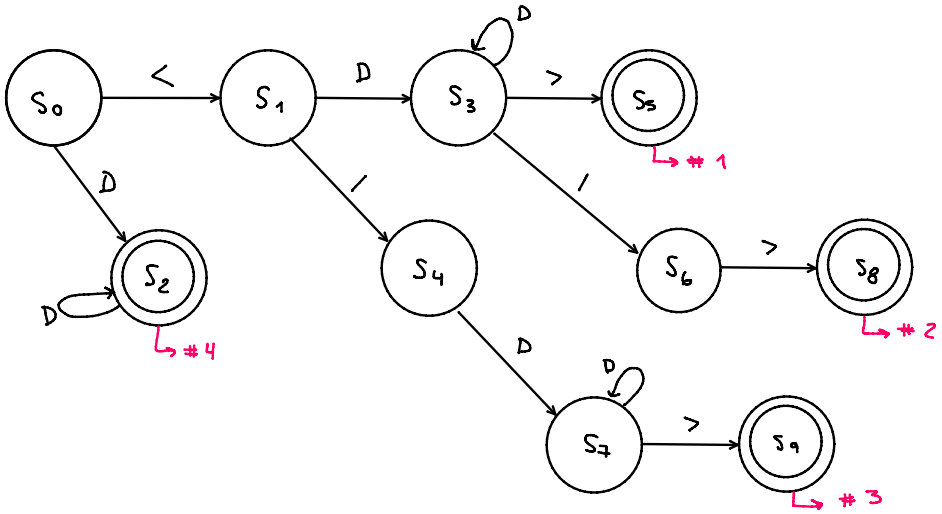
$T(S_3, /) = sig(7) = 8 = S_6$
7

$T(S_4, D) = sig(12) = 12, 13 = S_7$
D >

$T(S_6, >) = sig(8) = 9 = S_8$
#2

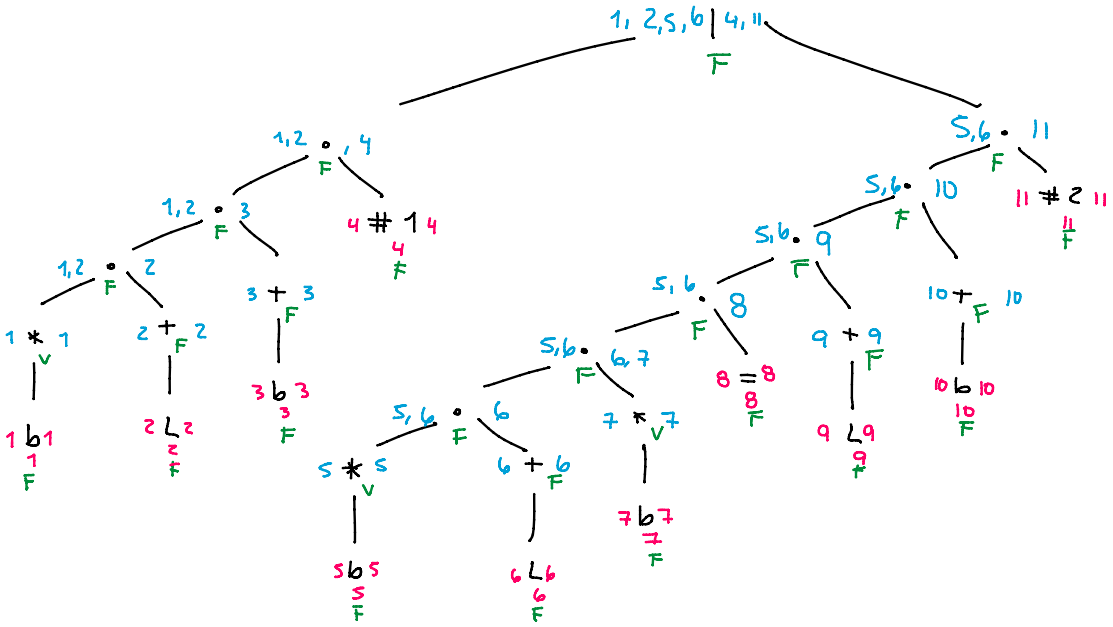
$T(S_7, D) = sig(12) = S_7$

$T(S_7, >) = sig(13) = 14 = S_9$
#3



Automata 2

$$(b^* L^+ b^+ \#1) \mid (b^* L^+ b^* = L^+)$$
1 2 3 4 5 6 7 8 9



#No	Simb	sig
1	b	1, 2
2	L	2, 3
3	b	3, 4
4	#1	
5	b	5, 6
6	L	6, 7, 8
7	b	7, 8
8	=	
9	L	9, 10
10	b	10, 11
11	#2	

$S_0 = 1, 2, 5, 6$
b L b L

$T(S_0, b) = sig(1) \cup sig(5) = 1, 2, 5, 6 = S_0$
b L b L

$T(S_0, L) = sig(2) \cup sig(6) = 2, 3, 6, 7, 8 = S_1$
L b L b =

$T(S_1, L) = sig(2) \cup sig(6) = S_1$

$T(S_1, b) = sig(3) \cup sig(7) = 3, 4, 7, 8 = S_2$
b #1 b =

$T(S_1, =) = sig(8) = 9 = S_3$

$T(S_2, b) = sig(3) \cup sig(7) = S_2$

$T(S_2, =) = sig(8) = 9 = S_3$

$T(S_3, L) = sig(9) = 9, 10 = S_4$

$T(S_4, L) = sig(9) = S_4$

$T(S_4, b) = sig(10) - 10, 11 = S_5$

$T(S_5, b) = S_5$