ANALIZADOR LÉXICO

LENGUAJES FORMALES Y DE PROGRAMACION

PROYECTO 1

2022

ESTUARDO SEBASTIÁN VALLE BANCES
202001954
3674674601O1

INDICE

1. Definición de Expresiones Regulares

Lenguaje de Entrada

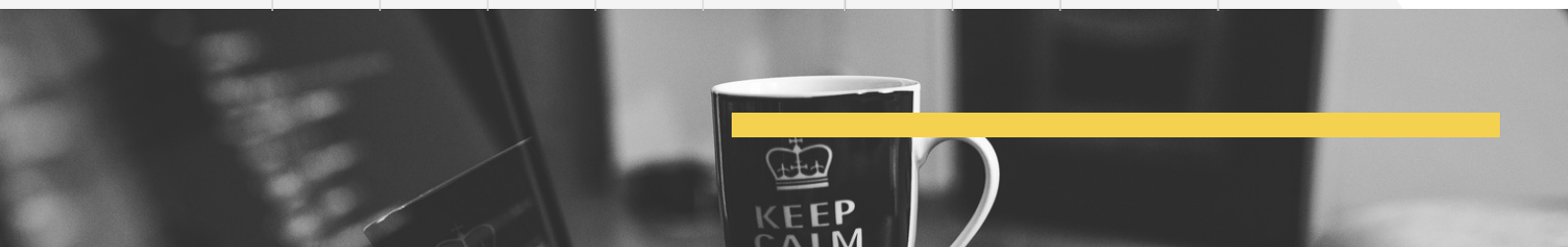
```
formulario ->> [
    <
        tipo: "etiqueta",
        valor: "Nombre:"
    >,
    <
        tipo: "texto",
        valor: "Nombre",
        fondo: "Ingrese nombre"
    >,
    <
        tipo: "grupo-radio",
        nombre: "sexo",
        valores: ['Masculino' , 'Femenino']
    >,
    <
        tipo: "grupo-option",
        nombre: "pais",
        valores: ['Guatemala', 'El Salvador', 'Honduras']
    >,
    <
        tipo: "boton",
        valor: "Valor",
        evento: <EVENTO>
    >
]
```

Expresiones Regulares

1. Definicion de Palabras Reservadas

Desglosando el Lenguaje

	PALABRAS RESERVADAS		
		PATRON	LEXEMA
1	formulario	Letras [a-zA-Z]	formulario
2	tipo	Letras[a-zA-Z]	tipo
3	nombre	Letras[a-zA-Z]	nombre
4	valores	Letras[a-zA-Z]	valores
5	valor	Letras[a-zA-Z]	valor
6	evento	Letras[a-zA-Z]	evento
7	fondo	Letras[a-zA-Z]	fondo
8	texto	Letras[a-zA-Z]	texto
9	etiqueta	Letras[a-zA-Z]	etiqueta
10	grupo-radio	Letras[a-zA-Z]	grupo-radio
11	grupo-option	Letras[a-zA-Z]	grupo-option
12	boton	Letras[a-zA-Z]	boton
13	entrada	Letras[a-zA-Z]	entrada
14	info	Letras[a-zA-Z]	info



1. Definición de Expresiones Regulares

Lenguaje de Entrada

1. Definicion de Signos

SIGNOS		
	PATRON	LEXEMA
<	Caracter <	<
>	Caracter >	>
~>>	Caracter ~>>	~>>
:	caracter :	:
,	Caracter ,	,
[	Caracter [	[
]	Caracter]	]

1. Definicion de Cadenas

Cadena = Todo lo que sea distinto de "

cadenas de tipo 1	c'
	"c"
]	]

Al final, la expresion regular tendrá la siguiente forma

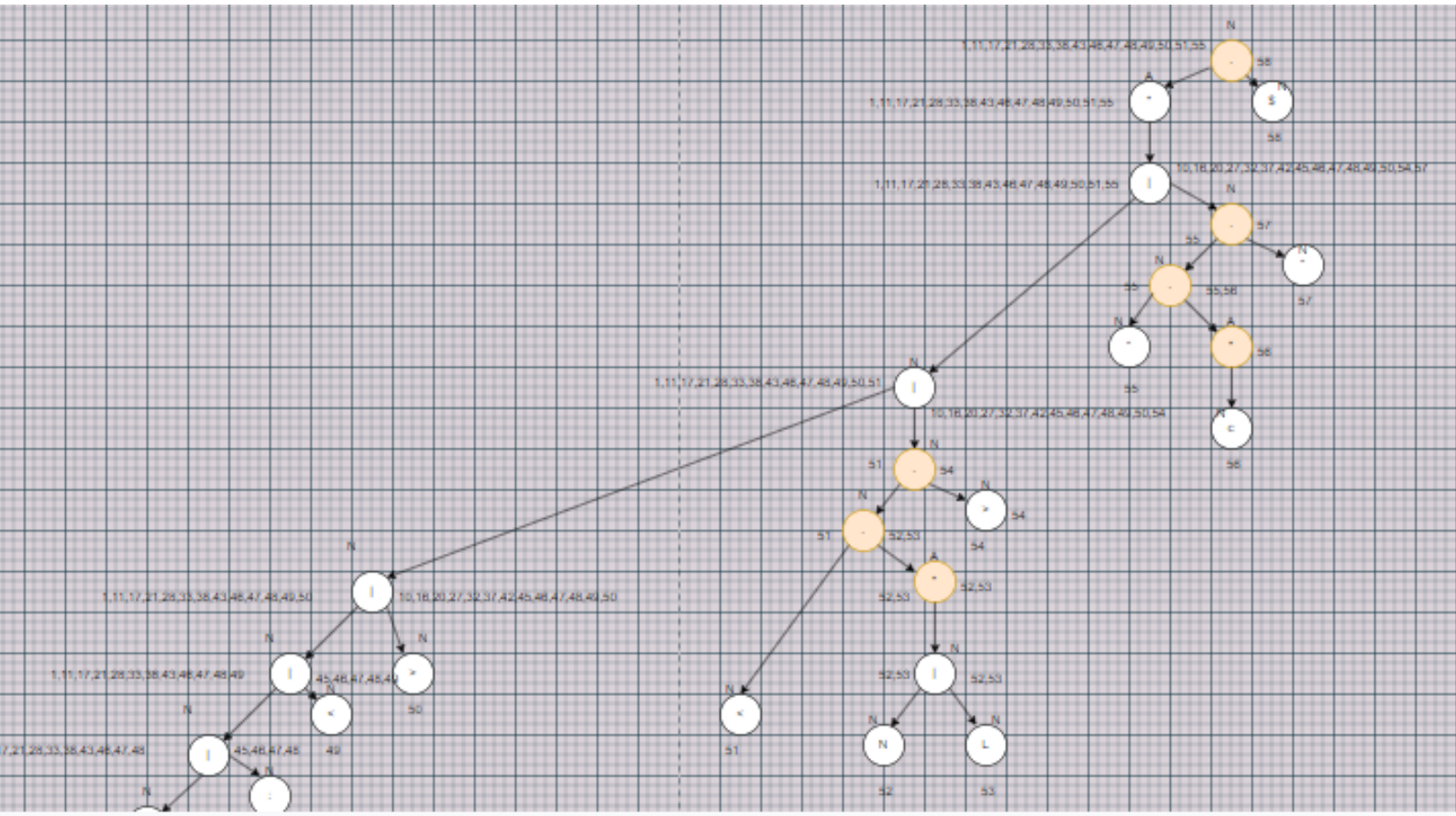
(formulario | tipo | nombre | valores | valor | evento | fondo | ->> | [|] | : | , | < | > | <(L|N)*)>| "c")*

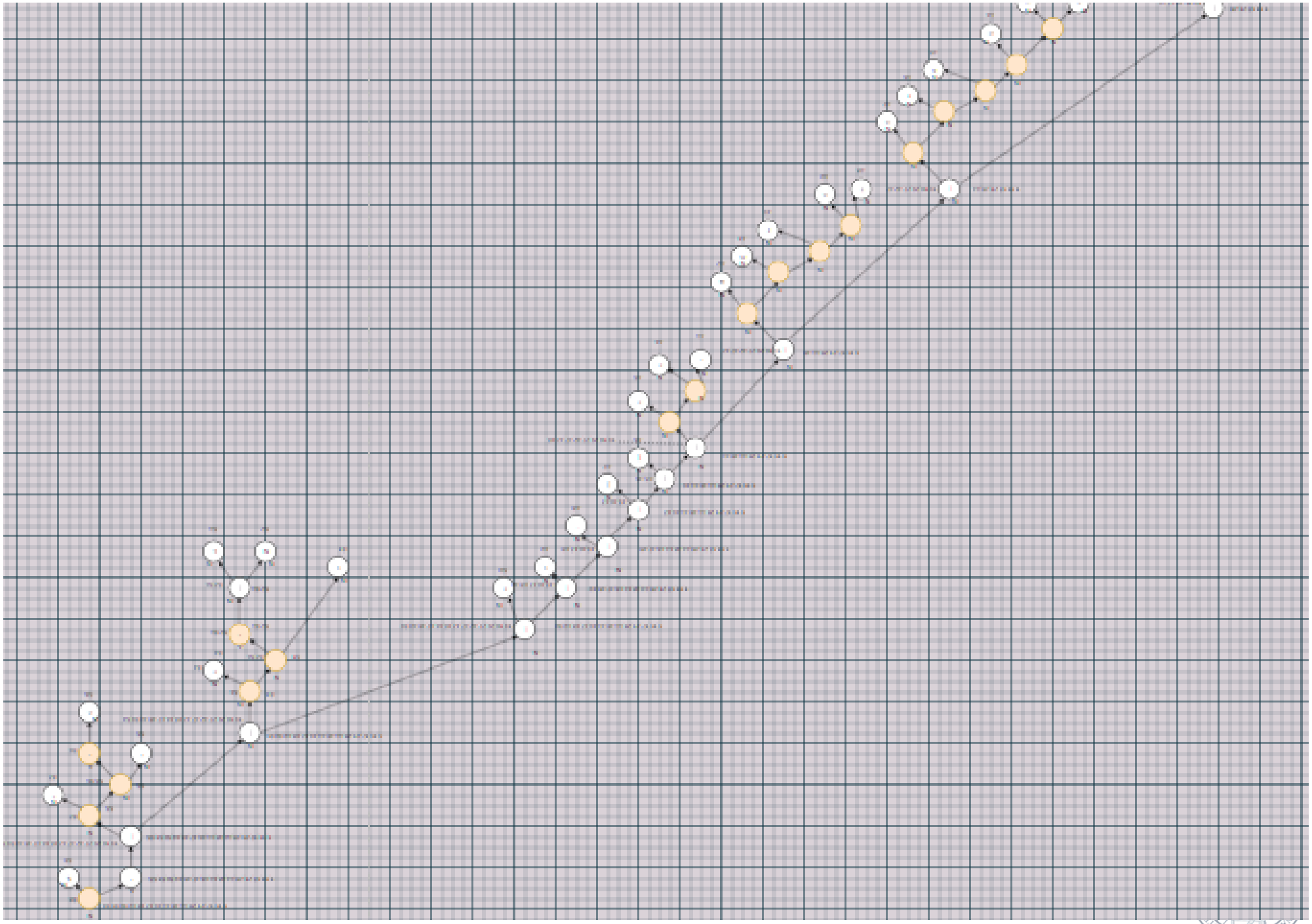
EXPRESIÓN REGULAR FINAL

(formulario | tipo | nombre | valores | valor | evento | fondo | ->> | [|] | : | , | < | > | <(L|N)*)>| "c")*

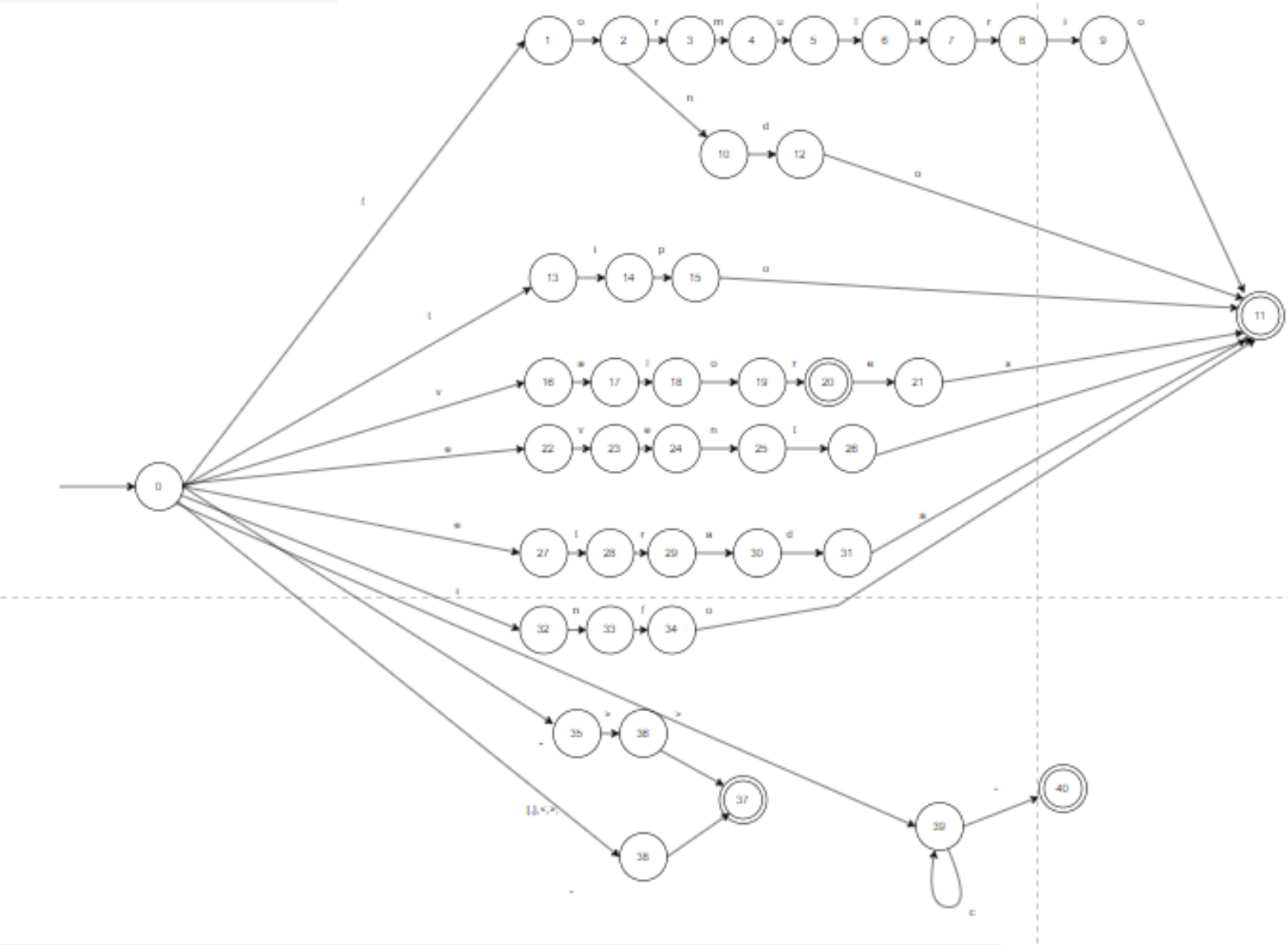
El lenguaje puede aceptar una serie de palabras reservadas (Definidas anteriormente) y/o los simbolos

Aplicando el método del Árbol:
https://app.diagrams.net/#G1Nl7TLcPzmo9pR_-fJyWGqjvP4lhGhFII





2. Automata Finito Determinista



https://app.diagrams.net/#G1Nl7TLcPzmo9pR_-fJyWGqjvP4lhGhFII

3. Implementación

```python
class Token:
    def __init__(self, token: str, lexema: str, fila: int, col: int) -> None:
        self.token: str = token
        self.lexema: str = lexema
        self.fila: int = fila
        self.col: int = col
```

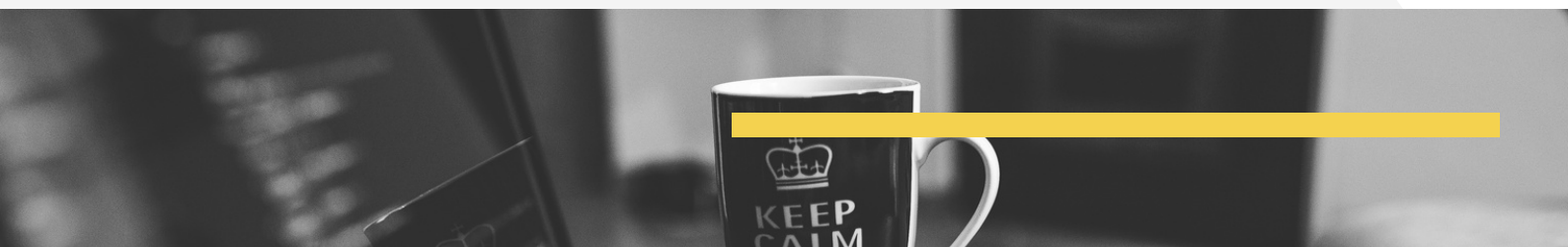
<tokenClass, lexema>

Al entrar en un estado de aceptación, se crea un objeto de Token y se añade a una Lista de Tokens

Además es necesario saber donde se encuentra el Token para un futuro analisis, por lo que col y fila declaran la columna y linea.

```python
class ErrorEntry:
    def __init__(self, linea: int, col: int, char: str) -> None:
        self.linea: int = linea
        self.col: int = col
        self.char: str = char
```

Si no se cumplen las condiciones del automata, este entra en un error. Los errores deben ser rastredos a partir de linea y columna.



3. Implementación

```
def automata(input: str) -> Tuple[Tuple[Token], Tuple[ErrorEntry]]:

    # Agregar char al final
    tokens: List[Token] = []  # Lista tokens
    errores: List[ErrorEntry] = []  # Lista errores
    estado: int = 1  # Estado inicial
    lexema: str = ''  # lexema actual
    index: int = 0  # indice

    row: int = 1  # fila
    col: int = 0  # columna
    print("El Archivo>>")
    print(str(len(input)))
```

El automata fue programado a partir de n WHILE que se mueve con la variable CHAR indicada por INDEX

```
while index < len(input):
    char = input[index]

    # Estado inicial
    if estado == 1:
        #Lista de transiciones
        if char == 'f' or char == 'F':
            estado = 2
            index += 1
            col += 1
            lexema += char
```

Envio a otro Estado

Siguiente Caracter

Guardado de Caracter en Lexema

cada estado tiene una serie de validaciones

```
elif estado == 11:
    #Sera estado de aceptacion para palabras reservadas
    estado = 1
    tokens.append(Token('PALABRA RESERVADA', lexema, row, col))
    lexema = ''
```

3. Implementación

```
def analyzeFile(tokens: List[Token]):
    elements = []
    errores = 0
    estado = 0
    index = 0
    tipos = ['ETIQUETA','TEXTO','GRUPO-RADIO','GRUPO-OPTION','BOTON']
    eventos = ['ENTRADA','INFO']

    while index < len(tokens):
        print(tokens[index].lexema)
        if estado == 0:
            if tokens[index].lexema == '<':
```

AnalyzeFile crea un listado de elementos para la generacion de HTML

```
class Element:
    def __init__(self) -> None:
        self.tipo = None
        self.valor = None
        self.valores = None
        self.fondo = None
        self.nombre = None
        self.evento = None
```

Que posteriormente se analizaran en Processfile

```
def generateForms(elements: List[Element]):
    scripts = '''<script>
                function show() {'''
    generalIFrame = ''
    htmlForms = '''
    <html>
  <head>
    <title>5K & 10K Registration Form</title>
    <link href="https://fonts.googleapis.com/css?family=Roboto:300,
    <link rel="stylesheet" href="https://use.fontawesome.com/releas
    <style>
      html, body {
      min-height: 100%;
      }
      body, div, form, input, select, textarea, label {
```