Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Lenguajes Formales y de Programación
Catedrático: Inga. Zulma Karina Aguirre Ordoñez
Tutor académico: Douglas Omar Arreola Martínez

SECCION B-

Primer semestre 2022

MANUAL TÉCNICO
Proyecto 1

Programa prototipo de análisis léxico y creación de formularios dinámicos

Nombre: Moises David Maldonado de León **Carné:** 202010833

Guatemala, 20 de marzo de 2022

Introducción

En el presente documento, se describe la estructura y los demás aspectos técnicos del analizador léxico y la creación de formularios dinámicos para el departamento de informática de la Universidad de San Carlos. Se conocerá no solo el detalle de estructura del programa sino también el proceso previo, es decir, las expresiones regulares utilizadas y el Autómata Finito Determinista (AFD) creado para la solución solicitada.

OBJETIVOS

Generales

Familiarizar al lector con la lógica planteada para el funcionamiento del sistema realizado mediante el uso del lenguaje de programación Python y aplicando un AFD para la parte del análisis léxico.

Específicos

- Mostrar la estructura del programa realizado.
- Dar a conocer con detalle los métodos y variables empleadas, así como su funcionalidad dentro del programa.
- Mostrar la tabla de tokens, expresiones regulares, el método del árbol y el autómata creado para el sistema.

Alcances

El programa se realizó con el fin de optimizar la creación de formularios, fue creado con el propósito de crear un entorno agradable donde el usuario puede verificar si existen errores en los archivos de entrada y a su vez, con una sola acción crear los formularios de manera automática.

Especificaciones

Requisitos del Hardware

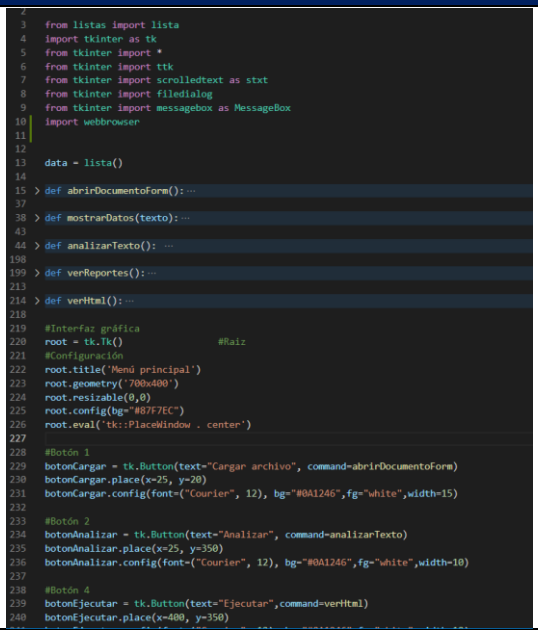
- Mínimo 2GB de memoria RAM
- Procesador Intel Core 2 Duo 2 o superior
- Espacio en disco duro: 10 MB

Requisitos del Software

- El programa fue realizado en Python, específicamente con la versión 3.10.0
- El IDE utilizado fue Visual Studio Code, versión 1.65.2
- Funcional en sistemas operativos como Windows XP, 7,8,10 o MAC OS
- Tener instaladas o con acceso a las siguientes librerías y paquetes: tkinter, webbrowser.

Lógica del programa

Descripción de Clases	
Clase	**Descripción**
Lista()	Clase que maneja la inserción de tokens y errores léxicos encontrados durante el análisis. También contiene los métodos para mostrar o eliminar datos y las funciones que generan los reportes y formularios.
Token1	Clase que contiene un constructor para guardar los objetos de tipo error. Posee métodos para devolver la información en un arreglo y métodos *"get"* para obtener el valor individual de la clase: caracter, tipo de error, fila y columna.
error	Clase que contiene un constructor para guardar los objetos de tipo token. Posee métodos para devolver la información en un arreglo y métodos *"get"* para obtener el valor individual de la clase: tipo de token, lexema, fila y columna.

Archivo main	
Es el archivo llamado *"main.py"*. Es el archivo principal y el cual debe ejecutarse para que el sistema se inicie. En principio contiene, además de las librerías y paquetes a necesitar, las funciones principales como abrir documento, ver reportes y el analizador léxico. Seguido de eso se encuentra toda la creación de la interfaz gráfica mediante el uso de tkinter.	

```
from listas import lista
import tkinter as tk
from tkinter import *
from tkinter import ttk
from tkinter import scrolledtext as stxt
from tkinter import filedialog
from tkinter import messagebox as MessageBox
import webbrowser


data = lista()

> def abrirDocumentoForm():···

> def mostrarDatos(texto):···

> def analizarTexto():···

> def verReportes():···

> def verHtml():···

#Interfaz gráfica
root = tk.Tk()                #Raiz
#Configuración
root.title('Menú principal')
root.geometry('700x400')
root.resizable(0,0)
root.config(bg="#E7F7EC")
root.eval('tk::PlaceWindow . center')

#Botón 1
botonCargar = tk.Button(text="Cargar archivo", command=abrirDocumentoForm)
botonCargar.place(x=25, y=20)
botonCargar.config(font=("Courier", 12), bg="#0A1246",fg="white",width=15)

#Botón 2
botonAnalizar = tk.Button(text="Analizar", command=analizarTexto)
botonAnalizar.place(x=25, y=350)
botonAnalizar.config(font=("Courier", 12), bg="#0A1246",fg="white",width=10)

#Botón 4
botonEjecutar = tk.Button(text="Ejecutar",command=verHtml)
botonEjecutar.place(x=400, y=350)
```

Descripción de las funciones		
Clase	**Método**	**Descripción**
Lista()	**__init__**	Se inicia al ser llamada la función y contiene los arreglos donde se irán guardando los tokens y errores.
	insertarToken(self,token,valor,fila,columna)	Método para la inserción de un token en el arreglo. Se deben de enviarle los parámetros solicitados.
	insertError(self, caracter,tipo,fila,columna)	Método para la inserción de un error en el arreglo. Se deben de enviarle los parámetros solicitados.
	def mostrarTokens(self) **def mostrarErrores(self)**	Métodos para mostrar los tokens y errores en la consola. Estos métodos son secundarios y es para pruebas de verificación de datos.
	def eliminarTodo(self)	Este método elimina la información en los arreglos y se manda a llamar en la función de analizar. Esto permite la inserción de datos sin que se sobrescriban datos o se repitan.
	def reporteTokens(self)	Función que recorre la lista de tokens y genera una tabla en un archivo HTML
	def reporteErrores(self)	Función que recorre la lista de errores y genera una tabla en un archivo HTML
	def crearHtml(self)	Función que recorre el arreglo de tokens y obtiene los datos necesarios para la creación de los formularios. Recuerde que el archivo de entrada lleva una estructura definida y puede contener cualquier elemento válido ente <>.
	def botonEvento(self, text)	Método para crear una página html con la información de entrada. Esta página se invoca para la creación del iframe en el formulario. Recibe el parámetro de texto. Se manda a llamar en la función de analizarTexto() y abrirDocumentoForm()

Descripción de las funciones		
Clase	**Método**	**Descripción**
Token1	**__init__**	Se inicia al ser llamada la función y contiene los atributos de los tokens: tipo de token, valor, fila y columna
	def enviarTokens(self)	Función que retorna los atributos del constructor en forma de arreglo.
	def getToken(self) **def getValor(self)** **def getFila(self)** **def getColumna(self)**	Funciones que retornan el valor de cada atributo respectivamente como una variable String.

Descripción de las funciones		
Clase	**Método**	**Descripción**
error	**__init__**	Se inicia al ser llamada la función y contiene los atributos de los errores: caracter, tipo de error, fila y columna
	def enviarErrores(self)	Función que retorna los atributos del constructor en forma de arreglo.
	def getCaracter(self) **def getTipo(self)** **def getFila(self)** **def getColumna(self)**	Funciones que retornan el valor de cada atributo respectivamente como una variable String.

Descripción de las funciones		
Archivo	**Método**	**Descripción**
Main.py	**def abrirDocumentoForm()**	Función que abre una ventana para seleccionar un archivo. Usa tkinter y contiene validaciones por si no se selecciona un archivo.
	def mostrarDatos(texto)	Función que muestra el texto leído en la caja de texto (ScrolledText) de la interfaz gráfica. Tiene un método de “delete” para que no muestre información repetida. Esta función es llamada en el método para abrir documento.
	def analizarTexto()	Función que maneja el autómata. Al inicio se manda a llamar el método de eliminarTodo() y la función botonEvento(). Contiene toda la estructura de análisis que se mostrará más adelante con el método del árbol.
	def verReportes()	Función que obtiene el dato del elemento ComboBox (lista desplegable con reportes y documentación) y muestra la información solicitada ya sea llamando los métodos o usando webbrowser.
	def verHtml()	Esta función es llamada por el botón de ejecutar y valida la creación de los formularios en html.

Flujo del programa

Terminales

Para el lenguaje utilizado se identificaron los siguientes terminales:

Terminales
1. Identificadores 2.Palabras reservadas 3. Símbolos 4. Cadenas entre comillas dobles y simples

Tabla de tokens

Se describen los conjuntos utilizados: letras minúsculas del alfabeto (L), números enteros entre 0 y 9 (D) y finalmente los símbolos reconocidos en el lenguaje (S).

L = [a-z] **D = [0-9]** **S=[' ~ ', ' > ', '< ', ' [', '] ', ' : ', ' , ']**

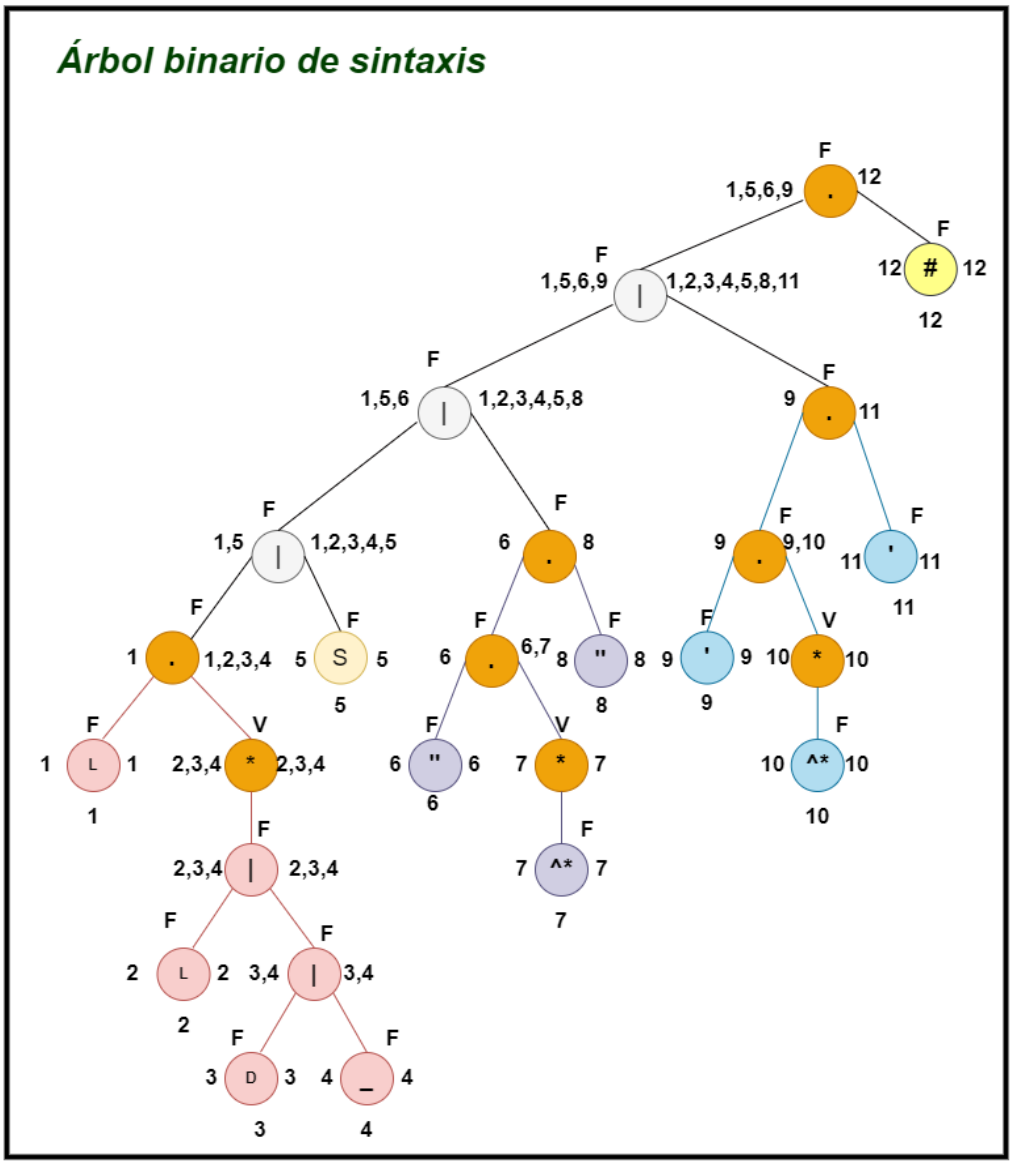
Token	Patrón	ER	Lexema
Identificador	Una letra, que puede ser seguida de una combinación de más letras o números o un guión bajo sin orden o cantidad específica.	L(L|D|'_')*	**tipo** **valor** **hola_**
Símbolo	Un símbolo de "~" o ">" o "<" o "[" o "]" o ":" o ","	S	**[** **:** **>**
Cadena doble	Inicia con una comilla doble seguida de una secuencia de cualquier cosa y termina con una comilla doble	(“) (^”)* (”)	**"etiqueta"** **"texto"**
Cadena simple	Inicia con una comilla simple seguida de una secuencia de cualquier cosa y termina con una comilla simple	(') (^”)* (')	**'masculino'** **'femenino'**

Nota: Las palabras reservadas se analizarán en código

Expresión Regular: **(L(L|D|'_')*) | S | ((“) (^”)* (”)) | ((') (^”)* (‘))**

Método del árbol

ER	((L(L|D|'_')*)| S | ((“) (^”)* (”)) | ((') (^”)* ('))) #



Valor	Hoja	Siguientes
L	1	2,3,4,12
L	2	2,3,4,12
D	3	2,3,4,12
_	4	2,3,4,12
S	5	12
"	6	7,8
^*	7	7,8
"	8	12
'	9	10,11
^*	10	10,11
'	11	12
#	12	-------

Tabla de transiciones

	Estado	Valores	Siguientes
Inicio	**S0**	L, S, ", ' 1,5,6,9	L:{2,3,4,12} = **S1** S:{12} = **S2** ":{7,8} = **S3** ':{10,11} = **S4**
Aceptación	**S1**	L, D, _ , # 2,3,4,12	L:{2,3,4,12} = **S1** D:{2,3,4,12} = **S1** _:{2,3,4,12} = **S1**
Aceptación	**S2**	# 12	----------
	S3	^*, " 7,8	^*:{7,8} = **S3** ":{12} = **S2**
	S4	^*, ' 10,11	^*:{10,11} = **S4** ':{12} = **S2**

Método del árbol – Construcción del AFD

	Estados	Σ						
		L	S	"	'	^*	_	D
Inicio	S0	S1	S2	S3	S4	-	-	-
#	S1	S1	-	-	-	-	S1	S1
#	S2	-	-	-	-	-	-	-
	S3	-	-	S2	-	S3	-	-
	S4	-	-	-	S2	S4	-	-