MANUAL TECNICO

Analizador Léxico y Sintáctico App:

Tabla de Contenido

Introducción...……..1
Objetivos ……...1
Información destacada ……...2
Conocimientos previos ……2
1. Requerimientos ……...3
2. Instalación y configuración ……..…4
3. Tabla de Tokens……...5
4. Método de Árbol…..…… ……5
5. Autómata Finito Determinista.……6
6. Gramática Libre de Contexto…….……..……6
6. Estructura del programa…….……..……7
7. Paradigma utilizado …….……...……..…..8
8. Fragmentos de códigos ..……...…….8

Introducción

El presente documento describe la seria de pasos que realizan algunos de los métodos o funciones que conforman el programa computacional (Analizador Léxico y Sintáctico App) por medio de fragmentos de código, como también se detalla los conocimientos previos que debe tener el lector de este manual para comprender de una mejor manera el funcionamiento de cada una de las partes del código que conforman el programa.

Objetivos

Instruir el uso adecuado del programa computacional, describiendo el diseño y la lógica del programa por medio de fragmentos de códigos.

Describir al usuario el funcionamiento del programa para el mejor uso de él y demostrar el proceso necesario para su ejecución.

Orientar al usuario a entender la estructura del programa, como lo son sus clases y cada uno de los métodos que componen dicha clase.

Información destacada

El manual técnico hace referencia a la información necesaria con el fin de orientar al personal en la concepción, planteamiento análisis programación e instalación del sistema. Es de notar que la redacción propia del manual técnico está orientada a personal con conocimientos en sistemas y tecnologías de información, conocimientos de programación avanzada sobre Python, responsables del mantenimiento e instalación del programa computacional en el computador.

Conocimientos Previos

Los conocimientos mínimos que deben tener las personas que operarán las páginas y deberán utilizar este manual son:

- Cocimientos y entendimientos en logaritmos
- Conocimientos en Python
- Programación Orientada a Objetos
- Interfaces graficas (Tkinter)
- Conocimiento básico de Windows

1. Requerimientos

El sistema puede ser instalado en cualquier sistema operativo que cumpla con los siguientes requerimientos:

- Sistema Operativo: Windows 7 o superior
- Procesador mínimo Intel Pentium (800MHz Intel Pentium)
- Mínimo 1GB en RAM
- IDE Visual Studio Code, o compatible con Python
- Exploradores: Internet Explorer 9 y superior

2. Instalación y configuración

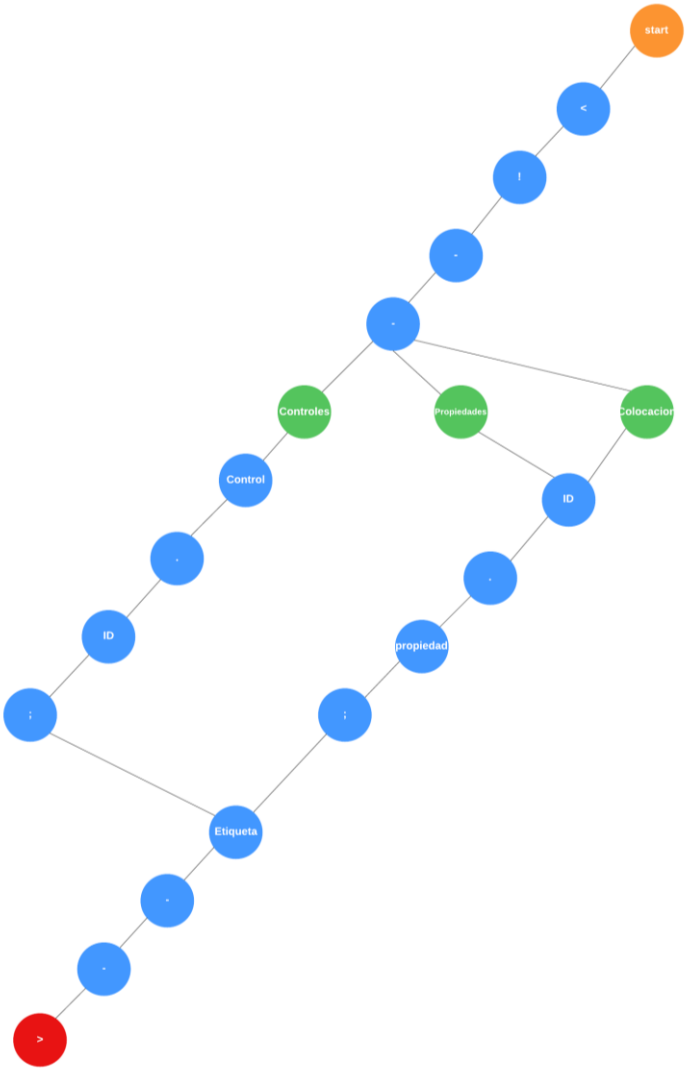
Para el proceso de instalación de esta aplicación únicamente es necesario tener instalado un IDE que sea compatible con el lenguaje Python para ejecutar la aplicación desde la terminal de este.

No es necesario tener alguna configuración ya que la configuración que trae por determinado el IDE es la necesaria para que el funcionamiento del programa sea posible.

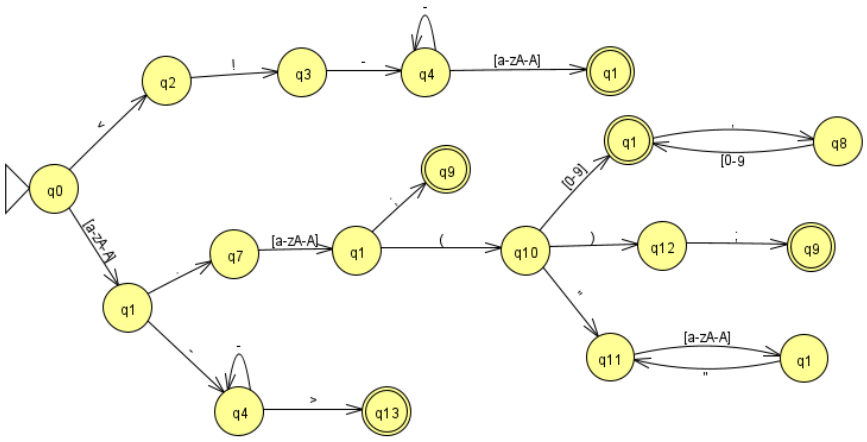
3. Tabla de Tokens

Nombre	Descipcion del patron	Epresion Regular	Ejemplo
Menor Que	tr_MenorQue	<	<
Exclamacion Cerrado	tr_ExclamacionCerrado	!	!
Guion	tr_Guion	-	-
Etiqueta	tr_Etiqueta	Controles	Controles
Tipo	tr_Tipo	Tipo	Boton
Punto	tr_Punto	.	.
Contenido	tr_contenido	[a-zA-Z]+	boton1
Punto y Coma	tr_PuntoYComa	;	;
Propiedad	tr_Propiedad	Propiedad	setAncho
Parentesis Abierto	tr_ParentesisAbierto	(	(
Comilla Doble	tr_ComillaDoble	"	"
Numero	tr_Contenido	\d+	150
Parentesis Cerrado	tr_ParentesisCerrado	)	)
Mayor Que	tr_MayorQue	>	>

4. Método del Árbol



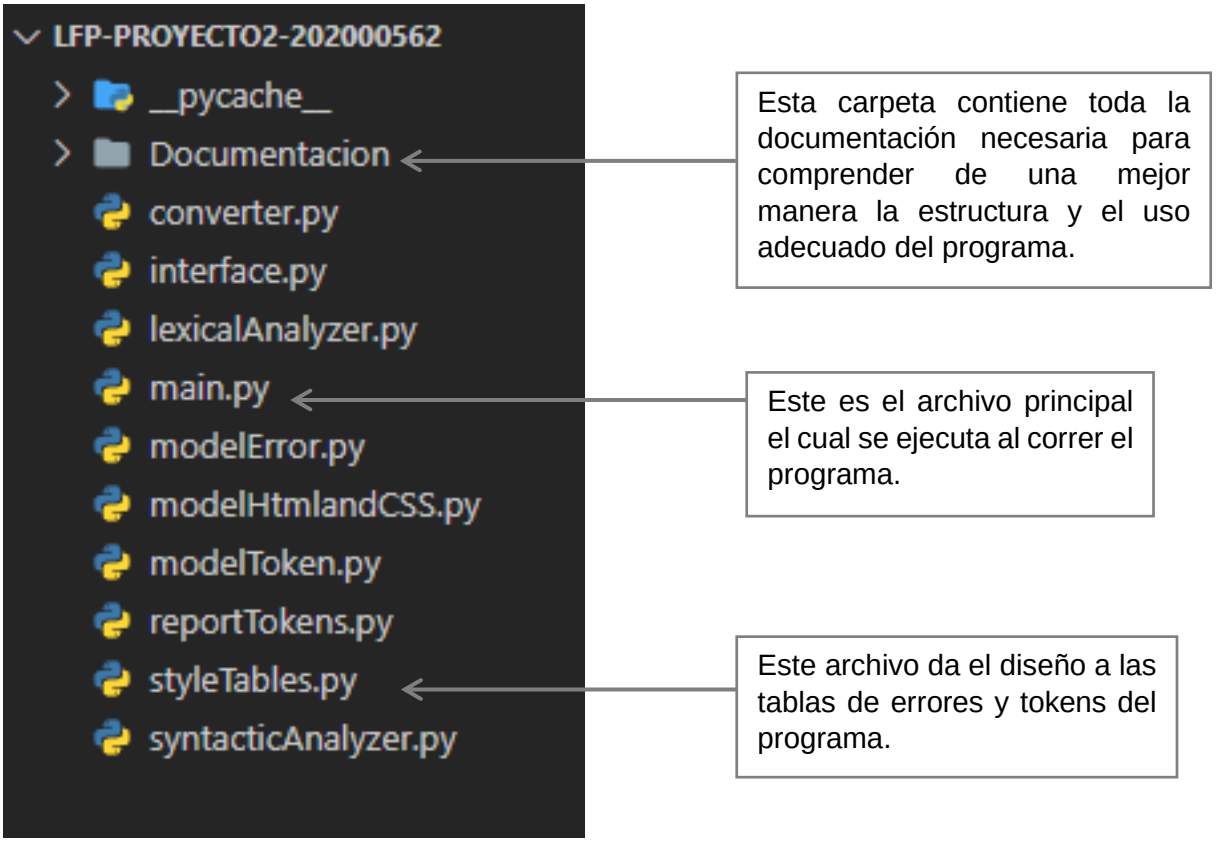
5. Autómata Finito Determinista



6. Gramática Libre de Contexto

simbolo inicial:	start
Controles:	tr_Label tr_Punto tr_Contenido tr_PuntoYComa
	| tr_Check tr_Punto tr_Contenido tr_PuntoYComa
	| tr_RadioBoton tr_Punto tr_Contenido tr_PuntoYComa
	| tr_Texto tr_Punto tr_Contenido tr_PuntoYComa
	| tr_AreaTexto tr_Punto tr_Contenido tr_PuntoYComa
	| tr_Clave tr_Punto tr_Contenido tr_PuntoYComa
	| tr_Contenedor tr_Punto tr_Contenido tr_PuntoYComa
propiedades:	tr_ID tr_Punto tr_Propiedad tr_ParentesisAbierto tr_Contenido tr_ParentesisCerrado tr_PuntoYComa
parametro:	tr_Contenido tr_Coma tr_Contenido
	|tr_ComillaDoble tr_Contenido tr_ComillaDoble
Colocacion:	tr_ID tr_Punto tr_Colocacion
Etiqueta	tr_MenorQue tr_AdmiracionCerrado tr_Guion tr_Guion tr_Etiqueta
	|tr_Etiqueta tr_Guion tr_Guion tr_MayorQue

7. Estructura del programa



8. Paradigma de Programación Usado

Para el desarrollo de este programa se utilizo el paradigma de programación orientada a objetos ya que este disminuye los errores y promociona la reutilización del código.

El paradigma de la programación orientada a objetos consiste en la representación de la realidad. En éste se manejan algunos conceptos básicos como son clases, objetos, atributos, métodos y se caracteriza por emplear la abstracción de datos, herencia, encapsulamiento y polimorfismo.

9. Fragmentos de Códigos

En este apartado se explican detalladamente los método y funciones más importantes que conforman el código del programa. Esto con el objetivo de que la persona a usar el programa necesite dar soporte a la aplicación se le realice una manera más sencilla comprender la lógica del programa.

Código del método “open”:

```
def open(self, event=None):
    r = self.new()
    if r:
        self.path = filedialog.askopenfilename(title="Abrir", filetypes=(("Archivo LFP", "*.gpw"), ("Todos los Archivos", "*.*")))
        if self.path != "":
            path, extension = os.path.splitext(self.path)
            if extension.lower() == ".gpw":
                self.textInput.delete(1.0, "end")
                file = open(self.path, 'r')
                self.contentArchive = file.readlines()
                file.close()
                self.contentArchive = "".join(self.contentArchive)
                self.textInput.insert('insert', self.contentArchive)
                self.contentTextArea = self.textInput.get(1.0, "end-1c")
            else:
                messagebox.showwarning("Advertencia", "¡La extension del archivo es incorrecta!\nUnica extension aceptada -> (.gpw)")
```

Este método primero verifica que el usuario haya seleccionado un archivo, de ser así la ruta del archivo será almacenada en la variable “self.ruta” de caso contrario no realizara nada. Luego de almacenar la ruta del archivo procede a verificar que la extensión del archivo seleccionado sea la correcta (.gpw) de no ser la correcta se mostrara un mensaje de error en pantalla.

Si la extensión del archivo es la correcta procede a leer línea por línea el archivo almacenando toda la información en la variable “self.contentArchive” para luego ser mostrado todo el contenido en la área de texto de la aplicación.

Código del método “save”:

```
def save(self, event=None):
    self.contentTextArea = self.textInput.get(1.0, "end-1c")
    if self.path != "":
        file = open(self.path, "w")
        file.write(self.contentTextArea)
        file.close()
        self.contentArchive = self.contentTextArea
        return True
    else:
        r = self.saveAs()
        return r
```

El método “save” funciona de la siguiente manera:

Este método es llamado cuando el usuario presiona el botón “guardar” que se muestra en la interfaz gráfica de la aplicación y funciona de la siguiente manera:

Guarda todo el contenido que tiene el área de texto de la aplicación en una variable, luego verifica si el contenido pertenece a un archivo abierto por medio de la aplicación y de ser así realiza los cambios que el usuario ingreso en el archivo. En caso de que no haya un archivo abierto el programa le preguntara al usuario si desea guardar el contenido en un archivo nuevo.

Al presionar el botón “guardar” el usuario puede optar por analizar el contenido del archivo. El programa analizara primero léxicamente, luego continuara sintácticamente, por ultimo si no encuentra errores en ninguno de los dos análisis procederá a generar el archivo salida el cual son dos archivos, uno de extensión tipo .css para el diseño y el otro de extensión .html para el contenido.

Clase “lexicalAnalyzer”

En esta clase se encuentra toda la lógica del código que compone el analizador lexico.

Esta clase tiene como objetivo analizar léxicamente todo el contenido que se encuentra en el área de texto de la aplicación. Verifica que en el área de texto no se encuentren errores léxicos, de ser así, el programa almacenar todos los errores en una lista, esto para poder mostrarlos al finalizar el análisis.

Código del método “s1”:

```python
def s1(self,lexeme:str):
    #State S1
    if lexeme.isalpha():
        self.state = 1
        self.scanner += lexeme
        self.numColumn += 1
    elif lexeme.isdigit():
        self.state = 1
        self.scanner += lexeme
        self.numColumn += 1
    elif lexeme == "." and self.scanner.isdigit():
        self.state = 1
        self.scanner += lexeme
        self.numColumn += 1
    else:
        numColumn = str(self.numColumn - (len(self.scanner)-1))
        if self.scanner in self.tokensLabel:
            self.addToken(self.scanner, "tr_{}".format("Etiqueta"), self.numLine, numColumn)
        elif self.scanner in self.tokensType:
            self.addToken(self.scanner, "tr_{}".format("Tipo"), self.numLine, numColumn)
        elif self.scanner in self.tokensProperty:
            self.addToken(self.scanner, "tr_{}".format("Propiedad"), self.numLine, numColumn)
        else:
            self.addToken(self.scanner, "Contenido", self.numLine, numColumn)
        self.state = 0
        self.i -= 1
```

Este método se encuentra en la clase “lexicalAnalyzer” y este se encarga de verificar el estado al que pertenece el lexema leído por el programa perteneciente al contenido del área de texto de la aplicación.

El lexema pasa por varios condicionales, en los primeros tres condicionales se verifica si es letra, numero o punto de ser así, se concatena a la variable “self.scanner”. Si no se cumple con los primeros tres condicionales, el lexema se agrega a la lista “self.tokensReservados”.

Clase “syntacticAnalyzer”

En esta clase se encuentra toda la lógica del código que compone el analizador sintáctico.

Esta clase tiene como objetivo analizar sintácticamente todo el contenido que se encuentra en el área de texto de la aplicación. Verifica que en el área de texto no se encuentren errores sintácticos, de ser así, el programa almacenar todos los errores en una lista, esto para poder mostrarlos al finalizar el análisis.

Código del método “controls”

```
def controls(self, line):
    token = self.observeToken(self.position)
    if token.getType() == "tr_Etiqueta":
        return
    if token.getType() == "tr_Tipo" and token.getLine() == line:
        self.position += 1
        print(token.getLexeme())
        token = self.observeToken(self.position)
        if token.getType() == "Contenido" and token.getLine() == line:
            self.position += 1
            print(token.getLexeme())
            self.variables.append(token.getLexeme())
            token = self.observeToken(self.position)
            if token.getType() == "tr_PuntoYComa" and token.getLine() == line:
                self.position += 1
                print(token.getLexeme())
                token = self.observeToken(self.position)
                self.controls(token.getLine())
            else:
                self.addError(token.getLexeme(), token.getLine(), token.getColumn(), "tr_PuntoYComa", "")
                self.advancePositions(line)
                token = self.observeToken(self.position)
                self.controls(token.getLine())
        else:
            self.addError(token.getLexeme(), token.getLine(), token.getColumn(), "Contenido", "")
            self.advancePositions(line)
            token = self.observeToken(self.position)
            self.controls(token.getLine())
    else:
        self.addError(token.getLexeme(), token.getLine(), token.getColumn(), "tr_Tipo", "")
        self.advancePositions(line)
        token = self.observeToken(self.position)
        self.controls(token.getLine())
```

Este método se encuentra localizado en la clase “lexicalAnalyzer” y funciona así, primero llama a la función “observeToken” la cual retorna un token de la lista de tokens reconocidos por el analizador léxico, luego de obtener dicho token verifica cual es el tipo al que pertenece dicho token. Si el token cumple con una de las condiciones descritas en la función, la función continua con las siguientes condiciones.

Al igual que en la primera condición la función vuelve a llamar al método “lexicalAnalyzer” el cual devuelve otro token distinto al anterior. La función comprueba que el tipo de token obtenido cumpla con la condición descrita, de ser así la función vuelve a realizar el mismo proceso.

La función termina cuando todas las condiciones sean cumplidas, de no cumplirse con alguna condición descrita, la función reportará esto como un error y este será almacenado en una lista que contiene todos los errores sintácticos reconocidos.

Todos los errores, tanto léxicos como sintácticos son mostrados en una tabla, cuando el programa termina su proceso de analizar léxicamente y sintácticamente el contenido del archivo.

Código del método “instructionsControls”

```
def instructionControls(self):
    newTag = None
    if "Etiqueta" in self.scanner:
        id = self.scanner.replace("Etiqueta", "", 1)
        newTag = ModelHTMLandCSS(type="label", id=id, font_size="12px", height="25px", width="100px")
    elif "Boton" in self.scanner:
        id = self.scanner.replace("Boton", "", 1)
        newTag = ModelHTMLandCSS(type="submit", id=id, text_align="left", font_size="12px", height="25px", width="100px")
    elif "Check" in self.scanner:
        id = self.scanner.replace("Check", "", 1)
        newTag = ModelHTMLandCSS(type="checkbox", id=id)
    elif "RadioBoton" in self.scanner:
        id = self.scanner.replace("RadioBoton", "", 1)
        newTag = ModelHTMLandCSS(type="radio", id=id)
    elif "Texto" in self.scanner:
        id = self.scanner.replace("Texto", "", 1)
        newTag = ModelHTMLandCSS(type="text", id=id, text_align="left", height="25px", width="100px")
    elif "AreaTexto" in self.scanner:
        id = self.scanner.replace("AreaTexto", "", 1)
        newTag = ModelHTMLandCSS(type="textarea", id=id, height="150px", width="150px")
    elif "Clave" in self.scanner:
        id = self.scanner.replace("Clave", "", 1)
        newTag = ModelHTMLandCSS(type="password", id=id, text_align="left", height="25px", width="100px")
    elif "Contenedor" in self.scanner:
        id = self.scanner.replace("Contenedor", "", 1)
        newTag = ModelHTMLandCSS(type="div", id=id, font_size="12px")
    if newTag != None:
        self.listTagforHtml.append(newTag)
        self.listTag.append(newTag)
```

Este método se encuentra localizado en el archivo “converter” y es el encargado de crear las etiquetas tanto de del archivo css como del archivo html.

El funcionamiento de este método es el siguiente:

La función verifica el tipo de propiedad que se le asignara a la variable ya creada, si se cumple la condición la función le agregara las propiedades leídas a la variable en específico. Este método es llamado la cantidad de veces que la variable a la que se le dará la propiedad aparezca en el contenido del archivo de entrada.