Universidad De San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Laboratorio Lenguajes Formales y de Programación
Auxiliar Douglas Omar Arreola Martínez
Sección “B -”

“MANUAL DE TECNICO”

Luis Angel Barrera Velásquez

Carné: 202010223

Índice

Objetivos

General:

Brindar al lector información acerca del desarrollo de este proyecto y como se le dio solución para una futura actualización o corrección de errores técnicos como actualizaciones del sistema.

Específicos:

- Detallar con palabras la solución presentada en el código paso a paso para seguir un orden y lógica del programa.

- Proporcionar los conceptos que fueron empleados para la solución de este proyecto con teoría para que sea más entendible la solución expresada en palabras.

Introducción

Este manual técnico está elaborado con el propósito de detallar el funcionamiento del código fuente del proyecto y en caso de que lo lea alguien con conocimientos sobre programación sea de ayuda para realizar futuras actualizaciones sobre el programa o corrección de algún error que suceda durante la ejecución. Es importante mencionar que este proyecto se realizó una fusión entre conceptos básicos de programación como también la utilización de expresiones regulares que viene siendo más complejo, pero con la implementación de estructuras básicas como ciclos o condicionales en la programación se puede controlar la lectura del archivo solicitado por medio de un autómata.

Descripción de la Solución

Para la solución de este proyecto fue necesario utilizar la lectura de archivos de texto plano y guardarlos en un buffer para realizar el respectivo análisis del archivo. Para esto fue necesario la utilización del concepto de expresiones regulares para generar a partir del método del árbol binario de sintaxis generar un autómata para el reconocimiento de tokens para su posterior análisis:

- Primero se realizó una interfaz gráfica por medio de la librería tkinter Python para que el usuario del software tenga una mejor perspectiva de la funcionalidad del software para esto se cargaron diferentes funciones en botones como cargar, analizar, reportes, áreas de texto tanto de entrada como de salida.

- Se realizo para cada token una expresión regular la cual servirá para el analizar léxicamente el archivo de entrada y comprobar que no existan errores léxicos (más adelante estará detallado cada expresión regular junto con su procedimiento para obtener su autómata finito determinista).

- Para el análisis se implementó un solo autómata utilizando un ciclo for en la cual entra el archivo de prueba y lo recorre carácter por carácter ignorando espacios, tabulaciones y saltos de línea, a través de la utilización de estados que son evaluados con condicionales if, elif y else como también estados para el reconocimiento de tokens también fueron utilizadas estructuras condicionales para el análisis sintáctico.

- El archivo debe de seguir con el autómata y al encontrar cada token fue guardado en una lista de objetos de tipo token y en caso de que se detectara un error se guarda el carácter o el lexema erróneo que causo el error en una lista de objetos de tipo error.

- Una vez teniendo una lectura exitosa tanto del analizador léxico como sintáctico este procede a mostrar todas las funcionalidades solicitadas por el usuario en el área de consola (partes detalladas en el manual de usuario).

Conceptos utilizados para la solución

Lectura de archivos de texto plano

La entrada y salida de ficheros (comúnmente llamada File I/O) es una herramienta que Python, como la mayoría de los lenguajes, nos brinda de forma estándar. Su utilización es similar a la del lenguaje C, aunque añade mayores características.

El primer paso para introducirnos es saber cómo abrir un determinado archivo. Para esto se utiliza la función open, que toma como primer argumento el nombre de un fichero.

```
1.  open("archivo.txt")
```

No está de más aclarar que esto no significa ejecutar el programa por defecto para editar el fichero, sino cargarlo en la memoria para poder realizar operaciones de lectura y escritura.

En caso de no especificarse la ruta completa, se utilizará la actual (donde se encuentre el script).

Como segundo argumento, open espera el modo (como una cadena) en el que se abrirá el fichero. Esto indica si se utilizará para lectura, escritura, ambas, entre otras.

```
1.  open("archivo.txt", "r")
```

La r indica el modo lectura. Si se intentara utilizar la función write para escribir algo, se lanzaría la excepción IOError. A continuación, los distintos modos.

- r – Lectura únicamente.
- w – Escritura únicamente, reemplazando el contenido actual del archivo o bien creándolo si es inexistente.
- a – Escritura únicamente, manteniendo el contenido actual y añadiendo los datos al final del archivo.
- w+, r+ o a+ – Lectura y escritura.

Escritura de archivos

Este concepto fue utilizado para la escritura del reporte HTML ya que fue escrito como si escribiéramos un txt pero se uso la sintaxis de HTML y guardarlo como *.html como fue solicitado en el enunciado.

```
1. f = open("archivo.txt", "w")
```

En este ejemplo he abierto el fichero archivo.txt que, en caso de no existir, será automáticamente creado en el directorio actual. A continuación escribiré algunos datos utilizando la función file.write.

```
1. f.write("Esto es un texto.")
```

Esta función fue utilizada hasta de ultimo ya que mientras la creación del html se fue concatenando una cadena bastante larga agregando cada etiqueta que se necesitara como por ejemplo:

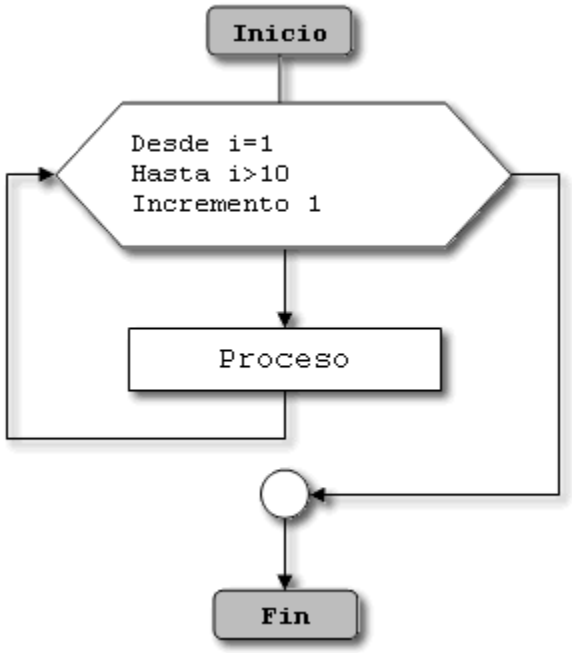
```
1. cadena+=”<table border=\"1\" width=\"{str(imagen.ancho)}\" height=\"{str(imagen.alto)}\">”
```

por ultimo se cerro el archivo leído o de escritura por medio de la función close

```
1. f.close()
```

Ciclos for

El bucle for se utiliza para recorrer los elementos de un objeto iterable (lista, tupla, conjunto, diccionario, …) y ejecutar un bloque de código. En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.



Condicional if

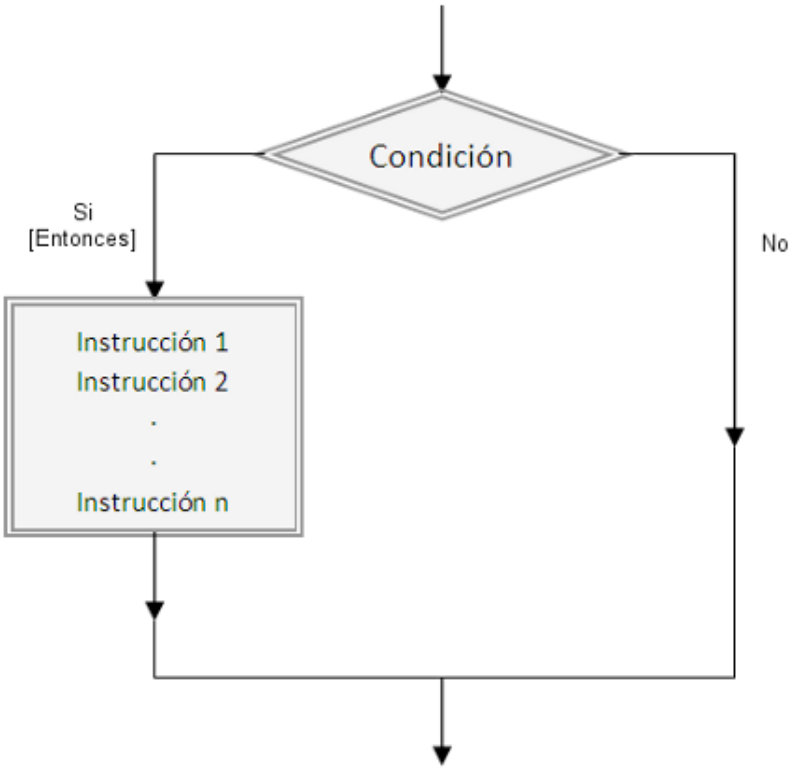
La estructura de control if ... permite que un programa ejecute unas instrucciones cuando se cumplan una condición. En inglés "if" significa "si" (condición).

La sintaxis de la construcción if es la siguiente:

```
if condición:
    aquí van las órdenes que se ejecutan si la condición es cierta
    y que pueden ocupar varias líneas
```

La ejecución de esta construcción es la siguiente:

- La condición se evalúa siempre.
    - Si el resultado es True se ejecuta el bloque de sentencias
    - Si el resultado es False no se ejecuta el bloque de sentencias.



Condicional if.. else

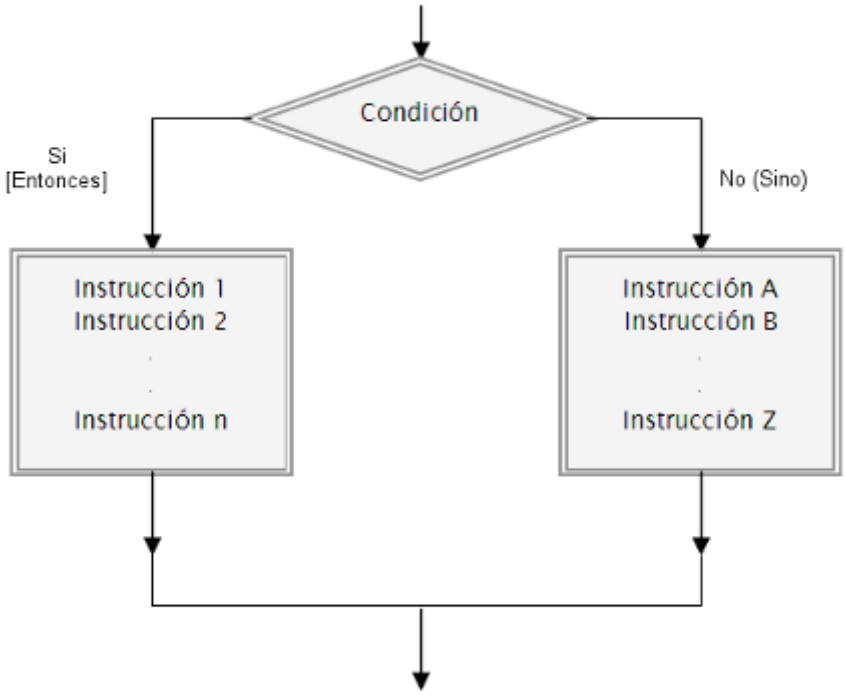
La estructura de control if ... else ... permite que un programa ejecute unas instrucciones cuando se cumple una condición y otras instrucciones cuando no se cumple esa condición. En inglés "if" significa "si" (condición) y "else" significa "si no". La orden en Python se escribe así:

La sintaxis de la construcción if ... else ... es la siguiente:

```
if condición:
    aquí van las órdenes que se ejecutan si la condición es cierta
    y que pueden ocupar varias líneas
else:
    y aquí van las órdenes que se ejecutan si la condición es
    falsa y que también pueden ocupar varias líneas
```

La ejecución de esta construcción es la siguiente:

- La condición se evalúa siempre.
    - Si el resultado es True se ejecuta solamente el bloque de sentencias 1
    - Si el resultado es False se ejecuta solamente el bloque de sentencias 2.



Variables globales

Las variables globales son las variables con alcance global. El alcance global significa que la variable es accesible desde cualquier lugar del programa. Las variables globales se declaran fuera de las funciones ya que su alcance no se limita a ninguna función. La vida útil de la variable global es igual al tiempo de ejecución del programa.

No tenemos que declarar explícitamente las variables antes de usarlas, por lo tanto, para diferenciar entre una variable local y una global, necesitamos especificar que la variable a la que estamos accediendo es la variable global o no. Podemos especificar una variable como global en Python usando la palabra clave global.

Si pasamos el valor a la variable global dentro de una función sin declararla variable global, el valor se pasará a la nueva variable con el mismo nombre. Y el alcance de la nueva variable estará restringido al alcance de la función.

Por ejemplo:

```
date = "17-06-2002"



def update_date():

  global date #declaracion de la variable global hace referencia a:

  date = "12-03-2021"



update_date()

print(date)
```

Analizador Léxico

Un analizador léxico o analizador lexicográfico (en inglés scanner o tokenizer) es la primera fase de un compilador, consistente en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos. Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico (en inglés parser).

La especificación de un lenguaje de programación a menudo incluye un conjunto de reglas que definen el léxico. Estas reglas consisten comúnmente en expresiones regulares que indican el conjunto de posibles secuencias de caracteres que definen un token o lexema.

Análisis

Esta etapa está basada usualmente en una máquina de estados finitos. Esta máquina contiene la información de las posibles secuencias de caracteres que puede conformar cualquier token que sea parte del lenguaje (las instancias individuales de estas secuencias de caracteres son

denominados lexemas). Por ejemplo, un token de naturaleza entero puede contener cualquier secuencia de caracteres numéricos.

- Token (o componente léxico): Secuencia de caracteres con significado sintáctico propio.
- Lexema: Secuencia de caracteres cuya estructura se corresponde con el patrón de un token.
- Patrón: Regla que describe los lexemas correspondientes a un token.

Analizador Sintáctico

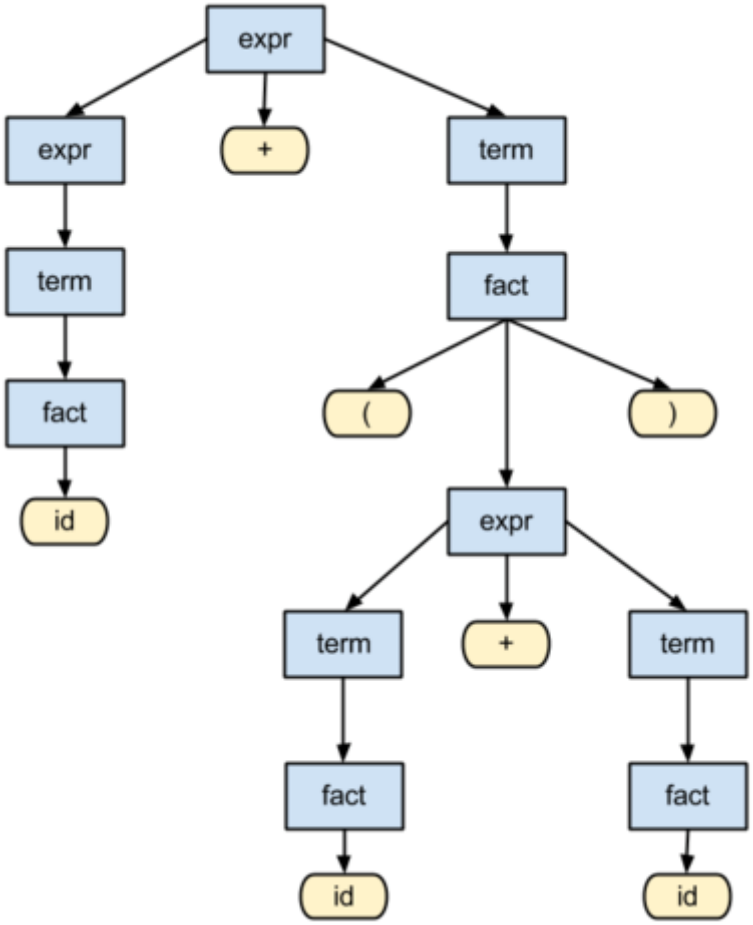
El análisis sintáctico convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea tokens de una secuencia de caracteres de entrada y son estos tokens los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo un árbol de análisis o árboles de sintaxis abstracta.

Los analizadores sintácticos fueron extensivamente estudiados durante los años 1970, detectándose numerosos patrones de funcionamiento en ellos, cosa que permitió la creación de programas generadores de analizadores sintáticos a partir de una especificación de la sintaxis del lenguaje en forma Backus-Naur por ejemplo, tales como yacc, GNU bison y javaCC.

Cómo funciona

Para analizar un texto, los analizadores suelen utilizar un analizador léxico separado (llamado lexer), que descompone los datos de entrada en fichas (símbolos de entrada como palabras). Los Lexers son por lo general máquinas de finitas, que siguen la gramática regular y por lo tanto aseguran un desglose adecuado. Los tokens obtenidos de esta manera sirven como caracteres de entrada para el analizador sintáctico.

El analizador actual maneja la gramática de los datos de entrada, realiza un análisis sintáctico de éstos y como regla general crea un árbol de sintaxis (árbol de análisis). Esto se puede utilizar para el procesamiento posterior de los datos, por ejemplo, la generación de código por un compilador o ejecutado por un intérprete (traductor). Por lo tanto, el analizador es el software que comprueba, procesa y reenvía las instrucciones del código fuente.



Expresiones Regulares

L= {a-z, A-Z}
S= { = | } | { | ; | [|] | (|) | , | ; } (para símbolos en código se identificara cada uno)
D= {0,1,2,3,4,5,6,7,8,9}
C={“}
P={.}

Token Reservada

L+

Token cadena

C(^”)*C

Token Comentarios

(#)(^”)*

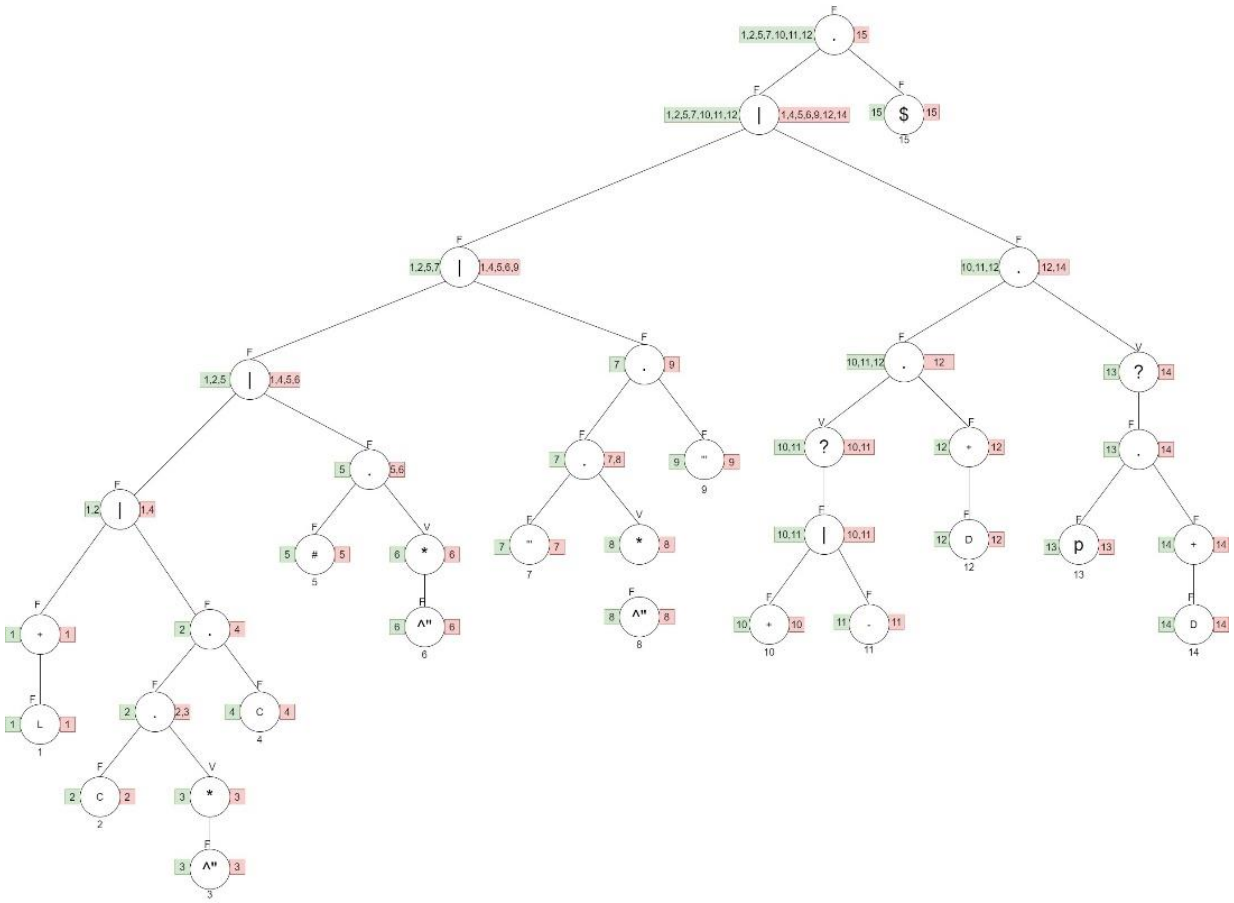
Token Multicomentario

(‘’’)(^”)*(‘’’)

Token Entero/Decimal

(+|-)?D+((P)(D+))?

Árbol Binario de Sintaxis:

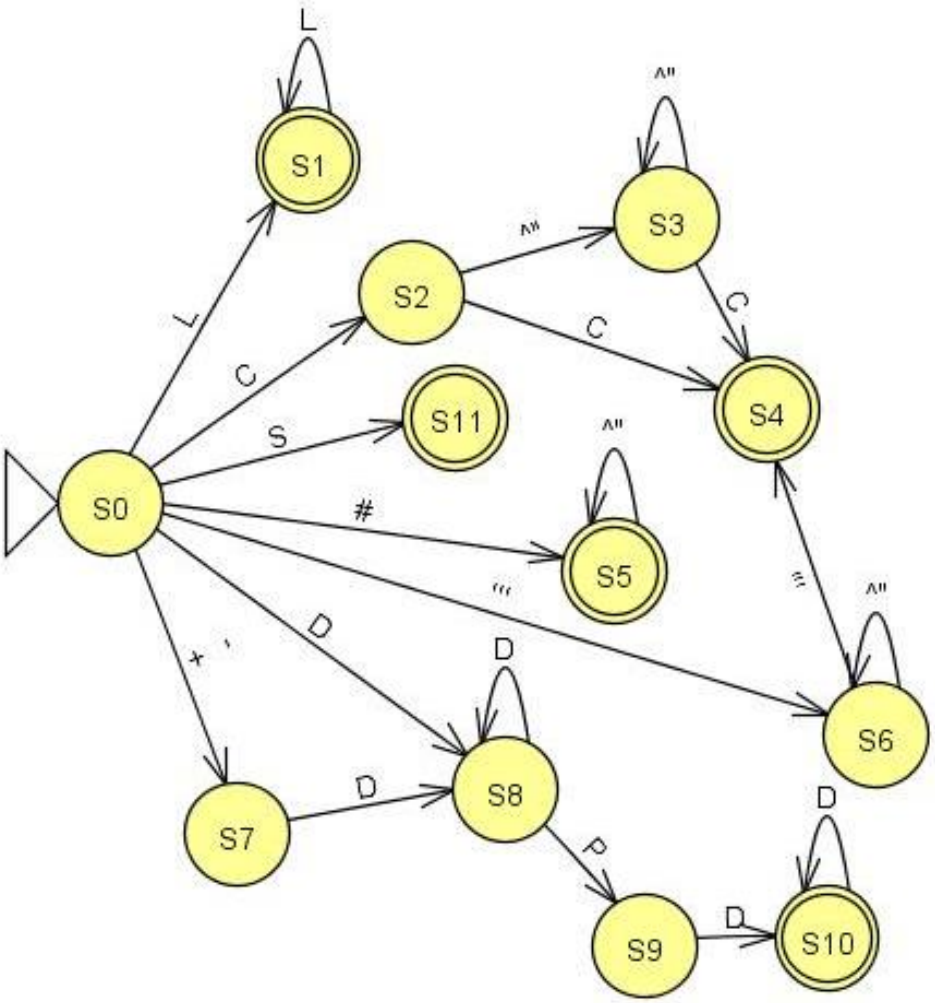


Árbol binario de sintaxis incluido en la carpeta de documentación del proyecto por si se requiere ver más a detalle.

Valor	Hoja	Siguientes
L	1	1,16
C	2	3,4
^”	3	4,3
C	4	16
#	5	6,16
^”	6	6,16
‘’	7	8,9
^”	8	8,9
‘’	9	16
+	10	12
-	11	12
D	12	12,13,16
P	13	14
D	14	14,16
S	15	16
$	16	---

	Estado	Valores	Siguientes
	S0	L C # ‘’ + - D S	L={1,16}=S1 C={3,4}=S2 #={6,16}=S5 ‘’={8,9}=S6 +={12}=S7 -={12}=S7 D={12,13,16}=S8 S={16}=S11
Aceptación	S1	L $	L={1,16}=S1
	S2	^” C	^”={4}=S3 C={16}=S4
	S3	C	C={16}=S4 ^”={4}=S3
Aceptación	S4	$	
Aceptación	S5	^” $	^”={6}=S5
	S6	^” ‘’	^”={8,9}=S6 ‘’={16}=S4
	S7	D	D={12,13,16}=S8
Aceptación	S8	D P $	D={12,13,16}=S8 P={14}=S9
	S9	D	D={14,16}=S10
Aceptación	S10	D $	D={14,16}=S10
Aceptación	S11	$	

	Estado	L	C	^”	#	‘’’	+	-	D	P	S
	S0	S1	S2		S5	S6	S7	S7	S8		S11
Aceptación	S1	S1									
	S2		S4	S3							
	S3		S4	S3							
Aceptación	S4										
	S5			S5							
	S6			S6		S4					
	S7								S8		
	S8								S8	S9	
	S9								S10		
	S10								S10		
Aceptación	S11										



Gramática Independiente del Contexto

Terminales={tk_cla, tk_reg, tk_imp, tk_impln, tk_cont, tk_prom,tk_contsi, tk_dat, tk_lei, tk_max, tk_min, tk_exp, tk_igual, tk_cadena, tk_num, tk_llavea, tk_llavec, tk_corchA, tk_corchC, tk_ptcom, tk_parentA, tk_parentC}

no Terminales={<inicio>, <claves>, <registros>, <instrucciones>, <cadenas>, <otracadena>, <variosregistros>, <valores>, <otrovalor>, <otroregistro>, <instrucciones>, <imprimir>, <imprimirln>, <conteo>, <promedio>, <contarsi>, <datos>, <leidos>, <max>, <min>, <exportarR>, <otrainstruccion> }

Producciones

<inicio> -> <claves> <registros><instrucciones>

<claves> -> tk_cla tk_igual tk_corchA <cadenas> tk_corchC

<cadenas> -> tk_cadena <otracadena>

<otracadena> -> tk_com tk_cadena <otracadena>
|e

<registros> -> tk_reg tk_igual tk_corchA <variosregistros> tk_corchC

<variosregistros> -> tk_llavea <valores> tk_llavec <otroregistro>

<otroregistro> -> tk_llavea <valores> tk_llavec <otroregistro>
|e

<valores> -> tk_cadena <otrovalor>
|tk_num <otrovalor>

<otrovalor> -> tk_com tk_cadena <otrovalor>
|tk_com tk_num <otrovalor>
|e

<instrucciones> -> <imprimir> <otrainstruccion>
|<imprimirln> <otrainstruccion>
|<conteo> <otrainstruccion>
|<promedio> <otrainstruccion>
|<contarsi> <otrainstruccion>
|<datos> <otrainstruccion>
|<leidos> <otrainstruccion>

|<max> <otrainstruccion>
|<min> <otrainstruccion>
|<exportarR><otrainstruccion>

<otrainstruccion> -> <imprimir> <otrainstruccion>
|<imprimirln> <otrainstruccion>
|<conteo> <otrainstruccion>
|<promedio> <otrainstruccion>
|<contarsi> <otrainstruccion>
|<datos> <otrainstruccion>
|<leidos> <otrainstruccion>
|<max> <otrainstruccion>
|<min> <otrainstruccion>
|<exportarR><otrainstruccion>
|e

<imprimir> -> tk_imp tk_parentA tk_cadena tk_parentC tk_ptcom

<imprimirln> -> tk_imp tk_parentA tk_cadena tk_parentC tk_ptcom

<conteo> -> tk_imp tk_parentA tk_parentC tk_ptcom

<promedio> -> tk_imp tk_parentA tk_cadena tk_parentC tk_ptcom

<contarsi> -> tk_imp tk_parentA tk_cadena tk_com tk_num tk_parentC tk_ptcom

<datos> -> tk_imp tk_parentA tk_parentC tk_ptcom

<leidos> -> tk_imp tk_parentA tk_parentC tk_ptcom

<max> -> tk_imp tk_parentA tk_cadena tk_parentC tk_ptcom

<min> -> tk_imp tk_parentA tk_cadena tk_parentC tk_ptcom

<exportarR> -> tk_imp tk_parentA tk_cadena tk_parentC tk_ptcom