UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE CIENCIAS Y SISTEMAS
LABORATORIO DE LENGUAJES FORMALES DE PROGRAMACION

PROYECTO #2
MANUAL DE TECNICO

DENNIS MAURICIO CORADO MUÑÓZ
CARNET: 202010406
CUI: 3032329780108
GUATEMALA, GUATEMALA, 28/10/2021

Objetivos y Alcances

Objetivos generales

Aplicar los conocimientos de clase y laboratorio para poder realizar la creación de un analizador Sintáctico por medio de la teoría de autómatas y así dar solución al problema planteado.

Objetivos específicos

• Analizar un archivo con una estructurada especificada y poder obtener información y validar su estructura

• Realizar un analizador sintáctico que determine errores en el archivo.

• Brindar información detallada de la estructura del archivo por medio de un reporte que contiene la lista de los tokens analizados, errores léxicos y errores sintácticos.

Especificación Técnica

Requisitos de Hardware (Mínimos)

- Microsoft Windows Vista
  - **Procesador:** Intel Pentium III o equivalente
  - **Memoria:** 512 MB
  - **Espacio en disco:** 750 MB libres

- Ubuntu 9.10
  - **Procesador:** Intel Pentium III o equivalente
  - **Memoria:** 512 MB
  - **Espacio en disco:** 650 MB libres

- Macintosh OS X 10.7 Intel
  - **Procesador:** Intel Pentium III o equivalente
  - **Memoria:** 512 MB
  - **Espacio en disco:** 750 MB libres

Cualquier hardware superior al establecido es capaz de ejecutar la aplicación y poder ver sus archivos por medio de un editor de código.

Requisitos de Software

- **Sistema Operativo:** Windows, masOS, Linux
- **Lenguaje de Programación:** Python 3.9 o superior
- **IDE:** Visual Studio Code, Intellij, Pycharm

Lógica del Programa

- **Clases, métodos y sus funciones dentro del programa:**
  El programa cuenta con varios archivos que buscan generar una estructura del código mas limpia y clara, por lo tanto el código se divide entre archivos que tienen una función específica.
    - Clase Gui: esta clase hace uso de la librería PyQt5 para poder generar la interfaz gráfica, dentro de esta clase se hace uso de OOP para poder darle funciones a los respectivos botones y cajas de texto dentro del programa.
    - Clase main: esta clase se encarga de realizar una instancia de la clase gui para poder iniciar el programa.
    - Clase Analizador: acá se encuentra un analizador léxico que haciendo uso de la teoría de autómatas se programo y verifica que los símbolos, palabras y números, pertenezcan al lenguaje utilizado, dentro del método reader de esta clase se realiza todo el análisis léxico y se apoya con la clase 'token', en la clase reader se lee el archivo dado carácter por carácter y se identifica que sean tokens que estén definidos en el sistema, esta clase cuenta con diferentes métodos, método 'agregarToken' como su nombre lo indica se encarga de agregar un token valido a un arreglo que contiene los objetos de tipo token, únicamente tokens valido. Método 'agregarError' similar al método agregarToken este método agrega los tokens desconocidos, que no pertenezcan al lenguaje a un arreglo que contiene dichos objetos. Método 'palabra_reservada', este método verifica a cuál de las palabras reservadas definidas por el lenguaje pertenece la palabra analizada, si no pertenece a ninguna palabra se procede a marcar dicha palabra como desconocida. Método instrucciones, similar al método anterior se encarga de verificar que la palabra obtenida pertenezca a algunas de estas palabras ya definidas si no pertenece se marca como desconocida. Los siguientes métodos tienen funciones simples, generalmente de devolver un valor especifico. Método 'can_generate' retorna un booleano que su valor en verdadero si no se encuentran errores léxicos y falsos si los hay. Método 'getTokens' devuelve el arreglo con los tokens validos analizados. Método 'getErrores' devuelve un arreglo con los tokens desconocidos. Método 'getErroresSint' devuelve los errores sintácticos una vez realizado el análisis sintáctico de la información. Método 'imprimir' se encarga de mostrar la información de los tokens leídos en consola. Método 'setTextEditor' se encarga de obtener la caja de texto donde se mostrará las funciones de los comandos utilizados en el archivo.
    - Clase Sintáctico:
      Dentro de esta clase se encuentra el proceso para realizar el análisis sintáctico, este análisis se realiza haciendo uso de la gramática de tipo 2 definida más abajo de este documento. Para poder programar esta gramática se debe hacer uso de recursividad, dentro de esta clase se encontraran definidos los campos descritos en la gramática en forma de funciones todos hacen uso del método 'match' que se encarga de verificar que el token proporcionado sea igual al token que está siendo leído en ese momento, de no ser igual se produce un error sintáctico, se deja de analizar el archivo, así como también se muestra por la caja de texto que muestra las salidas el error determinado y que se esperaba que viniese.

- Clase Funciones:
  Esta clase entra en ejecución después de realizar el análisis sintáctico, si el analizador sintáctico cuenta con un error no se hace uso de esta clase, pero si todo lleva la estructura correcta se proceden a realizar las instrucciones especificadas en el archivo bajo los comandos determinados. Prácticamente esta clase se encarga de aplicar los diferentes comandos que pueden ser utilizados en la aplicación y mostrar los resultados deseados en la caja de texto pertinente.
- Archivo Reportes:
  Dentro de este archivo se encuentran 3 clases cada una con un objetivo similar pero muestran diferente información, se encargan de generar un archivo HTML con la información pertinente de cada reporte, la clase 'Reporte_Funcion' se encarga de generar una tabla HTML con los registros y claves en el archivo. La clase 'Reporte_Tokens' genera un archivo HTML con los tokens validos analizados. 'Reporte_Errores' se encarga de devolver 2 tablas HTML una con los errores léxicos en el programa y otra con los errores sintácticos.
- Clase Token:
  El objetivo de esta clase es crear un tipo de clase enumerada con los diferentes tokens que soporta el programa, cada uno de estos esté definido por la estructura del archivo de entrada.
- Clase Árbol:
  Se encarga de generar un árbol de derivación haciendo uso de la herramienta graphviz, de los diferentes tokens analizados en el analizador sintáctico.

Analizador Léxico

Tabla de Tokens y Expresiones Regulares

Letras = L = [a-z, A-Z]+ = L+
Dígitos = D = [0 – 9]+ = D+
Resto = R = [Cualquier símbolo]
Signos = S = [igual, corchete-izq, corchete-der, coma, llave-izq, llave-der, paréntesis-izq, paréntesis-der]
Cadenas = “(L|D|R)*”
Números = (D+.D+) | (D+)
Comentario = (#(L|D|R)*) | (“’(L|D|R)*”’)

*****Debido a que el carácter ‘#’ estaría dentro de la definición dentro de la expresión de comentario se utilizó como estado de aceptación el signo ‘$’

Expresión Regular

Expresión regular obtenida en base a la tabla de tokens y la definición de las expresiones regulares que contaría el programa.

[(L)+ | (N)+ | (S)+ | “(L|D|R)*” | (D+.D+) | (D+) | (#(L|D|R)*) | (“‘ (L|D|R)*’”)]$

Desarrollo del Método del árbol

Árbol de la expresión regular:

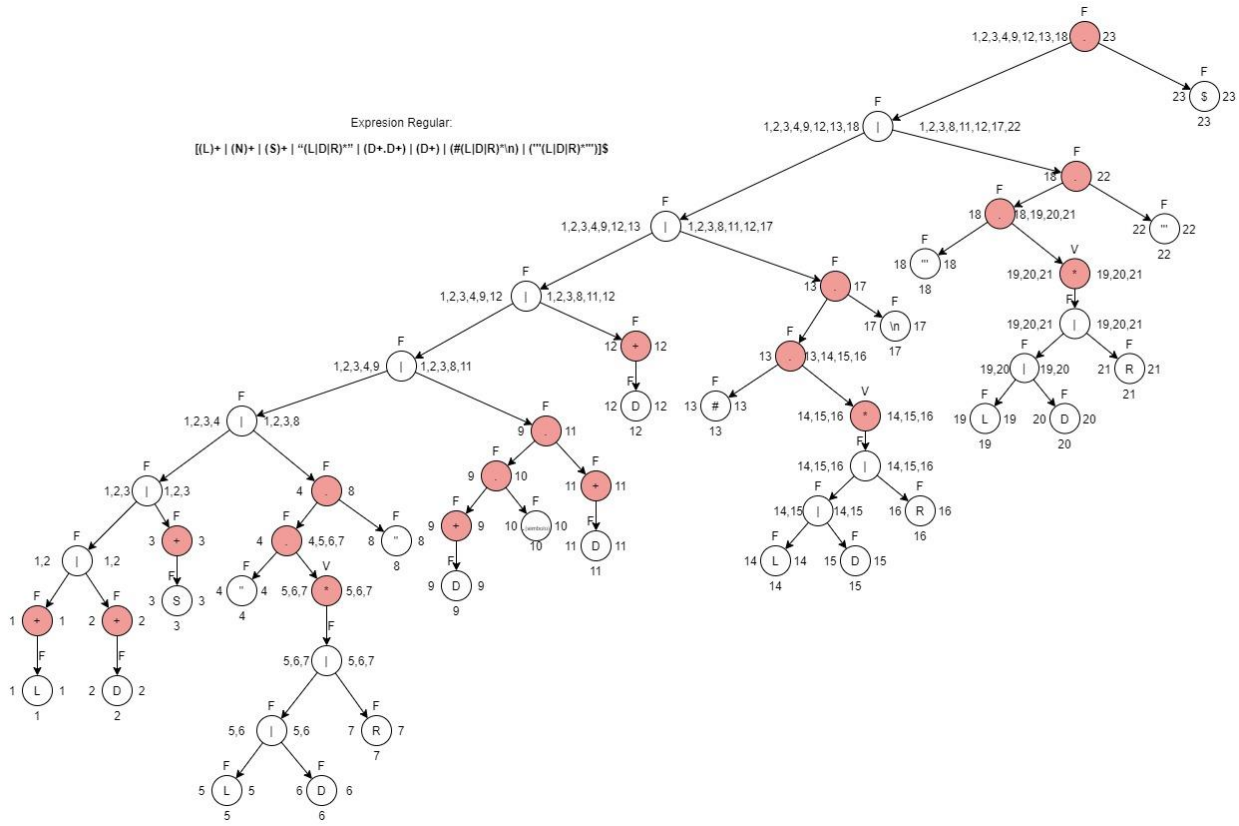


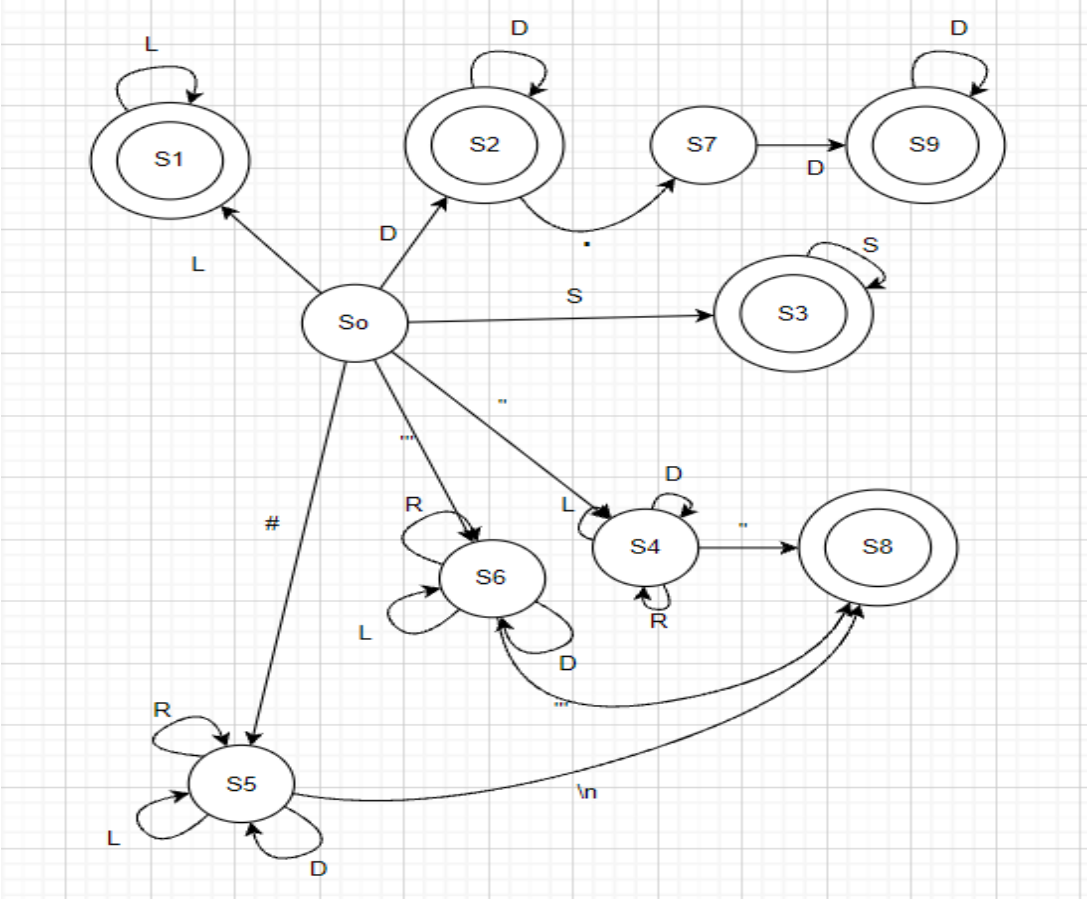
Tabla de Siguientes y gramática regular:

i	Valor	Sig(i)
1	L	1,23
2	D	2,23
3	S	3,23
4	"	5,6,7,8
5	L	5,6,7,8
6	D	5,6,7,8
7	R	5,6,7,8
8	"	23
9	D	9,10
10	.	11
11	D	11,23
12	D	12,23
13	#	14,15,16,17
14	L	14,15,16,17
15	D	14,15,16,17
16	R	14,15,16,17
17	\n	23
18	""	19,20,21,22
19	L	19,20,21,22
20	D	19,20,21,22
21	R	19,20,21,22
22	""	23
23	$	

So = {1,2,3,4,9,12,13,18} L D S " D D # ""
Sig(L) = Sig(1) = {1,23} = S1
Sig(D) = Sig(2) U Sig(9) U Sig(12) = {2,9,10,12,23} = S2
Sig(S) = Sig(3) = {3,23} = S3
Sig(") = Sig(4) = {5,6,7,8} = S4
Sig(#) = Sig(13) = {14,15,16,17} = S5
Sig("") = Sig(18) = {19,20,21,22} = S6
S1 = {1,23} L $
Sig(L) = Sig(1) = {1,23} = S1
S2 = {2,9,10,12,23} D D . D $
Sig(D) = Sig(2) U Sig(9) U Sig(12) = {2,9,10,12,23} = S2
Sig(.) = Sig(10) = {11} = S7
S3 = {3,23} S $
Sig(S) = Sig(3) = {3,23} = S3
S4 = {5,6,7,8} L D R "
Sig(L) = Sig(5) = {5,6,7,8} = S4
Sig(D) = Sig(6) = {5,6,7,8} = S4
Sig(R) = Sig(7) = {5,6,7,8} = S4
Sig(") = Sig(8) = {23} = S8
S5 = {14,15,16,17} L D R \n
Sig(L) = Sig(14) = {14,15,16,17} = S5
Sig(D) = Sig(15) = {14,15,16,17} = S5
Sig(R) = Sig(16) = {14,15,16,17} = S5
Sig(\n) = Sig(17) = {23} = S8
S6 = {19,20,21,22} L D R ""
Sig(L) = Sig(19) = {19,20,21,22} = S6
Sig(D) = Sig(20) = {19,20,21,22} = S6
Sig(R) = Sig(21) = {19,20,21,22} = S6
Sig("") = Sig(22) = {23} = S8
S7 = {11} D
Sig(D) = Sig(11) = {11,23} = S9
S8 = {23} $

S9 = {11,23} D $
Sig(D) = Sig(11) = {11,23} = S9

Autómata Finito Determinista



Analizador Sintáctico

Gramática Independiente del Contexto:

<Inicio> ::= <Claves> <Inicio>

| <Registros> <Inicio>

| <Comentario> <Inicio>

| <Imprimir> <Inicio>

| <Imprimir_ln> <Inicio>

| <Conteo> <Inicio>

| <Promedio> <Inicio>

| <Contar_Si> <Inicio>

| <Datos> <Inicio>

| <Sumar> <Inicio>

| <Max> <Inicio>

| <Min> <Inicio>

| <Exportar_Reporte> <Inicio>

<Claves> ::= tk_clave tk_igual tk_corA <Bloque_Claves> tk_corC

<Registros> ::= tk_registros tk_igual tk_corA <Bloque_Registros> tk_corC

<Comentario> ::= tk_comentario

<Imprimir> ::= tk_imprimir tk_parA tk_cadena tk_parC tk_ptComa

<Imprimir_ln> ::= tk_imprimir_ln tk_parA tk_cadena tk_parC tk_ptComa

<Conteo> ::= tk_conteo tk_parA tk_parC tk_ptComa

<Promedio> ::= tk_promedio tk_parA tk_cadena tk_parC tk_ptComa

<Contar_Si> ::= tk_contar_si tk_parA tk_cadena tk_coma tk_numero tk_parC tk_ptComa

<Datos> ::= tk_datos tk_parA tk_parC tk_ptComa

<Sumar> ::= tk_sumar tk_parA tk_cadena tk_parC tk_ptComa

<Max> ::= tk_max tk_parA tk_cadena tk_parC tk_ptComa

<Min> ::= tk_min tk_parA tk_cadena tk_parC tk_ptComa

<Exportar_Reporte> ::= tk_exportar tk_parA tk_cadena tk_parC tk_ptComa

<Bloque_Claves> ::= tk_cadena<Bloque_Claves>

| tk_cadena <Bloquec_coma>

<Bloquec_Coma> ::= tk_coma tk_cadena <Bloque_Claves>

| Epsilon

<Bloque_Registros> ::= <Cuerpo_Registro> <Bloque_Registros>

| <Cuerpo_Registro>

<Cuerpo_Registro> ::= tk_llaveA <Cuerpo_valor> tk_llaveC

<Cuerpo_valor> ::= tk_numero <Valorc_coma>

| tk_cadena <Valorc_coma>

<Valorc_coma> ::= tk_coma tk_numero <Cuerpo_valor>

| tk_coma tk_cadena <Cuerpo_valor>