CENTRO UNIVERSITARIO DE OCCIDENTE “CUNOC”
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
INGENIERÍA EN CIENCIAS Y SISTEMAS

“Analizador Sintáctico”

Dylan Amilcar Barrios Monterroso
201830409

Indice

“Manual de Usuario, Analizador Sintáctico”

En este manual se presenta un analizador de código, el cual se basa en recibir un archivo con extensión “txt”, este programa luego de leer archivo procede a realizar dos análisis:

-análisis Léxico

El archivo se separa por tokens y se analiza uno por uno a través de una matriz de estados la cual tiene estados de aceptación basado en el lenguaje que se quiera reconocer, si el token termina en un estado de aceptación se verifica que estado es para clasificarlo de lo contrario se muestra como un error.

-análisis sintáctico

El archivo analizado lexicamente se analiza ahora para saber si es correcto el tipo de dato con el dato asignado, esto se realiza a traves de otra una matriz de valores en las cuales se estará moviendo el token. Esta matriz se construye a través de la gramática que se quiere permitir y creando el analisis LL(1).

Instrucciones de Programa

1. Instalar JDK.
2. Ejecutar JAR (ejecutable).
3. Declarar función Principal
4. Buscar Archivo de Texto a Analizar.
5. Analizar Archivo Lexicamente.
6. Analizar Archivo SIntacticamente

Requerimientos Técnicos

- SO que soporte Java (Ejemp.: Linux, Windows, IOs, Android).
- Java 1.8.0_201 o compatibles.
- NetBeans IDEA (creación: v. 8.2) o cualquier editor de lenguaje JAVA.
- .Jar ejecutable o proyecto completo.
- Experiencia con aplicaciones Java (funcionamiento).

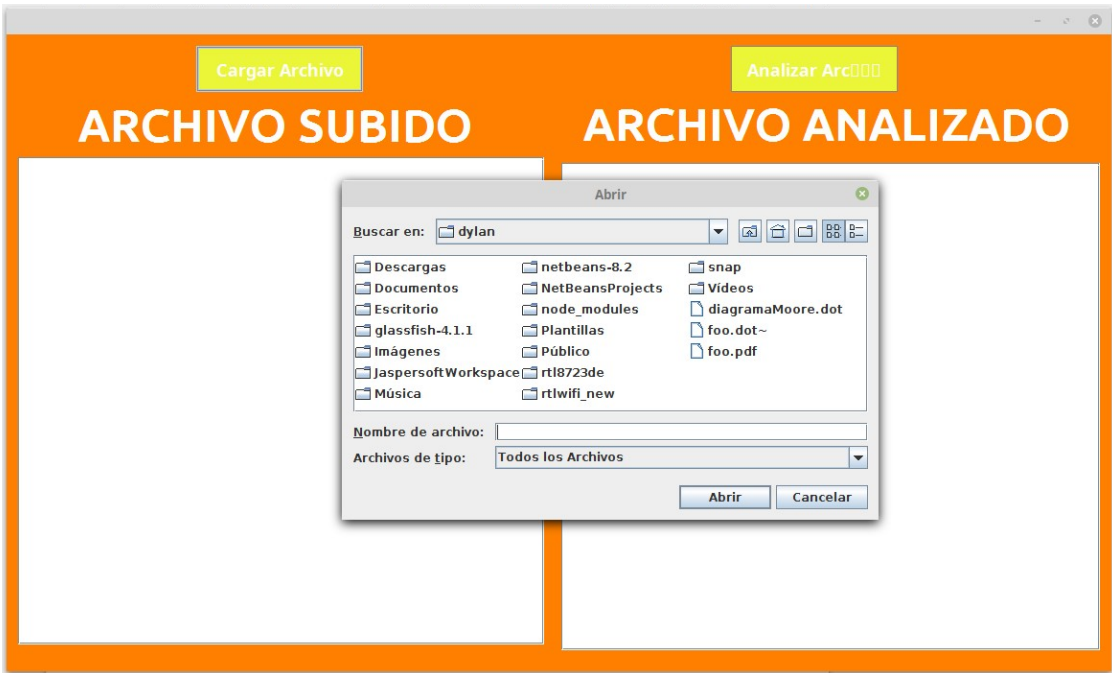
Como Acceder al Programa

Descargar el analizador desde github en el siguiente enlace: https://github.com/DylanBarrios/201830409_Dylan_Barrios_P2.git.
Descargarlo como ZIP O clonar el repositorio.

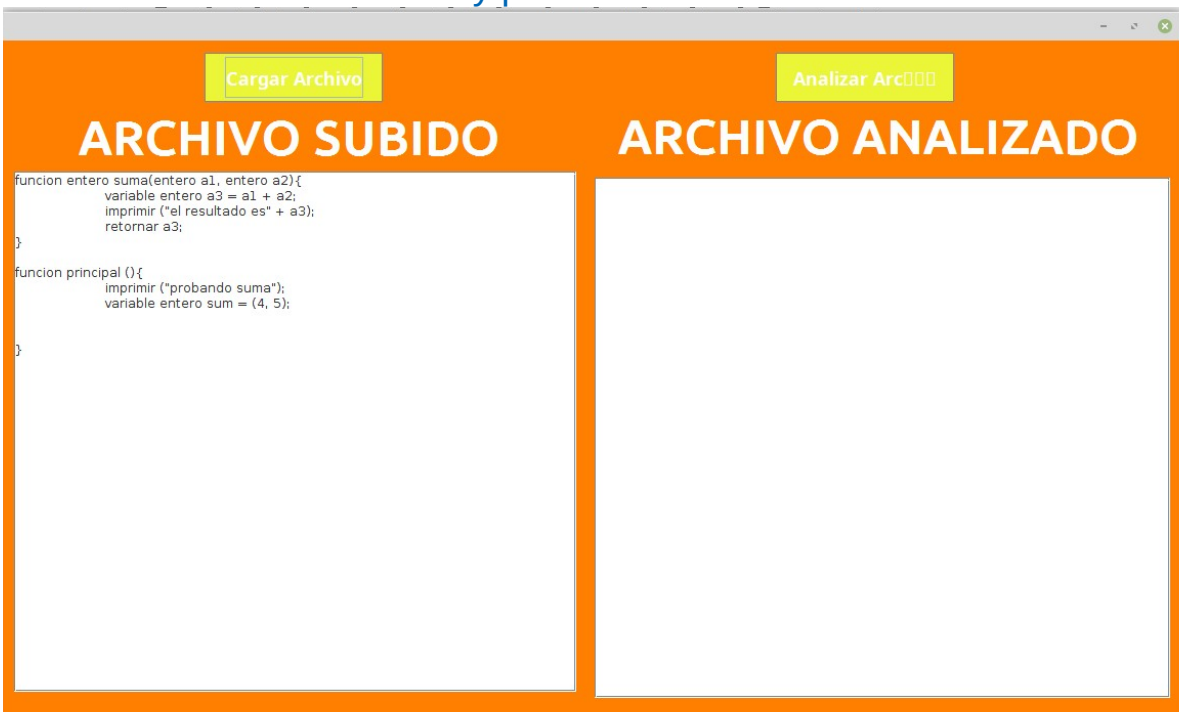
Pagina principal

Analizar archivo

Paso 1: Presionar Cargar Archivo

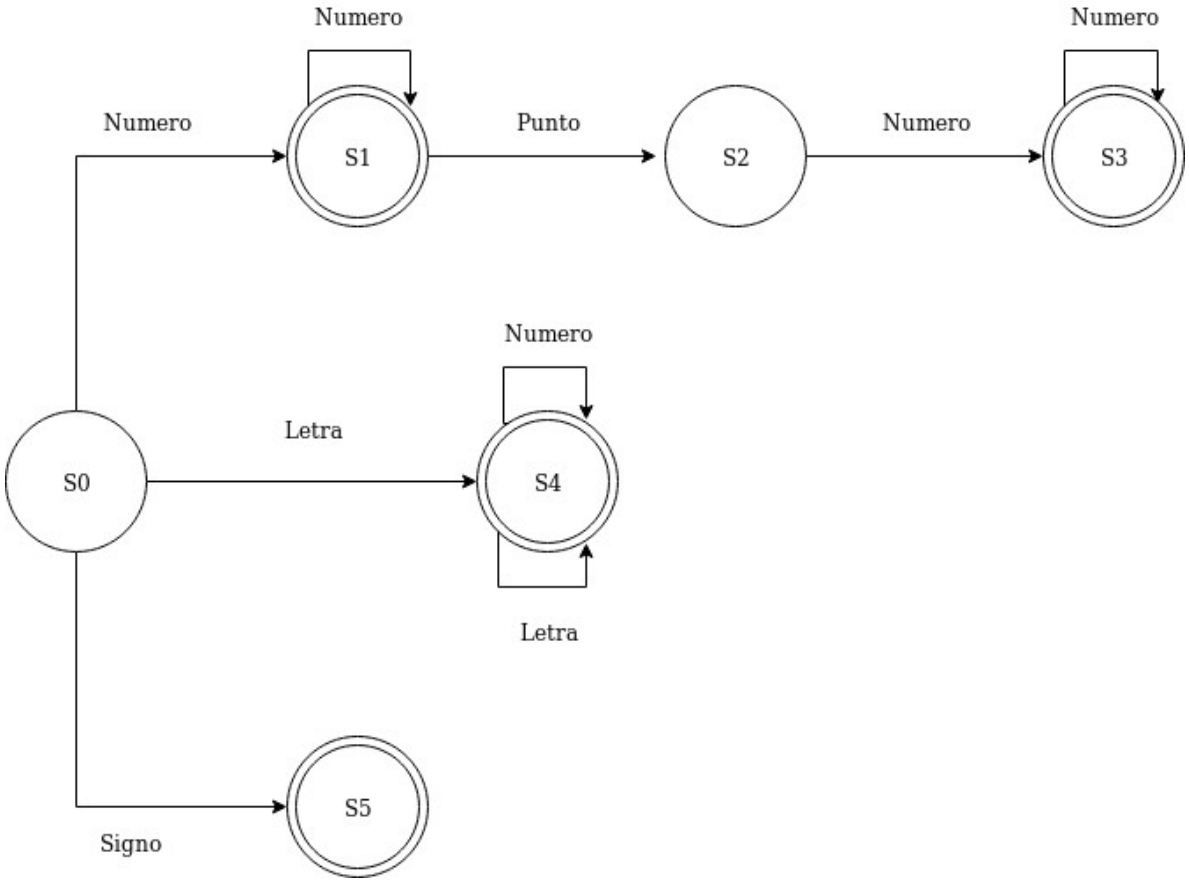


Paso 2: Selecciona el archivo y presionar abrir



ANALISIS LEXICO

Autómata para análisis Lexico



ANÁLISIS SINTÁCTICO

E --> funcion principal { **C** }
funcion ID (**PARAMETRO**) { **C** }
C --> **V ;**
V = **T L** ;
imprimir ( );
while( ){ **C** }
if ( ) { C }
for (**V** = **NUM ;** ID **<** **NUM ;** ID++) { **C** }
ID (ID) ;
V --> variable **D** ID
D --> entero
decimal
booleano
cadena
caracter
T --> **NUM**
NUM . **NUM**
true
false
"cadena"
L --> ID **OPERADOR** ID
e
PARAMETRO --> **D** ID
e
NUM --> **DIGITO DIGITO**
DIGITO --> 0|1|2|3|4|5|6|7|8|9
OPERADOR --> +|-|*|/|%|=|==|<|>|>=|<=

No terminal	Primeros
E	funcion principal, ID
C	Variable, imprimir, while, if, for, ID
V	Variable
D	entero, decimal, booleano, carácter, cadena
T	0, 1, 2, 3, 4, true, false, cadena
L	Id, e
Parametro	entero, decimal, booleano, carácter, cadena, e
Num	0, 1, 2, 3, 4 , 5 ,6 ,7 ,8 ,9
Dig	0, 1, 2, 3, 4 , 5 ,6 ,7 ,8 ,9
Ope	+|-|*|/|%|=|==|<|>|>=|<=

No terminal	Siguientes
E	$
C	$
V	; ,
D	id
T	id, e, $
L	;
Parametro	variable, imprimir, whie
Num	; , id, e, $
Dig	; , id, e , $
Ope	id

	funcion principal	id	variable	imprimir	mientras
E	funcion principal { **C** }	funcion ID (PARAMETRO) { C }			
C		for (**V** = **NUM ;** ID **< NUM ;** ID++) { **C** }	V ;	**V** = **T L** ;	imprimir ( );
V			variable **D** ID		
D					
T					
L		ID **OPERADOR** ID			
Parametro			e	e	e
Num					
Dig					
Ope					

si	para	entero	decimal	booleano	carácter
while( ){ **C** }	if ( ) { C }				
		entero	decimal	booleano	carácter
		D ID	e	e	e

cadena	**true**	**false**	**1, 2, 3**	**+|-|*|/|%|=|==|<|>|>=|<=**	**;**	**$**
cadena						
false	NUM.NUM	true	NUM			
					e	
e						
			DIGITO DIGITO			
			0|1|2|3|4|5|6|7|8|9			
				+|-|*|/|%|=|==|<|>|>=|<=		