UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Traductores de Lenguaje II

Reporte de práctica

Nombre del alumno:	Nombre del alumno
Profesor:	Erasmo Gabriel Martínez Soltero
Título de la práctica:	"Tarea 02 - Analizador Léxico"
Fecha:	14 septiembre 2021

Introducción

En esta actividad se realizará un analizador léxico capaz de reconocer los *tokens y lexemas* de una cadena ingresada. Recordemos que los tokens o también llamadas *componentes léxicos* son una secuencia de caracteres con un significado sintáctico propio, por ejemplo, los tipos de datos o id's son considerados como tokens. Por otra parte, entendemos como lexema a aquella secuencia de caracteres cuya estructura se corresponde con el patrón de un token, por ejemplo, las cadenas *int, float, char, void, etc* son los posibles lexemas del token *tipo de dato*.

Para llevar a cabo el analizador, será necesario implementar una interfaz gráfica en la que se le permita al usuario poder ingresar texto para posteriormente se analizado, además, se deberán de mostrar los resultados obtenidos en una tabla en la cual se mostrarán todos los resultados, es decir, la lista de tokens, lexemas y sus correspondientes números. Se deberán contemplar todos los tokens especificados en clase, respetando los números indicados ya que serán importantes para futuras actividades.

Metodología

Los tokens y lexemas a contemplar para esta actividad son los siguientes:

Lexema	**Número**
Tipo de dato	0
Id	1
;	2
,	3
(	4
)	5
{	6
}	7
=	8
if	9
while	10
return	11
else	12
Constante	13
opSuma	14
opLogico	15
opMultiplicacion	16
opRelacional	17
$	18

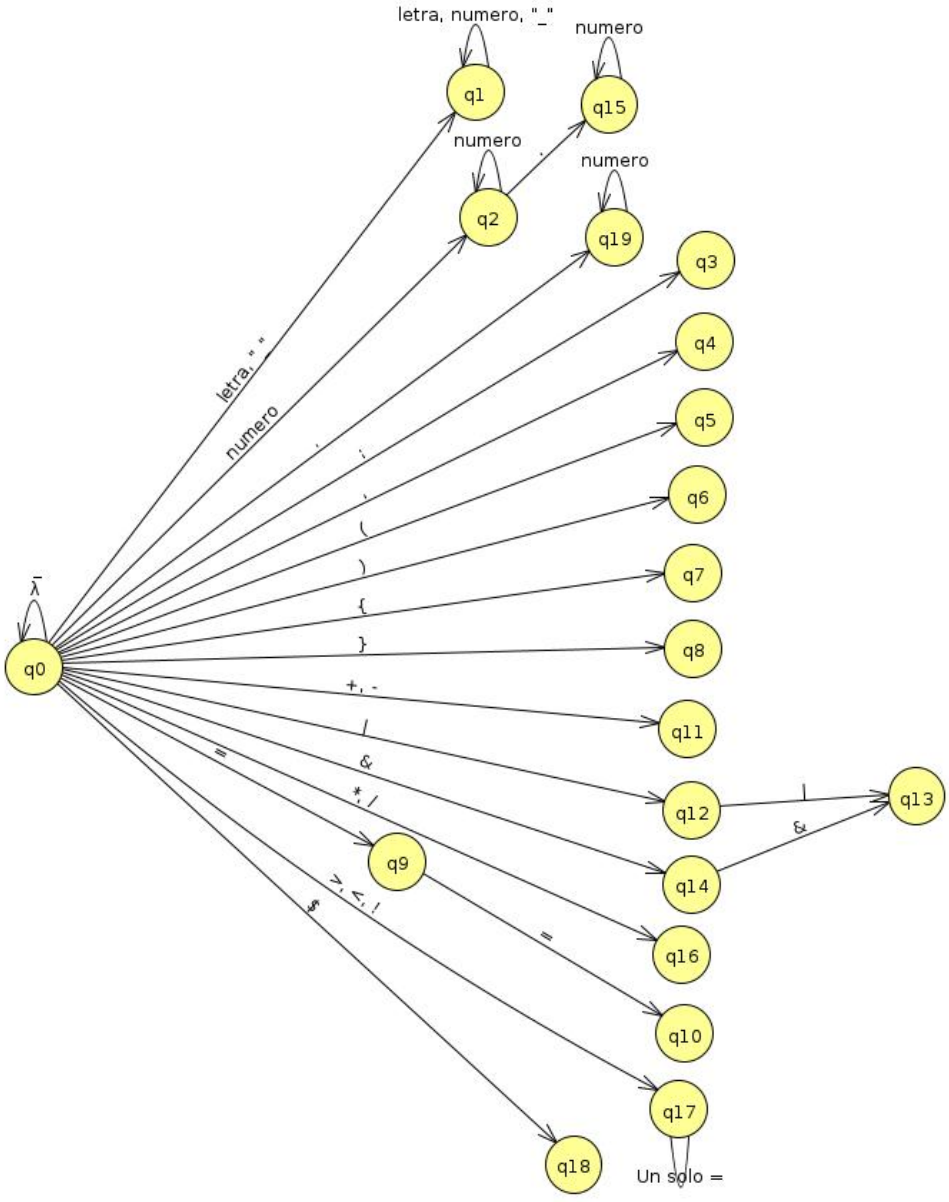

Figura 1: Autómata diseñado para el analizador léxico

Teniendo en cuenta lo anterior, se desarrolló la interfaz gráfica necesaria utilizando ***Python*** y ***PyQt5***, en la cual se agregan los espacios correspondientes para introducir el código a analizar, el área donde se mostrarán los resultados obtenidos y un par de botones mediante los cuales se ejecutarán las acciones básicas del programa. De igual manera, se deja el espacio para el futuro código en ensamblador (esto en el caso del seminario, para esta actividad no tiene ni tendrá ningún uso este espacio).

Cabe mencionar que la interfaz contempla los casos en los que se quiera analizar una cadena vacía y que reinicia", por decirlo de alguna manera, el programa cada que se ingresa una nueva cadena o también permite limpiar toda la pantalla mediante el uso del botón correspondiente. Son detalles simples pero que ayudan a que el analizador no sea tan tedioso de probar.

A continuación, se muestra la interfaz diseñada para esta actividad:

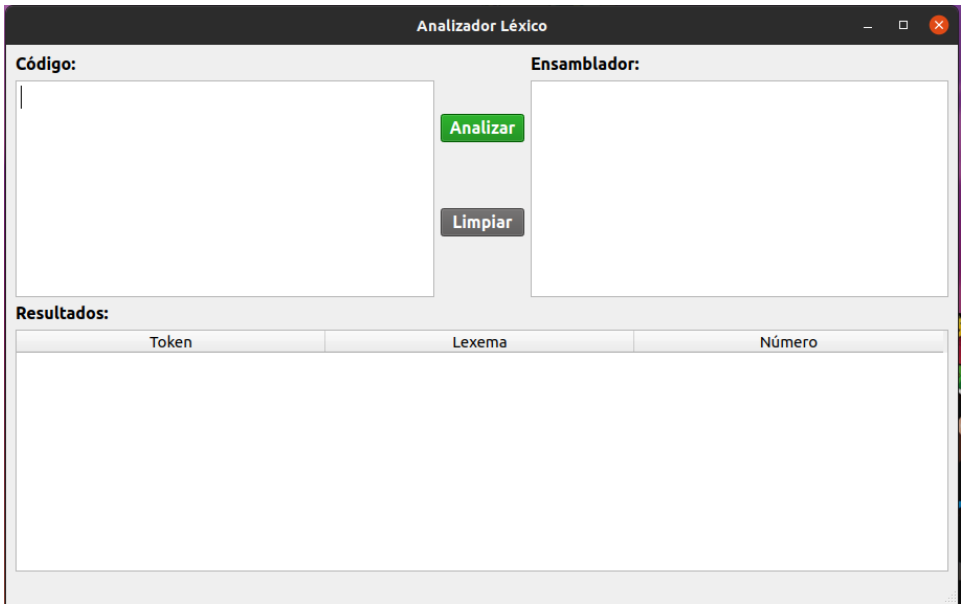

Figura 2: Interfaz gráfica diseñada para el analizador léxico

Respecto a la codificación del autómata previamente ilustrado, es una clase separada la cual tendrá 2 atributos importantes: *El estado en el que se encuentra* y *una lista* con todos los resultados que se obtendrán; al tratarse de Python, esta estructura es más bien un diccionario de datos.

El funcionamiento del autómata es bastante similar al visto en clase y al descrito en la 1. El algoritmo se compone de dos ciclos principales, al comenzar a analizar la cadena, le concatenamos el sigo de pesos al final de esta. El ciclo principal nos ayuda a colocarnos en el estado inicial e ir recorriendo los caracteres de la cadena uno por uno. Por su parte, el ciclo anidado al principal, se encarga de recorrer la cadena también pero es aquí donde se

asignan los estados correspondientes según el carácter leído, la única manera de salir de este ciclo será entonces el terminar de leer la cadena o el entrar en el estado de *ESCAPE*.

El estado de escape es un estado al que accedemos cuando al estar leyendo la cadena y estar en un estado determinado, se lee algo que no pertenece al estado en el que se encuentra, por ejemplo, si llevamos varios caracteres leídos y nos encontramos en el estado de *ID* pero nos llega un punto y coma, el autómata no contempla que los ID's lleven un punto y coma, por lo que pasamos al estado de escape y no pasamos al siguiente carácter de la cadena, sino que con ese mismo punto y coma volvemos al estado inicial y procedemos a verificar el token, lexema y estado al que nos cambia.

El *token* que se use dependerá de la transición de los estados que se vaya realizando, mientras que el *lexema* contendrá todos los caracteres que se fueron leyendo mientras estos permanecían en el estado correspondiente al token. De esta manera, cuando entramos al estado de escape, además de cambiar al estado inicial, añadimos el token, lexema y el número correspondiente al diccionario de resultados. En caso de que los caracteres leídos no pertenezcan a ningún estado contemplado por el autómata, el token almacenará un .$^{\text{er}}$rorz el lexema quedará vacío.

Finalmente, cuando toda la cadena haya sido analizada, se procede a cambiar el token y valor de todas las palabras reservadas que hayan quedado catalogadas como un ID. Esto facilita mucho las cosas, ya que de esta manera no hace falta que creemos un autómata tan complejo en el que se contemplen los casos de cada palabra reservada. Cuando ya tenemos el diccionario de todos los resultados obtenidos, lo retornamos a la vista del programa y procedemos a insertar los valores correspondientes en una tabla.

Resultados

Los resultados obtenidos han sido positivos ya que el autómata implementado funciona correctamente y tras varias pruebas realizadas se observa como hace correctamente la separación de los componentes de la cadena y los clasifica en los tokens correctos.

A continuación se muestra una serie de capturas de pantalla del programa funcionando:

0.1. Prueba 1

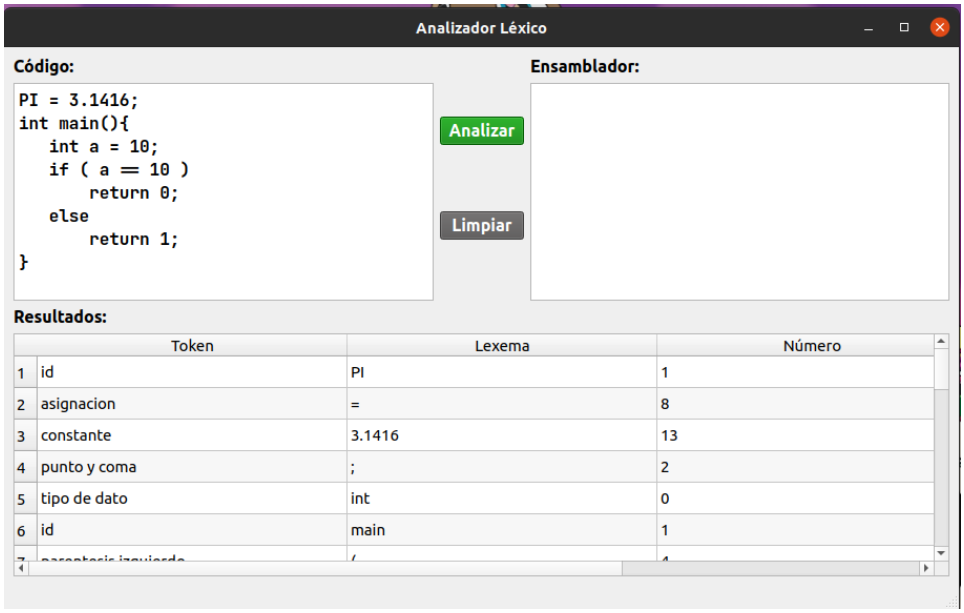

Figura 3: Prueba 1, cadena ingresada y resultados del 1 al 6

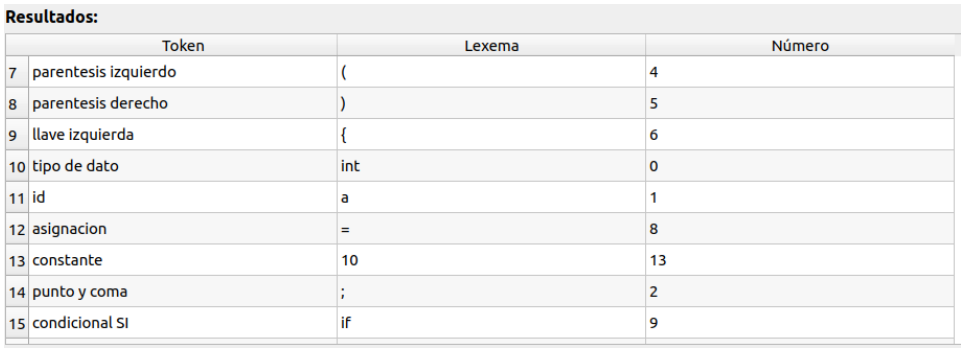

Figura 4: Prueba 1, resultados del 7 al 15

Resultados:

	Token	Lexema	Número
16	parentesis izquierdo	(	4
17	id	a	1
18	operador relacional	==	17
19	constante	10	13
20	parentesis derecho	)	5
21	retorno	return	11
22	constante	0	13
23	punto y coma	;	2
24	condicional SI NO	else	12

Figura 5: Prueba 1, resultados del 16 al 24

Resultados:

	Token	Lexema	Número
21	retorno	return	11
22	constante	0	13
23	punto y coma	;	2
24	condicional SI NO	else	12
25	retorno	return	11
26	constante	1	13
27	punto y coma	;	2
28	llave derecha	}	7
29	pesos	$	18

Figura 6: Prueba 1, resultados del 25 al 29

0.2. Prueba 2

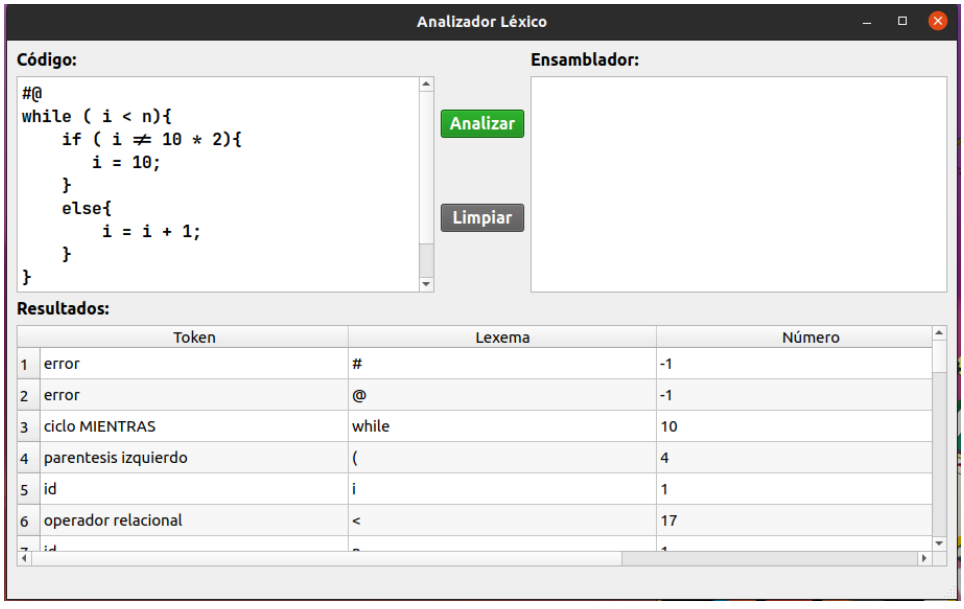

Figura 7: Prueba 2, cadena ingresada y resultados del 1 al 6

Resultados:

	Token	Lexema	Número
7	id	n	1
8	parentesis derecho	)	5
9	llave izquierda	{	6
10	condicional SI	if	9
11	parentesis izquierdo	(	4
12	id	i	1
13	operador relacional	!=	17
14	constante	10	13
15	operador multiplicacion	*	16

Figura 8: Prueba 2, resultados del 7 al 15

Resultados:

	Token	Lexema	Número
16	constante	2	13
17	parentesis derecho	)	5
18	llave izquierda	{	6
19	id	i	1
20	asignacion	=	8
21	constante	10	13
22	punto y coma	;	2
23	llave derecha	}	7
24	condicional SI NO	else	12

Figura 9: Prueba 2, resultados del 16 al 24

Resultados:

	Token	Lexema	Número
25	llave izquierda	{	6
26	id	i	1
27	asignacion	=	8
28	id	i	1
29	operador suma	+	14
30	constante	1	13
31	punto y coma	;	2
32	llave derecha	}	7
33	llave derecha	}	7

Figura 10: Prueba 2, resultados del 25 al 33

Resultados:

	Token	Lexema	Número
29	operador suma	+	14
30	constante	1	13
31	punto y coma	;	2
32	llave derecha	}	7
33	llave derecha	}	7
34	retorno	return	11
35	constante	1	13
36	punto y coma	;	2
37	pesos	$	18

Figura 11: Prueba 2, resultados del 33 al 35

0.3. Prueba 3

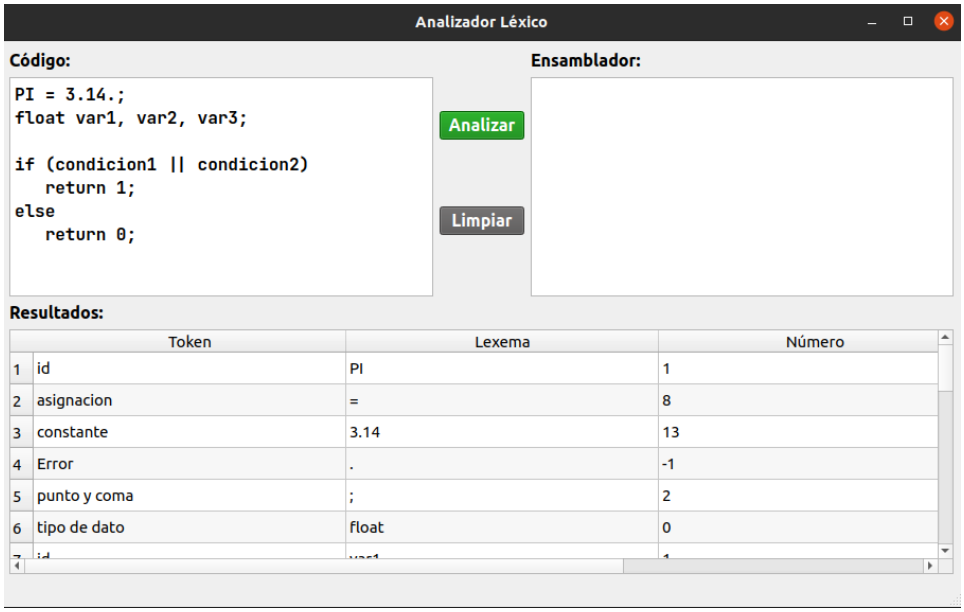

Figura 12: Prueba 3, cadena ingresada y resultados del 1 al 6

Resultados:

	Token	Lexema	Número
7	id	var1	1
8	coma	,	3
9	id	var2	1
10	coma	,	3
11	id	var3	1
12	punto y coma	;	2
13	condicional SI	if	9
14	parentesis izquierdo	(	4
15	id	condicion1	1

Figura 13: Prueba 3, resultados del 7 al 15

Resultados:

	Token	Lexema	Número
15	id	condicion1	1
16	operador logico	||	15
17	id	condicion2	1
18	parentesis derecho	)	5
19	retorno	return	11
20	constante	1	13
21	punto y coma	;	2
22	condicional SI NO	else	12
23	retorno	return	11

Figura 14: Prueba 3, resultados del 16 al 23

Resultados:

	Token	Lexema	Número
18	parentesis derecho	)	5
19	retorno	return	11
20	constante	1	13
21	punto y coma	;	2
22	condicional SI NO	else	12
23	retorno	return	11
24	constante	0	13
25	punto y coma	;	2
26	pesos	$	18

Figura 15: Prueba 3, resultados del 24 al 26

Conclusiones

Llevar acabo este analizador léxico resultó ser una oportunidad para reaprender muchas de las cosas vistas en la clase de *teoría de la computación* ya que tenía tiempo sin dibujar un autómata ni mucho menos representarlo en código. Además, me animé a realizarlo en Python porque es un lenguaje que quiero seguir practicando, es por eso que decidí implementar la interfaz gráfica con un *framwework* como lo es PyQt5.

En términos generales, implementar el autómata no resultó tan complicado como esperaba, si que es verdad que al tener que contemplar tantos estados resulta un tanto tedioso pero de ahí en más todo fluyó correctamente. Me pareció interesante y sobre todo útil la idea de cachar las palabras reservadas como un ID para posteriormente solo reemplazar estas palabras con el token correspondiente, esto es algo que no solo nos ahorró tiempo sino también disminuyó de manera significativa la complejidad del código y el autómata.

He de mencionar que estoy bastante satisfecho con los resultados obtenidos y también el hecho de que ya lo hubiera realizado para el seminario previamente, me permitió factorizar algunas aspectos relacionados con la in-

terfaz o el como mostraba los datos en la tabla. Como conclusión, puedo decir que me ha parecido un programa bastante interesante de llevar a cabo debido a que nunca había realizado un analizador léxico antes y considero que las herramientas utilizadas me permitieron aprender cosas nuevas sobre el lenguaje. También cabe recalcar que la explicación realizada en clase sobre el funcionamiento de este y el código proporcionado fueron de suma importancia para llevar a cabo esta actividad.

Referencias

- Explicaciones y código visto en la clase de Traductores de Lenguaje II impartida por Erasmo Gabriel Martinez Soltero.