INSTITUTO TECNOLÓGICO JOSÉ MARIO MOLINA PASQUEL Y HENRÍQUEZ

UNIDAD ACADÉMICA LAGOS DE MORENO

Ingeniería (Sistemas Computacionales)

“Proyecto Parcial III”

Mtro. Néstor Chico Rojas

(Carlos Antonio Zambrano Gómez)

Lagos de Moreno, Jal., a (28 de mayo de 2022)

Introducción.

Un compilador permite comprender y desarrollar mejor una lógica de la programación debido al desarrollo del mismo. Por eso la razón de este sencillo compilador realizado con un objeto simple que permite comprender fácilmente dicha lógica.

Logo.

Objetivo General.

Desarrollo de un compilador diseñado para una catapulta que permita el funcionamiento de la misma mediante Arduino.

Análisis Léxico.

Descripción.

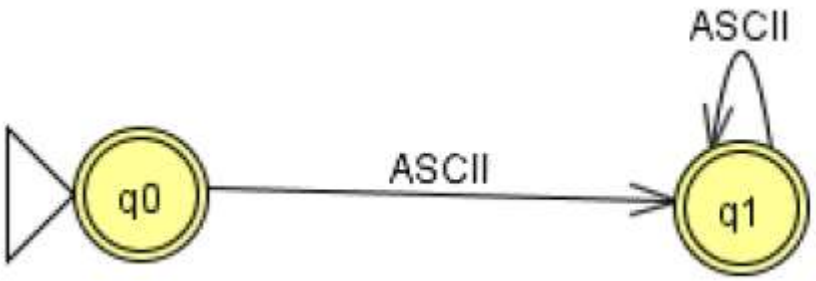
Esta es la primera fase del compilador, la cual permite separar e identificar los tokens (palabras) de un código.

Gramática del compilador (autómatas, matriz de transición y expresiones regulares).

Palabras reservadas.

- Inicio.
- Fin.

ER->ASCII(ASCII)*

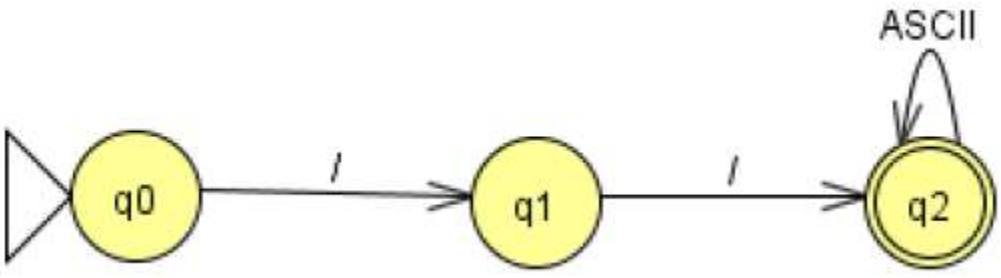


.	ASCII
q0	q1
q1	q1

Comentarios.

- //Soy un comentario.

ER-> //(ASCII)*

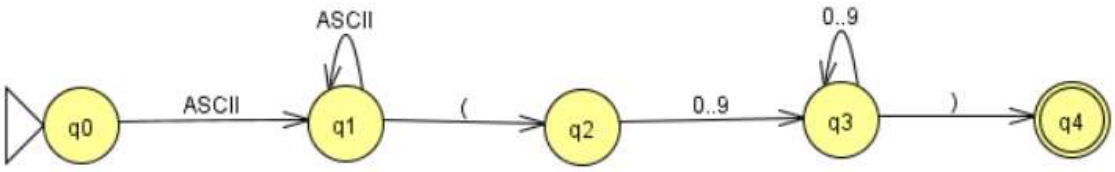


.	/	ASCII
q0	q1	
q1	q2	
q2		q2

Instrucciones / Expresiones.

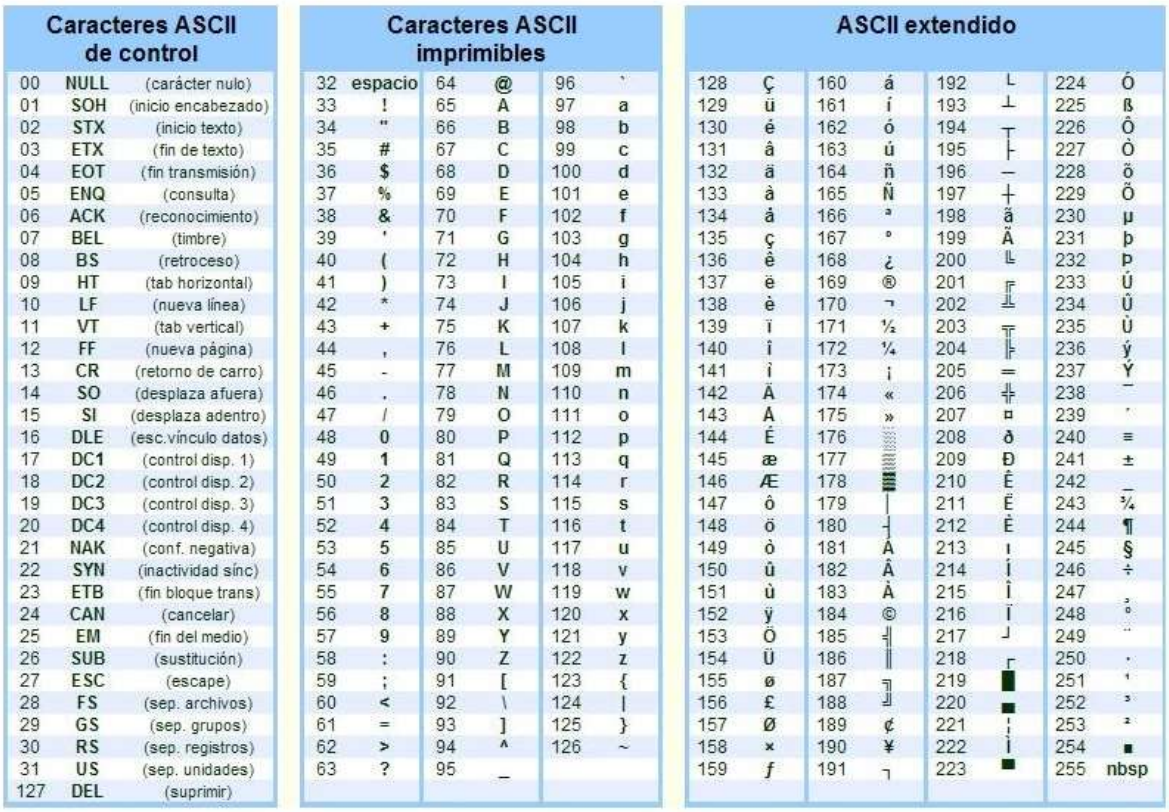
- Cargar (valor).

ER-> ASCII(ASCII)*((0..9)(0..9)*)



.	ASCII	(	0…9	)
q0	q1			
q1	q1	q2		
q2			q3	
q3			q3	q4

Tabla de símbolos (ASCII).



Caracteres ASCII de control		
00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles					
32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Error Léxico.

El error léxico aparece cuando el analizador léxico detecta que hay cadenas de caracteres que no encaja con ninguno de los patrones establecidos. Esto sirve para saber si se introdujo una palabra errónea que no esté dentro de la gramática del código.

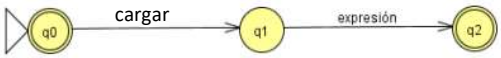
Análisis Sintáctico.

Descripción.

Esta es la segunda fase del compilador y se enfoca en revisar que la estructura del compilador sea la correcta.

Sintaxis de toda la gramática con los árboles sintácticos.

Instrucción.	Sintaxis.	Árbol sintáctico.

Cargar	qCargar (valor)	

Estructura correcta de un programa en el compilador.

Inicio

//Programa brazo robótico

Cargar (180) //el valor son los grados a los que girará, la posición inicial son 90°.

Fin

Análisis Semántico.

Descripción.

Esta es la tercera fase del compilador, la cual se encarga de revisar y verificar que la lógica de las instrucciones sea la correcta.

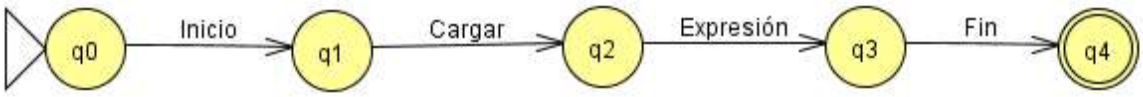
Tabla de reglas.

No.	Regla	Error y sugerencias al usuario	Tipo
1	La primera línea de un código es Inicio.	La primera línea no es un inicio, verifique.	R(p)
2	La última línea de un código es Fin.	La última línea no es un fin, verifique.	R(p)

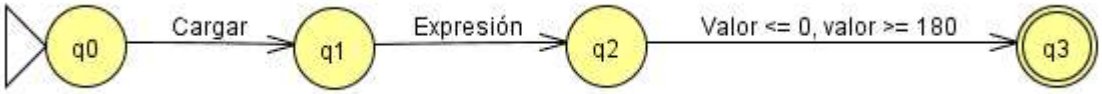
3	No se puede duplicar Inicio.	Hay dos o más inicio, solo puede existir uno, verifique.	R(p)
4	No se puede duplicar Fin.	Hay dos o más fin, solo puede existir uno verifique.	R(p)
5	No se puede colocar una expresión después de otra expresión.	Expresión doble solo debe de haber una, verifique.	R(p)
6	No se puede colocar una expresión sola.	Expresión sola, la expresión debe estar acompañada de una instrucción.	R(p)
7	El valor de una expresión debe ser numérico entre 1 y 255.	Valor fuera de rango, el valor debe de ser entre 1 y 255.	R(a)

Árboles semánticos.

Árbol semántico estructura de programa.



Árbol semántico de instrucción.



Traducción.

Descripción.

Esta es otra fase del compilador en la que se trabaja la traducción de las instrucciones del compilador a un lenguaje de programación en este caso para Arduino.

Tabla de traducción.

No.	Instrucción	Código

1	Inicio	#include <Servo.h>
		Servo servoMotor;
		Servo servoMotor2;
		int c = 0;
		void setup()
		{
		Serial.begin(9600);
		servoMotor.attach(13);

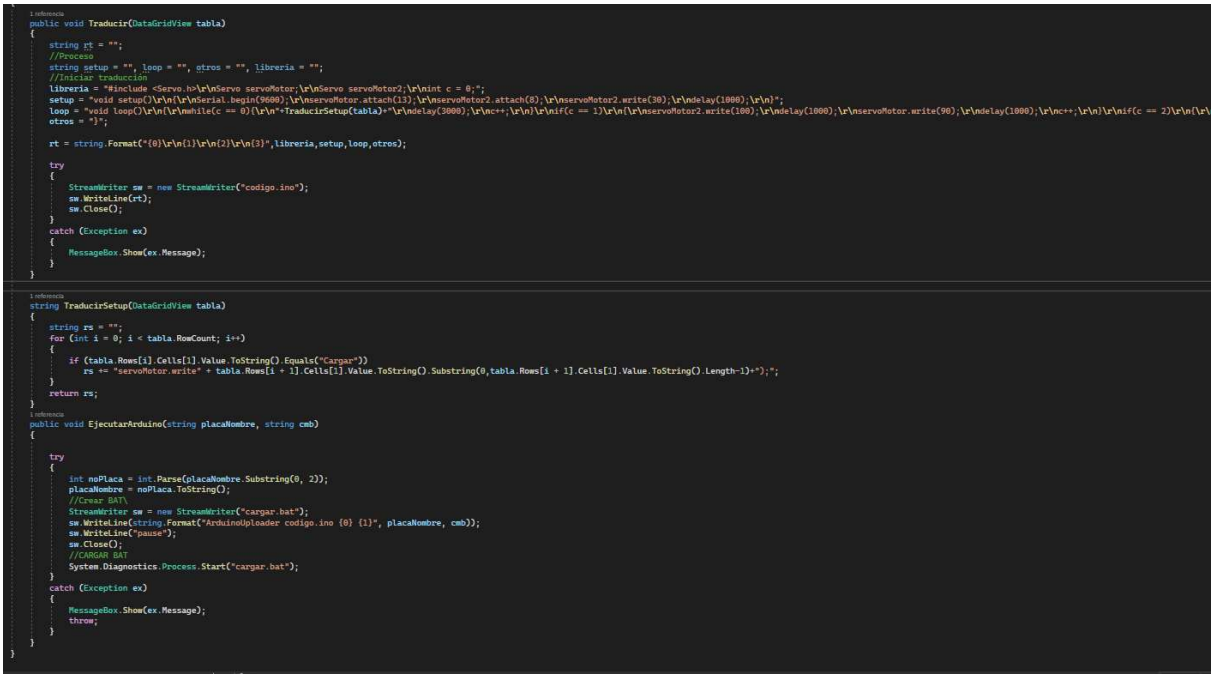
		servoMotor2.attach(8);
		servoMotor2.write(30);
		delay(1000);
		}

2	Fin	}

3	Cargar	void loop()
		{
		while(c == 0){
		servoMotor.write(0);
		delay(3000);
		c++;
		}
		if(c == 1)
		{
		servoMotor2.write(100);
		delay(1000);
		servoMotor.write(90);
		delay(1000);
		c++;

		}
		
		if(c == 2)
		
		{
		
		servoMotor2.write(30);
		
		delay(1000);
		
		}

4	Comentario	//Soy un comentario



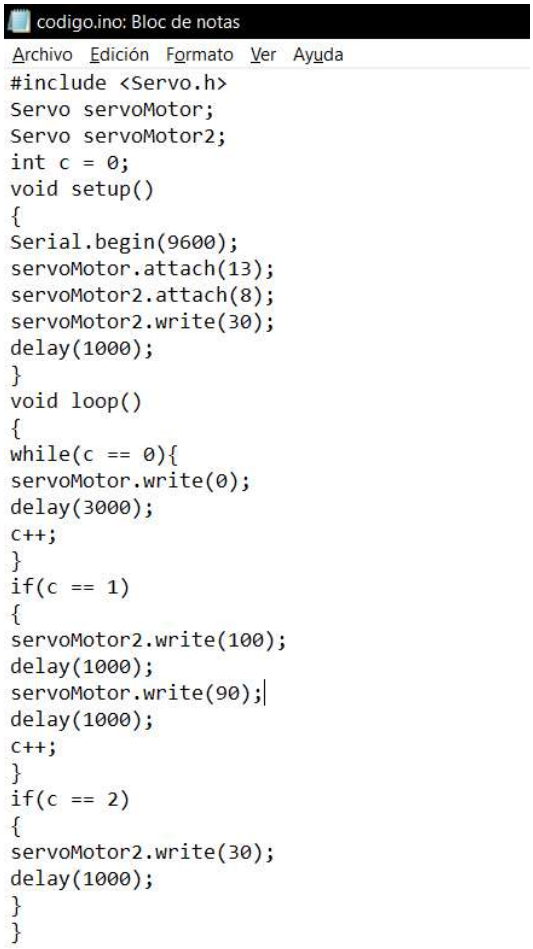
```
1 referencia
public void Traducir(DataGridView tabla)
{
    string rt = "";
    //Proceso
    string setup = "", loop = "", otros = "", libreria = "";
    //Iniciar traducción
    libreria = "#include <Servo.h>\r\n\nServo servoMotor;\r\nServo servoMotor2;\r\nint c = 0;";
    setup = "void setup()\r\n{\r\nSerial.begin(9600);\r\nservoMotor.attach(13);\r\nservoMotor2.attach(8);\r\nservoMotor2.write(30);\r\ndelay(1000);\r\n}";
    loop = "void loop()\r\n{\r\nwhile(c == 0){\r\n"+TraducirSetup(tabla)+"\r\ndelay(3000);\r\nc++;\r\n}\r\nif(c == 1)\r\n{\r\nservoMotor2.write(180);\r\ndelay(1000);\r\nservoMotor.write(90);\r\ndelay(1000);\r\nc++;\r\n}\r\nif(c == 2)\r\n{\r\
    otros = "}";

    rt = string.Format("{0}\r\n{1}\r\n{2}\r\n{3}",libreria,setup,loop,otros);

    try
    {
        StreamWriter sw = new StreamWriter("codigo.ino");
        sw.WriteLine(rt);
        sw.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

```
1 referencia
string TraducirSetup(DataGridView tabla)
{
    string rs = "";
    for (int i = 0; i < tabla.RowCount; i++)
    {
        if (tabla.Rows[i].Cells[1].Value.ToString().Equals("Cargar"))
            rs += "servoMotor.write" + tabla.Rows[i + 1].Cells[1].Value.ToString().Substring(0,tabla.Rows[i + 1].Cells[1].Value.ToString().Length-1)+");";
    }
    return rs;
}
1 referencia
public void EjecutarArduino(string placaNombre, string cmb)
{

    try
    {
        int noPlaca = int.Parse(placaNombre.Substring(0, 2));
        placaNombre = noPlaca.ToString();
        //Crear BAT\
        StreamWriter sw = new StreamWriter("cargar.bat");
        sw.WriteLine(string.Format("ArduinoUploader codigo.ino {0} {1}", placaNombre, cmb));
        sw.WriteLine("pause");
        sw.Close();
        //CARGAR BAT
        System.Diagnostics.Process.Start("cargar.bat");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        throw;
    }
}
}
```


codigo.ino: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
#include <Servo.h>
Servo servoMotor;
Servo servoMotor2;
int c = 0;
void setup()
{
Serial.begin(9600);
servoMotor.attach(13);
servoMotor2.attach(8);
servoMotor2.write(30);
delay(1000);
}
void loop()
{
while(c == 0){
servoMotor.write(0);
delay(3000);
c++;
}
if(c == 1)
{
servoMotor2.write(100);
delay(1000);
servoMotor.write(90);
delay(1000);
c++;
}
if(c == 2)
{
servoMotor2.write(30);
delay(1000);
}
}
```

Optimización.

Descripción.

Esta parte ya no es como tal parte del compilador, sirve para optimizar el código fuente para que todo tenga mejor rendimiento.

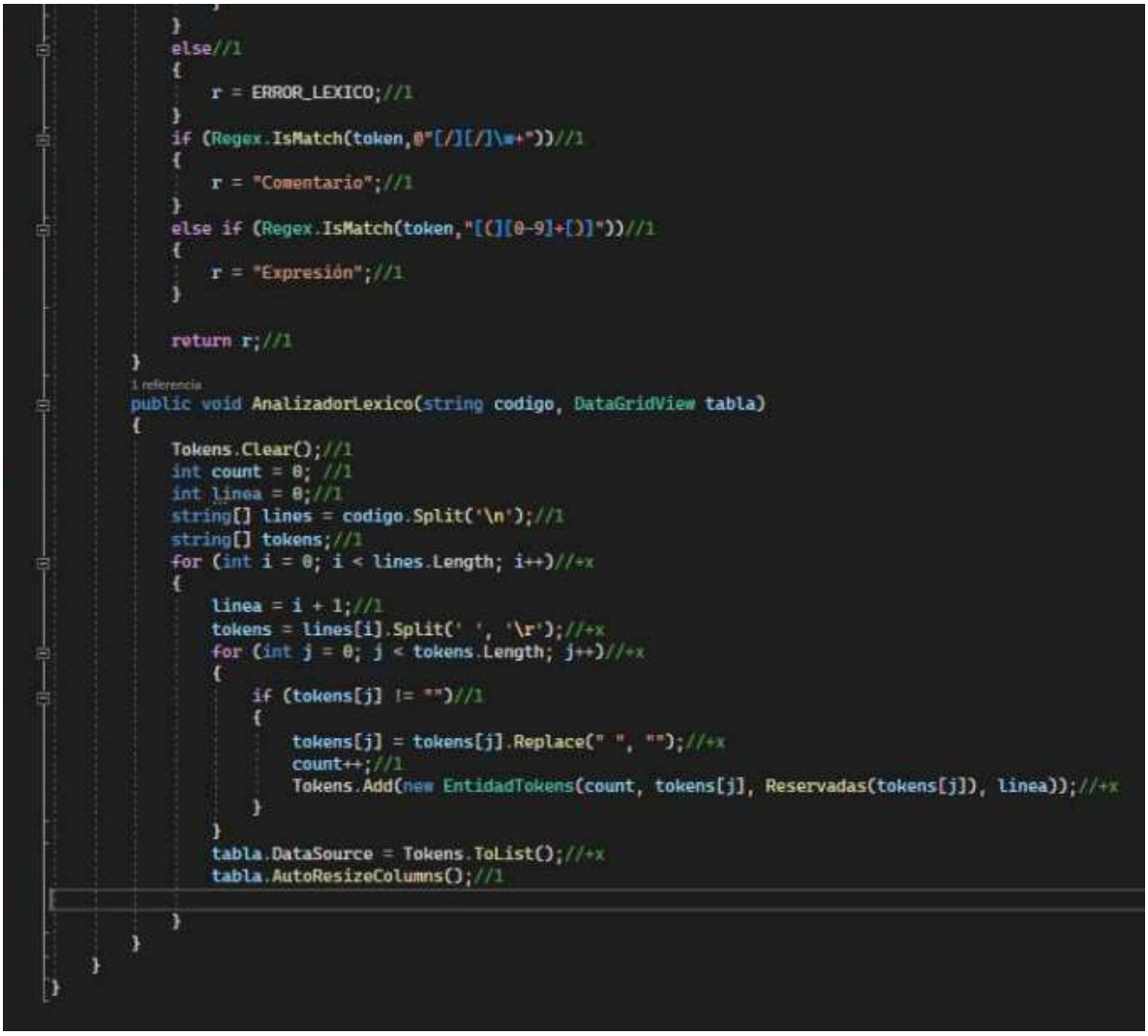
Notación Big O.

Optimización de Código (Léxico)

Ecuación sin optimizar $34n + x$

```
public class ManejadorLexico
{
    string[] reservadas = { "Inicio", "Fin", "Cargar", "Soltar", "Izquierda", "Derecha"};// 6
    public static List<EntidadTokens> Tokens = new List<EntidadTokens>(); // 4
    const string ERROR_LEXICO = "No identificado";//1
    1 referencia
    public string Reservadas(string token)
    {
        string r = "";//1
        bool validar = false;//1
        for (int i = 0; i < reservadas.Length; i++)//6
        {
            if (token.Equals(reservadas[i]))//1
            {
                validar = true;//1
                break;//1
            }
        }
        if (validar == true)//1
        {
            if (token.Equals("Inicio") || token.Equals("Fin")
                || token.Equals("Cargar") || token.Equals("Soltar")
                || token.Equals("Izquierda") || token.Equals("Derecha"))//1
            {
                r = "Palabra Reservada";//1
            }
        }
        else//1
        {
            r = ERROR_LEXICO;//1
        }
        if (Regex.IsMatch(token,@"[/][/]\w+"))//1
        {
            r = "Comentario";//1
        }
        else if (Regex.IsMatch(token,"[(][0-9]+[)]"))//1
        {
            r = "Expresión";//1
        }

        return r;//1
    }
    1 referencia
    public void AnalizadorLexico(string codigo, DataGridView tabla)
    {
        Tokens.Clear();//1
        int count = 0; //1
        int linea = 0;//1
        string[] lines = codigo.Split('\n');//1
        string[] tokens;//1
        for (int i = 0; i < lines.Length; i++)//+x
        {
            linea = i + 1;//1
            tokens = lines[i].Split(' ', '\r');//+x
            for (int i = 0; i < tokens.Length; i++)//+x
```



```
            }
            else//1
            {
                r = ERROR_LEXICO;//1
            }
            if (Regex.IsMatch(token,@"[/][/]\w+"))//1
            {
                r = "Comentario";//1
            }
            else if (Regex.IsMatch(token,"[(][0-9]+[)]"))//1
            {
                r = "Expresión";//1
            }

            return r;//1
        }
        1 referencia
        public void AnalizadorLexico(string codigo, DataGridView tabla)
        {
            Tokens.Clear();//1
            int count = 0; //1
            int linea = 0;//1
            string[] lines = codigo.Split('\n');//1
            string[] tokens;//1
            for (int i = 0; i < lines.Length; i++)//+x
            {
                linea = i + 1;//1
                tokens = lines[i].Split(' ', '\r');//+x
                for (int j = 0; j < tokens.Length; j++)//+x
                {
                    if (tokens[j] != "")//1
                    {
                        tokens[j] = tokens[j].Replace(" ", "");//+x
                        count++;//1
                        Tokens.Add(new EntidadTokens(count, tokens[j], Reservadas(tokens[j]), linea));//+x
                    }
                }
                tabla.DataSource = Tokens.ToList();//+x
                tabla.AutoResizeColumns();//1

            }
        }
    }
}
```

Ecuación con optimización $24n$

```
4 referencias
public class ManejadorLexico
{
    string[] reservadas = { "Inicio", "Fin", "Cargar", "Soltar", "Izquierda", "Derecha"};// 6
    public static List<EntidadTokens> Tokens = new List<EntidadTokens>(); // 4
    const string ERROR_LEXICO = "No identificado";//1
    2 referencias
    public string Reservadas(string token, int i)
    {
        string r = "";//1
        bool validar = false;//1
        //for (int i = 0; i < reservadas.Length; i++)//6
        if (i < reservadas.Length)
        {
            if (token.Equals(reservadas[i]))//1
            {
                validar = true;//1
                i = reservadas.Length;
                //break;//1
            }
            else
            {
                return Reservadas(token, i = i + 1);
            }
        }
            if (validar == true)//1
        {
            if (token.Equals("Inicio") || token.Equals("Fin")
                || token.Equals("Cargar") || token.Equals("Soltar")
                || token.Equals("Izquierda") || token.Equals("Derecha"))//1
            {
                r = "Palabra Reservada";//1
            }
        }
        else//1
        {
            r = ERROR_LEXICO;//1
        }
        if (Regex.IsMatch(token,@"[/][/]\w+"))//1
        {
            r = "Comentario";//1
        }
        else if (Regex.IsMatch(token,"[[][0-9]+[]]"))//1
        {
            r = "Expresión";//1
        }



        return r;//1
    }
    int count = 0; //1
```

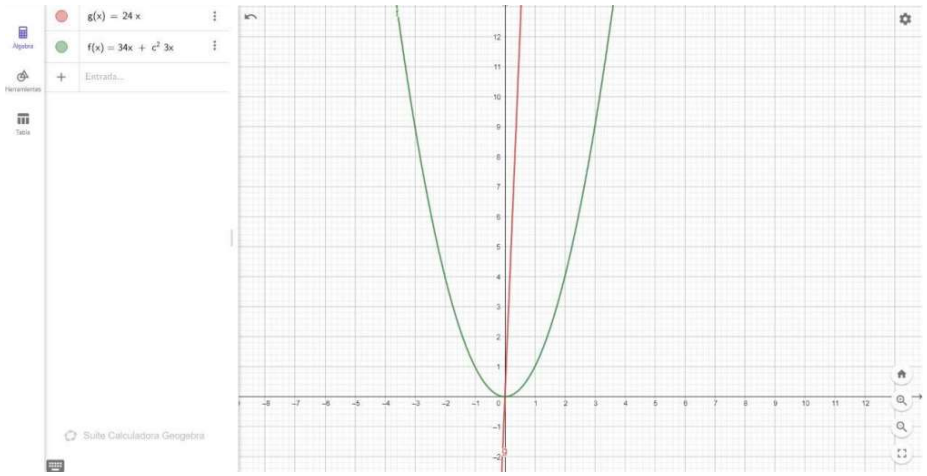
```
        int count = 0; //1
        1 referencia
        public void Analizar(string codigo, DataGridView tabla)
        {
            Tokens.Clear();//1

            int linea = 0;//1
            string[] lines = codigo.Split('\n');//1
            string[] tokens = null;//1
            AnalizadorLexico(codigo, linea, tabla, lines, tokens, 0);
            tabla.DataSource = Tokens.ToList();//+x
            tabla.AutoResizeColumns();//1
        }
        2 referencias
        public string AnalizadorLexico(string codigo, int linea, DataGridView tabla, string[] lines, string []tokens, int i)
        {


            //for (int i = 0; i < lines.Length; i++)//+x
            if( i < lines.Length)
            {
                linea = i + 1;//1
                tokens = lines[i].Split(' ', '\r');//+x
                AnalizadorLexico2(0, tokens, linea);
                return AnalizadorLexico(codigo, linea, tabla, lines, tokens, i = i + 1);
            }
            else
            {
                return "";
            }
        }
        2 referencias
        public string AnalizadorLexico2(int j, string[] tokens, int linea)
        {
            //for (int j = 0; j < tokens.Lth; j++)//+x
            if(j < tokens.Length)
            {
                if (tokens[j] != "")//1
                {
                    tokens[j] = tokens[j].Replace(" ", "");//+x
                    count++;//1
                    Tokens.Add(new EntidadTokens(count, tokens[j], Reservadas(tokens[j], 0), linea));//+x
                }
                return AnalizadorLexico2(j = j + 1, tokens, linea);
            }
            else
            {
                return "";
            }

        }
    }
}
```

Gráfica.



Conclusión.

Desarrollar un compilador es de gran ayuda para obtener una mejor visión de todo lo que hay detrás de un lenguaje de programación ya que hay muchas reglas, gramática, patrones, etc. Son pequeñas cosas en conjunto que se deben de considerar para generar un lenguaje de programación. Este proyecto enriquece la lógica que se tiene al desarrollar y permite una mejor comprensión de lo que se esté desarrollando.