

Implementación del analizador léxico

Estudiante	Escuela	Asignatura
Jerson Ernesto Chura Pacci jchurap@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Compiladores

Estudiante	Escuela	Asignatura
Andrea J. Ticona Mamani aticonam@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Compiladores

Estudiante	Escuela	Asignatura
Iben Omar Flores Polanco ifloresp@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Compiladores

Estudiante	Escuela	Asignatura
Joshua David Ortiz Rosas jortizr@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Compiladores

Informe	Tema	Duración
02	Implementación del analizador léxico	06 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2025 - I	22/03/25	25/03/25

Índice

1. Ejercicio	2
1.1. Palabras Reservadas	2
1.2. Tokens	2
1.3. Expresiones regulares	2
1.3.1. Expresion regular para controlar un error lexico	4
1.4. Almacenar tokens	4
2. Ejemplos	4
2.1. Hola mundo	4
2.2. Bucles anidados	6
2.3. Fibonacci recursivo	8
2.4. Error Lexico	9
3. URL de Repositorio Github	10

1. Ejercicio

- Implemente el analizador léxico para el lenguaje propuesto. Su programa deberá leer el código fuente de archivo en disco (debe proporcionar varios ejemplos) y luego deberá mostrar todos los tokens de manera similar al ejemplo mostrado en la Sección 5 (del archivo que el docente nos envió). Asegurese, de que los tokens retornados estén en una lista de objetos o diccionario.

1.1. Palabras Reservadas

- En esta sección se definen las palabras claves del lenguaje FusionCod.

Listing 1: Palabras reservadas

```
1 palabras_reservadas = {  
2     'fn': 'funcion', 'main': 'principal', 'show': 'imprimir', 'return': 'devolver', 'stop':  
3     'detener',  
4     'int': 'tentero', 'float': 'tflotante', 'text': 'tcadena', 'bool': 'tbooleano', 'void':  
5     'tvacio', 'if': 'si',  
6     '&&': 'y', 'and': 'y', '||': 'o', 'or': 'o', 'elif': 'sino', 'else': 'entonces', 'while':  
7     'mientras', 'for': 'para',  
8     'true': 'nbooleano', 'false': 'nbooleano', 'read': 'leer'  
9 }
```

1.2. Tokens

- En esta sección se definen los tokens de FusionCod.

Listing 2: Token

```
1 tokens = [  
2     'funcion', 'principal', 'pabierto', 'pcerrado', 'imprimir', 'comillas', 'id', 'coma',  
3     'fsentencia', 'devolver', 'detener', 'llaveabi', 'llavecerr', 'tentero',  
4     'tflotante', 'tbooleano', 'tcadena', 'tvacio', 'si', 'y', 'o', 'sino', 'entonces',  
5     'mientras', 'para', 'suma', 'resta', 'mul', 'div', 'residuo', 'menorque',  
6     'mayorque', 'menorigualque', 'mayorigualque', 'igual', 'igualbool',  
7     'diferentede', 'nentero', 'nflotante', 'ncadena', 'nbooleano', 'leer'  
8 ]
```

1.3. Expresiones regulares

- En esta sección se definen las expresiones regulares de FusionCod.
- Se ignora los comentarios, espacios y tabulaciones.
- Cada vez que el analizador encuentre una nueva línea, la función `t_newline` actualiza el número de línea del lexer y lo incrementa en la cantidad de nuevas líneas.

Listing 3: Expresiones Regulares

```
1 # Expresiones regulares de FusionCod.  
2 t_fsentencia = r';'  
3 t_pabierto = r'\(  
4 t_pcerrado = r'\)
```

```
5 t_llaveabi = r'\{'
6 t_llavecerr= r'\}'
7 t_suma = r'\+'
8 t_resta = r'\-'
9 t_mul = r'\*'
10 t_div = r'\/'
11 t_residuo= r'\%'
12 t_menorque = r'\<'
13 t_mayorque = r'\>'
14 t_menorigualque = r'\<='
15 t_mayorigualque = r'\>='
16 t_igual = r'\='
17 t_igualbool = r'\=='
18 t_diferentede = r'\<>'
19 t_coma = r','
20
21 # Ignorar espacios y tabulaciones.
22 t_ignore = ' \t'
23
24 # Expresion regular para cadenas de texto.
25 def t_ncadena(t):
26     r'"[^"]*"'
27     t.value = t.value[1:-1]
28     return t
29
30 # Expresion regular para identificadores.
31 def t_id(t):
32     r'[a-zA-Z_][a-zA-Z0-9_]*'
33     t.type = palabras_reservadas.get(t.value, 'id')
34     return t
35
36 # Expresion regular para numeros flotantes.
37 def t_nflotante(t):
38     r'-?\d+\.\d+'
39     t.value = float(t.value)
40     return t
41
42 # Expresion regular para numeros enteros.
43 def t_nentero(t):
44     r'-?\d+'
45     t.value = int(t.value)
46     return t
47
48 # Expresion regular para comentarios.
49 def t_comentario(t):
50     r'\#.*'
51     pass # Ignorar los comentarios.
52
53 # Manejo de saltos de lineas.
54 def t_newline(t):
55     r'\n+'
56     t.lexer.lineno += len(t.value)
```

1.3.1. Expresión regular para controlar un error léxico

- Cuando el analizador encuentre un carácter que no coincide con ninguno de los tokens definidos, guarda esa información sobre ellos en `lista_errores_lexicos` y omite el carácter para continuar el análisis.

Listing 4: Error léxico

```
1 def t_error(t):
2     columna = t.lexpos - t.lexer.lexdata.rfind('\n', 0, t.lexpos)
3     error = ErrorLexico(t.value[0], t.lineno, columna)
4     lista_errores_lexicos.append(error)
5     t.lexer.skip(1)
6
7 print(f"\nCódigo a compilar: {archivo}")
```

1.4. Almacenar tokens

- la función `generar_tokens` recorre el código fuente y extrae todos los tokens y los almacena en una lista.
- Se inicia un bucle hasta que no haya más tokens y sale del bucle.
- Se crea un objeto `Token` que contiene:
 - Tipo de token.
 - Valor (Lexema).
 - Número de línea.
 - Columna en la que se encuentra.
- Y finalmente devuelve la lista completa de los tokens analizados.

Listing 5: Generar tokens

```
1 def generar_tokens():
2     lista_de_tokens = []
3     while True:
4         tok = lexer.token()
5         if not tok: break
6         token_obj = Token(tok.type, tok.value, tok.lineno, tok.lexpos)
7         lista_de_tokens.append(token_obj)
8     return lista_de_tokens
```

2. Ejemplos

2.1. Hola mundo

Listing 6: Hola mundo

```
1 fn main () int {
2     show("Hola mundo");
3     return 0;
4 }
```

Código a compilar: holamundo.txt

Lista de Tokens:

Tipo	Valor	Línea	Columna
funcion	fn	1	0
principal	main	1	3
pabierto	(1	8
pcerrado)	1	9
tentero	int	1	11
llaveabi	{	1	15
imprimir	show	2	18
pabierto	(2	22
ncadena	Hola mundo	2	23
pcerrado)	2	35
fsentencia	;	2	36
devolver	return	3	42
nentero	0	3	49
fsentencia	;	3	50
llavecerr	}	4	52


Análisis léxico exitoso 

Figura 1: Imagen de Hola Mundo

2.2. Bucles anidados

```
1 fn generar_patron(n int) void {
2     i int = 1;
3     while (i <= n) {
4         j int = 1;
5         while (j <= i) {
6             if (j % 2 == 0) {
7                 show("* "); # Si j es par, muestra un asterisco
8             } elif (j % 3 == 0) {
9                 show("# "); # Si j es mltiplo de 3, muestra un numeral
10            } else {
11                show(j + " "); # De lo contrario, muestra el nmero
12            }
13            j = j + 1;
14        }
15        show("\n");
16        i = i + 1;
17    }
18 }
19
20 fn main() int {
21     num int;
22     show("Ingrese un nmero para generar el patrn:");
23     read(num);
24
25     if (num <= 0) {
26         show("El nmero debe ser mayor que 0.");
27     } else {
28         generar_patron(num);
29     }
30
31     return 0;
32 }
```

Código a compilar: buclesanidados.txt

Lista de Tokens:

Tipo	Valor	Línea	Columna
funcion	fn	1	0
id	generar_patron	1	3
pabierto	(1	17
id	n	1	18
tentero	int	1	20
pcerrado)	1	23
tvacio	void	1	25
llaveabi	{	1	30
id	i	2	36
tentero	int	2	38
igual	=	2	42
nentero	1	2	44
fsentencia	;	2	45
mientras	while	3	51
pabierto	(3	57
id	i	3	58
menorigualque	<=	3	60
id	n	3	63
pcerrado)	3	64
llaveabi	{	3	66
id	j	4	76
tentero	int	4	78
igual	=	4	82
nentero	1	4	84
fsentencia	;	4	85
mientras	while	5	95
pabierto	(5	101
id	j	5	102
menorigualque	<=	5	104
id	i	5	107
pcerrado)	5	108

Figura 2: Ejecución de Bucles Anidados

llaveabi	{	5	110
si	if	6	124
pabierto	(6	127
id	j	6	128
residuo	%	6	130
nentero	2	6	132
igualbool	==	6	134
nentero	0	6	137
pcerrado)	6	138
llaveabi	{	6	140
imprimir	show	7	158
pabierto	(7	162
ncadena	*	7	163
pcerrado)	7	167
fsentencia	;	7	168
llavecerr	}	8	219
sino	elif	8	221
pabierto	(8	226
id	j	8	227
residuo	%	8	229
nentero	3	8	231
igualbool	==	8	233
nentero	0	8	236
pcerrado)	8	237
llaveabi	{	8	239
imprimir	show	9	257
pabierto	(9	261
ncadena	#	9	262
pcerrado)	9	266
fsentencia	;	9	267
llavecerr	}	10	326
entonces	else	10	328
llaveabi	{	10	333
imprimir	show	11	351
pabierto	(11	355
id	j	11	356
suma	+	11	358
ncadena		11	360

Figura 3: Ejecución de Bucles Anidados

pcerrado)	11	363
fsentencia	;	11	364
llavecerr	}	12	415
id	j	13	429
igual	=	13	431
id	j	13	433
suma	+	13	435
nentero	1	13	437
fsentencia	;	13	438
llavecerr	}	14	448
imprimir	show	15	458
pabierto	(15	462
ncadena	\n	15	463
pcerrado)	15	467
fsentencia	;	15	468
id	i	16	478
igual	=	16	480
id	i	16	482
suma	+	16	484
nentero	1	16	486
fsentencia	;	16	487
llavecerr	}	17	493
llavecerr	}	18	495
funcion	fn	20	498
principal	main	20	501
pabierto	(20	505
pcerrado)	20	506
tentero	int	20	508
llaveabi	{	20	512
id	num	21	518
tentero	int	21	522
fsentencia	;	21	525
imprimir	show	22	531
pabierto	(22	535
ncadena	Ingrese un número para generar el patrón:	22	536
pcerrado)	22	579
fsentencia	;	22	580
leer	read	23	586

Figura 4: Ejecución de Bucles Anidados

pabierto	(23	590
id	num	23	591
pcerrado)	23	594
fsentencia	;	23	595
si	if	25	606
pabierto	(25	609
id	num	25	610
menorigualque	<=	25	614
nentero	0	25	617
pcerrado)	25	618
llaveabi	{	25	620
imprimir	show	26	630
pabierto	(26	634
ncadena	El número debe ser mayor que 0.	26	635
pcerrado)	26	668
fsentencia	;	26	669
llavecerr	}	27	675
entonces	else	27	677
llaveabi	{	27	682
id	generar_patron	28	692
pabierto	(28	706
id	num	28	707
pcerrado)	28	710
fsentencia	;	28	711
llavecerr	}	29	717
devolver	return	31	728
nentero	0	31	735
fsentencia	;	31	736
llavecerr	}	32	738

Análisis léxico exitoso ✓

Figura 5: Ejecución de Bucles Anidados

2.3. Fibonacci recursivo

```

1 fn fibonacci_recursivo (n int) int {
2   if (n <= 1) {
3     return n;
4   }

```



```

5   return fibonacci_recursivo(n - 1) + fibonacci_recursivo(n - 2);
6 }
7
8 fn main () int {
9     numero int = 5;
10    show("La secuencia fibonacci en " + numero + " es: " + fibonacci_recursivo(numero));
11    return 0;
12 }

```

Código a compilar: fibonaccirecursivo.txt			
Lista de Tokens:			
Tipo	Valor	Línea	Columna
funcion	fn	1	0
id	fibonacci_recursivo	1	3
pabierto	(1	23
id	n	1	24
tentero	int	1	26
pcerrado)	1	29
tentero	int	1	31
llaveabi	{	1	35
si	if	2	38
pabierto	(2	41
id	n	2	42
menorigualque	<=	2	44
nentero	1	2	47
pcerrado)	2	48
llaveabi	{	2	50
devolver	return	3	61
id	n	3	68
fsentencia	;	3	69
llavecerr	}	4	76
devolver	return	5	79
id	fibonacci_recursivo	5	86
pabierto	(5	105
id	n	5	106
resta	-	5	108
nentero	1	5	110
pcerrado)	5	111
suma	+	5	113
id	fibonacci_recursivo	5	115
pabierto	(5	134
id	n	5	135
resta	-	5	137
nentero	2	5	139
pcerrado)	5	140
fsentencia	;	5	141
llavecerr	}	6	143
funcion	fn	8	146
principal	main	8	149
pabierto	(8	154
pcerrado)	8	155
tentero	int	8	157
llaveabi	{	8	161
id	numero	9	164
tentero	int	9	171
igual	=	9	175
nentero	5	9	177
fsentencia	;	9	178
imprimir	show	10	181
pabierto	(10	185
ncadena	La secuencia fibonacci en	10	186
suma	+	10	216
id	numero	10	218
suma	+	10	225
ncadena	es:	10	227
suma	+	10	235
id	fibonacci_recursivo	10	237
pabierto	(10	256
id	numero	10	257
pcerrado)	10	263
pcerrado)	10	264
fsentencia	;	10	265
devolver	return	11	268
nentero	0	11	275
fsentencia	;	11	276
llavecerr	}	12	278

Figura 6: Ejecución de Fibonacci Recursivo

2.4. Error Lexico

- En este ejemplo se observa un error en la línea 5, donde nuestro lenguaje lo detecta mostrando una alerta, especificando cual es el caracter y en que línea y columna se encuentra, pero también nos muestra nuestra la tabla de tokens.

```

1 fn sumanumeros (n1 float,n2 float) float {
2     return n1+n2;
3 }
4
5 fn $main () int {
6     n1 float=126.2;
7     n2 float=267.4;
8     show(sumanumeros(n1,n2));
9     return 0;
10 }

```

Lista de Tokens:			
Tipo	Valor	Línea	Columna
funcion	fn	1	0
id	sumanumeros	1	3
pabierto	(1	15
id	n1	1	16
tflotante	float	1	19
coma	,	1	24
id	n2	1	25
tflotante	float	1	28
pcerrado)	1	33
tflotante	float	1	35
llaveabi	{	1	41
devolver	return	2	47
id	n1	2	54
suma	+	2	56
id	n2	2	57
fsentencia	;	2	59
llavecerr	}	3	61
funcion	fn	5	64
principal	main	5	68
pabierto	(5	73
pcerrado)	5	74
tentero	int	5	76
llaveabi	{	5	80
id	n1	6	86
tflotante	float	6	89
igual	=	6	94
nflotante	126.2	6	95
fsentencia	;	6	100
id	n2	7	106
tflotante	float	7	109
igual	=	7	114
nflotante	267.4	7	115
fsentencia	;	7	120
imprimir	show	8	123
pabierto	(8	127
id	sumanumeros	8	128
pabierto	(8	139
id	n1	8	140

coma	,	8	142
id	n2	8	143
pcerrado)	8	145
pcerrado)	8	146
fsentencia	;	8	147
devolver	return	9	153
nentero	0	9	160
fsentencia	;	9	161
llavecerr	}	10	163

XXX Análisis léxico fallido XXX
 Errores léxicos detectados:
 Carácter ilegal \$ en la línea 5, columna 4

Figura 7: Error lexico

3. URL de Repositorio Github

- URL del Repositorio GitHub:
- <https://github.com/JersonCh1/compiladores-25-I.git>