Implementación del analizador léxico

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

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

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

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

Informe	Tema	Duración
02	Implementación del analizador	06 horas
	léxico	

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

Semestre V Página 1 Compiladores

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 esten en una lista de objetos o diccionario.

1.1. Palabras Reservadas

• En esta seccion se definen las palabras claves del lenguaje FusionCod.

Listing 1: Palabras reservadas

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

1.2. Tokens

■ En esta seccion se definen los tokens de FusionCod.

Listing 2: Token

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

1.3. Expresiones regulares

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

Listing 3: Expresiones Regulares

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

```
t_{laveabi} = r' \setminus {'}
   t_llavecerr= r'\}'
   t_suma = r' + '
   t_resta = r' - 
   t_mul = r' \
10 t_div = r'/'
t_residuo= r'%'
t_menorque = r'<'
t_mayorque = r'>'
   t_menorigualque = r'<='
   t_mayorigualque = r'>='
   t_igual = r'='
   t_igualbool = r'=='
   t_diferentede = r'<>'
   t_{coma} = r',
20
   # Ignorar espacios y tabulaciones.
21
   t_ignore = ' \t'
22
23
   # Expresion regular para cadenas de texto.
   def t_ncadena(t):
25
      r'"[^"]*"'
26
       t.value = t.value[1:-1]
27
       return t
28
   # Expresion regular para identificadores.
   def t_id(t):
31
       r'[a-zA-Z_][a-zA-Z0-9_]*'
       t.type = palabras_reservadas.get(t.value, 'id')
       return t
34
35
   # Expresion regular para numeros flotantes.
   def t_nflotante(t):
      r'-?\d+\.\d+'
       t.value = float(t.value)
39
       return t
40
   # Expresion regular para numeros enteros.
   def t_nentero(t):
       r'-?\d+'
       t.value = int(t.value)
       return t
46
47
   # Expresion regular para comentarios.
48
   def t_comentario(t):
      r'\#.*'
50
       pass # Ignorar los comentarios.
   # Manejo de saltos de lineas.
53
   def t_newline(t):
54
       r'\n+'
55
       t.lexer.lineno += len(t.value)
```

1.3.1. Expresion regular para controlar un error lexico

 Cuando el analizador encuentre un caracter 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 analisis.

Listing 4: Error lexico

```
def t_error(t):
    columna = t.lexpos - t.lexer.lexdata.rfind('\n', 0, t.lexpos)
    error = ErrorLexico(t.value[0], t.lineno, columna)
    lista_errores_lexicos.append(error)
    t.lexer.skip(1)

print(f"\nCdigo a compilar: {archivo}")
```

1.4. Almacenar tokens

- la función generar_tokens recorre el codigo 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 linea.
 - Columna en la que se encuentra.
- Y finalmente devuelve la lista completa de los tokens analizados.

Listing 5: Generar tokens

```
def generar_tokens():
    lista_de_tokens = []
    while True:
        tok = lexer.token()
        if not tok: break
        token_obj = Token(tok.type, tok.value, tok.lineno, tok.lexpos)
        lista_de_tokens.append(token_obj)
    return lista_de_tokens
```

2. Ejemplos

2.1. Hola mundo

Listing 6: Hola mundo

```
fn main () int {
    show("Hola mundo");
    return 0;
}
```

Semestre V Página 4 Compiladores

Código a compilar: holamundo.txt

Lista de Tokens:

Тіро	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

aSalle

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

ista 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

llayeabi	{	5	110
si	if	6	124
pabierto	, I	6	127
id	i	6	128
residuo	j %	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
11avecerr	ĺ	8	219
sino	elíf	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

Semestre V Página 7 Compiladores

pcerrado)	11	363
fsentencia	į.	11	364
llavecerr	} j	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	j	22	580
leer	read	23	586

Figura 4: Ejecución de Bucles Anidados

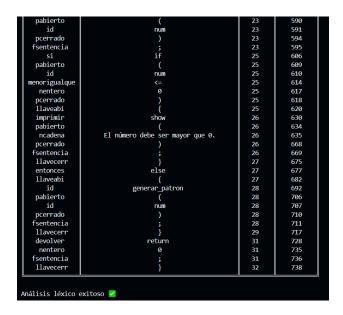


Figura 5: Ejecución de Bucles Anidados

2.3. Fibonacci recursivo

```
fn fibonacci_recursivo (n int) int {
   if (n <= 1) {
      return n;
   }
}</pre>
```

Semestre V Página 8 Compiladores

```
return fibonacci_recursivo(n - 1) + fibonacci_recursivo(n - 2);

fn main () int {
   numero int = 5;
   show("La secuencia finbonacci en " + numero + " es: " + fibonacci_recursivo(numero));
   return 0;
}
```

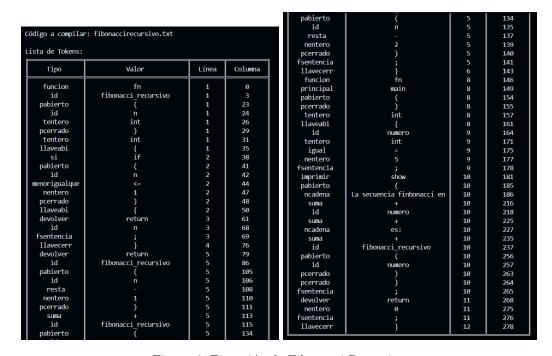


Figura 6: Ejecución de Fibonacci Recursivo

2.4. Error Lexico

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

```
fn sumanumeros (n1 float,n2 float) float {
    return n1+n2;
}

fn $main () int {
    n1 float=126.2;
    n2 float=267.4;
    show(sumanumeros(n1,n2));
    return 0;
}
```

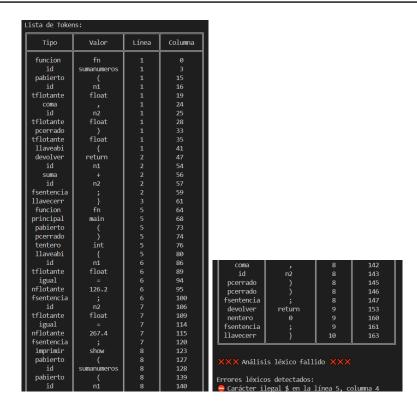


Figura 7: Error lexico

3. URL de Repositorio Github

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

Semestre V Página 10 Compiladores