Universidad Rafael Landivar
Facultad de Ingeniería
Compiladores, sección 01
Ing.Diana Gutierrez Cuellar

Fase #1 - Analizador Léxico (MiniLand)

Ana Paula Ortiz Hernandez - 1186624

Juan Palo Mazariegos Sepúlveda - 1140024

Guatemala 11 de febrero de 2026

I. INTRODUCCIÓN

El presente documento detalla el diseño del analizador léxico para el lenguaje MiniLand. Se implementa un escáner capaz de reconocer tokens, manejar indentación significativa y reportar errores léxicos.

Conjunto de Tokens de palabras reservadas.

Tabla No.1

Token	Lexema
INT	int
FLOAT	float
Tipo_String	string
BOOL	bool
IF	if
ELSE	else
WHILE	while
FUNCION	funcion
READ	read
RETURN	return
IMPRIME	imprime
TRUE	true
FALSE	false

Descripción: Esta tabla define el conjunto de palabras reservadas del lenguaje MiniLand. Su función principal es establecer la correspondencia entre un Lexema (la secuencia de caracteres escrita en el código fuente) y su Token respectivo (la categoría gramatical que el compilador reconoce).

Identificadores

ID Máximo de 31 caracteres

[a-zA-Z_][a-zA-Z0-9_]{0,30}

Literales

NUMENTERO- detecta los números enteros
NUMDECIMAL-detecta los números decimales
CADENASTRING-Detecta la cadena de texto entre comillas

Operadores

Token	Simbolo
SUMA	+
RESTA	-
MULTI	*
DIV	/
IGUAL	=
MAYORQ	>
MENORQ	<
MAYORIGU	>=
MENORIGU	<=
EQUIVA	==
NEGA	!=

Descripción:Esta tabla describe el nombre de los simbolos dce entrada al programa y les asigna su nombre respectivo.**s** que el analizador léxico de MiniLand debe reconocer para procesar la lógica y estructura del código

Símbolos especiales

token	Símbolo
PARENI	(
PAREND	)
COMA	,
DOSP	:

PUNCOM	;

Descripción: Esta tabla demuestra los tokens aceptados por el alfabeto.
El analizador léxico de MiniLand debe reconocer para procesar la lógica y estructura del código.

Tokens Estructurales

Toke	Simbolo
NEWLINE	Salto de linea
INDENT	aumento de indentacion
DEDENT	disminucion de identacion
EOF	$

Expresiones Regulares

Tipo	ER
Identificadores	[a-zA-Z_][a-zA-Z0-9_]{0,30}
Números Enteros:	[0-9]+
Números decimales	([0-9]+)\.([0-9]+)
Cadenas	“[^"\n]*”
Espacios y Tabs	[\t]+
Comentarios	#.*

II. GRAMÁTICA EN FORMA DE BACKUS–NAUR (BNF)

Para garantizar un análisis determinístico y evitar ambigüedades, se define la siguiente gramática:

Programa

<programa> ::= <lista_sentencias> EOF

Lista de sentencias

<lista_sentencias> ::= <sentencia> NEWLINE <lista_sentencias>
| ε

Sentencias

<sentencia> ::= <declaracion>
| <asignacion>
| <if_stmt>
| <while_stmt>
| <function_def>
| <return_stmt>
| <io_stmt>

Declaración

<declaracion> ::= <tipo> ID

<tipo> ::= "int"
| "float"
| "string"
| "bool"

Asignación

<asignacion> ::= ID "=" <expresion>

Expresiones

<expresion> ::= <termino> <expresion_tail>

```
<expresion_tail> ::= "+" <termino> <expresion_tail>
                  | "-" <termino> <expresion_tail>
                  | ε
```

Términos

```
<termino> ::= <factor> <termino_tail>

<termino_tail> ::= "*" <factor> <termino_tail>
                 | "/" <factor> <termino_tail>
                 | ε
```

Factor

```
<factor> ::= ID
           | NUMENTERO
           | NUMDECIMAL
           | CADENASTRING
           | "true"
           | "false"
           | "(" <expresion> ")"
           | <llamada_funcion>
```

Llamada a función

```
<llamada_funcion> ::= ID "(" <arg_list> ")"

<arg_list> ::= <expresion> <arg_tail>
             | ε

<arg_tail> ::= "," <expresion> <arg_tail>
             | ε
```

If

```
<if_stmt> ::= "if" <condicion> ":" NEWLINE
              INDENT <lista_sentencias> DEDENT <else_part>

<else_part> ::= "else" ":" NEWLINE
                INDENT <lista_sentencias> DEDENT
              | ε
```

While

<while_stmt> ::= "while" <condicion> ":" NEWLINE
INDENT <lista_sentencias> DEDENT

Condición

<condicion> ::= <expresion> <operador_relacional> <expresion>

<operador_relacional> ::= ">"
| "<"
| ">="
| "<="
| "=="
| "!="

Función

<function_def> ::= "function" ID "(" <param_list> ")" ":" NEWLINE
INDENT <lista_sentencias> DEDENT

Parámetros

<param_list> ::= ID <param_tail>
| ε

<param_tail> ::= "," ID <param_tail>
| ε

Return

<return_stmt> ::= "return" <expresion>

Entrada / Salida

<io_stmt> ::= "read" ID
| "write" <expresion>

IV. Manejo de Indentación

La indentación es gestionada por el analizador léxico mediante los tokens `INDENT` y `DEDENT`.

- Si el nivel de espacios aumenta → se genera `INDENT`

- Si disminuye → se generan uno o más `DEDENT`
- Se utiliza una pila (stack) para controlar los niveles
- Máximo recomendado: 5 niveles

Manejo de Errores Léxicos

El analizador léxico no se detiene ante errores.
Se detectan:

- Carácter inválido
- Cadena sin cerrar
- Número mal formado
- Identificador mayor a 31 caracteres
- Indentación inválida

Formato:

line <n>, col <m>: ERROR <descripción>