

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

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

Símbolos especiales

token	Símbolo
PAREN1	(
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 indentacion
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:

BNF Programa

`<programa> ::= <lista_sentencias> EOF`

BNF lista de Sentencias

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

BNF Sentencias

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

BNF Declaracion

`<declaracion> ::= <tipo> ID`

`<tipo> ::= int`

| float

| string

| bool

BNF Asigancion

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

BNF Expresiones

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

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

BNF Terminos

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

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

BNF Factor

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

BNF IF

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

BNF While

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

BNF Condicion

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

```
<operador_relacional> ::= ">"
```

```
| "<"
| ">="
| "<="
| "=="
```

```
| "!="
```

BNF Funcion

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

BNF Parametros

```
<param_list> ::= ID <param_tail>| ε
```

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

BNF Return

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

BNF Entrada/Salida

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

IV. MANEJO DE INDENTACIÓN

Así también se implementan la lectura de espacios en blanco o con espacio.

Se utiliza una pila (stack) para gestionar los tokens INDENT y DEDENT:

Al aumentar el nivel de espacios, se apila el valor y emite INDENT.

Al disminuir, se desapila hasta igualar el nivel y emite DEDENT.

Se limitó a un máximo de 5 niveles recomendado

Manejo de errores Léxicos

El analizador 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>