**UNIVERSIDAD DE GUADALAJARA**

**CENTRO UNIVERSITARIO DE TONALÁ**

**Logotipo

Descripción generada automáticamente**

**Materia: Traductores de Lenguaje.**

**Tema: Errores Léxicos**

**Integrantes:**

// Celestino Martinez Cristopher

// Ramirez Rivas Luis Fernando

// Rivas Tejeda Hector Manuel

**Fecha:** 03 de febrero del 2025

**Lenguaje asignado C**

#include stdio.h

 Celestino Martinez Cristopher

 Ramirez Rivas Luis Fernando

 Rivas Tejeda Hector Manuel

int main() {

   // Imprimir nombres completos en orden alfab�tico

    printf(TRADUCTORES DE LENGUAJESn);

    printf(Actividad 2n);

   // Caracter no reconocido (�)

    int x = 10 �;

   // Identificador mal formado (empieza con n�mero)

    int 3var = 5;

    //N�mero flotante incorrecto (dos puntos decimales)

    float pi = 3..14;

   // Cadena mal formada (falta comilla de cierre)

    printf(Hola mundo);

   // Comentario mal cerrado

     Esto es un comentario mal cerrado...

    //Uso de un car�cter especial no v�lido (@)

    char nombre@ = Cristopher;

    // Uso de un n�mero con una letra en medio

    int numero = 12a3;

   // Operador desconocido ($)

    int resultado = 5 $ 2;

    // Falta punto y coma al final de la declaraci�n

    int edad = 25

   // Uso incorrecto de secuencias de escape

    printf(Mensaje de error q);

   // Uso de comillas simples en una cadena de texto (debe ser comillas dobles)

    printf('Este es un mensaje incorrecto');

   // N�mero hexadecimal mal formado (falta '0x')

    int hex = x12AB;

    //Comentario con una sola barra inclinada (inv�lido)

    Esto no es un comentario v�lido

    return 0;

}

   // N�mero hexadecimal mal formado (falta '0x')

    int hex = x12AB;

    //Comentario con una sola barra inclinada (inv�lido)

    Esto no es un comentario v�lido

    return 0;

}

1. **Falta de los signos de mayor y menor en #include <stdio.h>**: El código debería incluir la librería estándar stdio.h usando los signos de mayor y menor (< y >).
2. **Cadena no entre comillas en printf**: Los argumentos de printf deben ir entre comillas dobles, pero el código usa texto sin comillas.
3. **Carácter no reconocido (�)**: El símbolo � es un carácter no válido en C, causando un error léxico.
4. **Identificador mal formado (empieza con número)**: En C, los identificadores (como 3var) no pueden comenzar con números.
5. **Número flotante incorrecto (dos puntos decimales)**: Los números flotantes deben tener solo un punto decimal, pero hay dos en 3..14.
6. **Cadena mal formada (falta comilla de cierre)**: Una cadena debe tener una comilla de cierre que falta en printf("Hola mundo).
7. **Comentario mal cerrado**: Un comentario de varias líneas no está cerrado correctamente, lo que genera un error.
8. **Uso de carácter especial no válido (@)**: El uso de un carácter especial como @ en un identificador no es permitido en C.
9. **Uso de un número con una letra en medio**: Los números no pueden contener letras, como en 12a3, que no es un valor válido.
10. **Operador desconocido ($)**: $ no es un operador reconocido en C, lo que genera un error.
11. **Falta el punto y coma**: En C, cada declaración debe terminar con un punto y coma, como en int edad = 25.
12. **Uso incorrecto de secuencias de escape**: En las cadenas, se están usando secuencias de escape incorrectas que no están entre comillas dobles.
13. **Uso de comillas simples en una cadena de texto**: En C, las cadenas deben ir entre comillas dobles, no simples.
14. **Número hexadecimal mal formado**: Los números hexadecimales deben comenzar con 0x, pero aquí se usa x12AB, lo que es incorrecto.
15. **Comentario con una sola barra inclinada**: Un comentario de una sola línea debe comenzar con //, pero se usa una barra inclinada simple, lo cual es incorrecto.

Estos errores son principalmente de sintaxis y de uso incorrecto de caracteres en el lenguaje C.

**Main**

from lexer import lexer

def main():

    # Leer el archivo de prueba

    with open('C:/Users/user/Desktop/traductores/Lex-main/tests/example.c', 'r') as f:

        code = f.read()

    # Alimentar el lexer

    lexer.input(code)

    # Mostrar tokens generados

    print("Tokens encontrados:")

    while True:

        tok = lexer.token()

        if not tok:

            break

        print(tok)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Explicación del Código**

1. **Importación del Lexer**:

python

Copy

from lexer import lexer

Aquí se importa un objeto lexer desde un módulo llamado lexer. Este lexer es probablemente una instancia de un analizador léxico generado por una herramienta como **PLY** (una implementación de Lex y Yacc en Python).

1. **Función main**:  
   La función main es el punto de entrada del programa. Aquí se realiza lo siguiente:
   * **Lectura del archivo de código fuente**:

with open('C:/Users/user/Desktop/traductores/Lex-main/tests/example.c', 'r') as f:

code = f.read()

Se abre el archivo example.c en modo lectura ('r') y se lee todo su contenido en la variable code.

* + **Alimentar el lexer**:

lexer.input(code)

El contenido del archivo (code) se pasa al lexer para que lo procese.

* + **Mostrar los tokens generados**:

print("Tokens encontrados:")

while True:

tok = lexer.token()

if not tok:

break

print(tok)

Se entra en un bucle infinito donde se llama repetidamente a lexer.token() para obtener los tokens generados por el lexer. Cuando no hay más tokens (tok es None), el bucle se detiene. Cada token se imprime en la consola.

1. **Ejecución del script**:

if \_\_name\_\_ == '\_\_main\_\_':

main()

Este bloque asegura que la función main se ejecute solo cuando el script se ejecuta directamente (no cuando se importa como módulo).

**¿Qué hace este código?**

Este script toma un archivo de código fuente en C (example.c) y lo procesa con un lexer para extraer los tokens. Los tokens son las unidades básicas del lenguaje, como palabras clave, identificadores, números, operadores, etc. El lexer identifica estos tokens según las reglas definidas en su implementación.

**Lexer**

import ply.lex as lex

# Lista de tokens

tokens = [

    'IDENTIFIER', 'INTEGER', 'FLOAT', 'STRING',  # Identificadores y literales

    'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'MODULO',  # Operadores aritméticos

    'LT', 'GT', 'LE', 'GE', 'EQ', 'NE',  # Operadores relacionales

    'ASSIGN',  # Operador de asignación

    'AND', 'OR', 'NOT',  # Operadores lógicos

    'LPAREN', 'RPAREN', 'LBRACE', 'RBRACE', 'LBRACKET', 'RBRACKET',  # Paréntesis, llaves y corchetes

    'COMMA', 'SEMICOLON', 'DOT',  # Otros símbolos

    'COMMENT\_SHARP', 'CHARACTER\_UNKNOWN', 'INVALID\_HEX'  # Otros errores específicos

]

# Palabras reservadas

reserved = {

    'if': 'IF', 'else': 'ELSE', 'while': 'WHILE', 'return': 'RETURN',

    'int': 'INT', 'float': 'FLOAT\_K', 'char': 'CHAR', 'void': 'VOID'

}

tokens += list(reserved.values())

# Expresiones regulares para tokens simples

t\_PLUS = r'\+'

t\_MINUS = r'-'

t\_TIMES = r'\\*'

t\_DIVIDE = r'/'

t\_MODULO = r'%'

t\_ASSIGN = r'='

t\_LT = r'<'

t\_GT = r'>'

t\_LE = r'<='

t\_GE = r'>='

t\_EQ = r'=='

t\_NE = r'!='

t\_AND = r'&&'

t\_OR = r'\|\|'

t\_NOT = r'!'

t\_LPAREN = r'\('

t\_RPAREN = r'\)'

t\_LBRACE = r'\{'

t\_RBRACE = r'\}'

t\_LBRACKET = r'\['

t\_RBRACKET = r'\]'

t\_COMMA = r','

t\_SEMICOLON = r';'

t\_DOT = r'\.'

# Manejo de caracteres no reconocidos

def t\_CHARACTER\_UNKNOWN(t):

    r'[^\w\s\+\-\\*/\(\)\[\]\{\},;=<>!\&\|\%\']'

    print(f"Error léxico: Carácter no reconocido '{t.value[0]}' en línea {t.lexer.lineno}")

    t.lexer.skip(1)  # Saltar el carácter no reconocido

# Manejo de identificadores mal formados

def t\_IDENTIFIER(t):

    r'[a-zA-Z\_][a-zA-Z0-9\_]\*'

    if t.value[0].isdigit():

        print(f"Error léxico: Identificador mal formado '{t.value}' en línea {t.lexer.lineno}")

        t.value = ""  # Limpiar el valor del token

    else:

        t.type = reserved.get(t.value, 'IDENTIFIER')

    return t

# Manejo de números flotantes incorrectos

def t\_FLOAT(t):

    r'\d+\.\.\d+'

    print(f"Error léxico: Número flotante incorrecto '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de cadenas mal formadas

def t\_STRING(t):

    r'"([^"\\]|\\.)\*"'

    if t.value.count('"') % 2 == 1:  # Verifica si hay una comilla de cierre

        print(f"Error léxico: Cadena mal formada en línea {t.lexer.lineno}")

        t.value = ""  # Limpiar el valor del token

    else:

        t.value = t.value[1:-1]  # Remover comillas

    return t

# Manejo de comentarios mal cerrados

def t\_COMMENT\_SHARP(t):

    r'\#.\*'

    pass

def t\_COMMENT(t):

    r'//.\*|/\\*[\s\S]\*?\\*/'

    if t.value.startswith('/\*') and not t.value.endswith('\*/'):

        print(f"Error léxico: Comentario mal cerrado en línea {t.lexer.lineno}")

    pass

# Manejo de caracteres especiales no válidos (@)

def t\_INVALID\_CHAR(t):

    r'@'

    print(f"Error léxico: Uso de un carácter especial no válido '@' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de números con letras en medio

def t\_INVALID\_NUMBER(t):

    r'\d+[a-zA-Z]\d+'

    print(f"Error léxico: Uso de un número con una letra en medio '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de operadores desconocidos ($)

def t\_UNKNOWN\_OPERATOR(t):

    r'\$'

    print(f"Error léxico: Operador desconocido '$' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de punto y coma faltante

def t\_SEMICOLON\_MISSING(t):

    r'(?<=\w)\s\*$'

    print(f"Error léxico: Falta punto y coma al final de la declaración en línea {t.lexer.lineno}")

    return t

# Manejo de uso incorrecto de secuencias de escape

def t\_INVALID\_ESCAPE\_SEQUENCE(t):

    r'\\[^abfnrtv\\\'"]'

    print(f"Error léxico: Uso incorrecto de secuencias de escape en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de comillas simples en lugar de dobles en cadenas

def t\_SINGLE\_QUOTE\_IN\_STRING(t):

    r"'([^'\\]|\\.)\*'"

    print(f"Error léxico: Uso de comillas simples en lugar de dobles en cadena de texto en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de números hexadecimales mal formados

def t\_INVALID\_HEX(t):

    r'\b[^0x0-9A-Fa-f]\d+[A-Fa-f0-9]{2,}\b'

    print(f"Error léxico: Número hexadecimal mal formado '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de comentarios con una sola barra (inválido)

def t\_INVALID\_SINGLE\_SLASH\_COMMENT(t):

    r'[^/]/.\*'

    print(f"Error léxico: Comentario con una sola barra inclinada (inválido) en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Ignorar espacios y tabulaciones

t\_ignore = ' \t'

# Manejo de nuevas líneas

def t\_newline(t):

    r'\n+'

    t.lexer.lineno += len(t.value)

# Manejo de errores mejorado

def t\_error(t):

    print(f"Error léxico: Carácter no reconocido '{t.value[0]}' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Construir el lexer

lexer = lex.lex()

tokens += list(reserved.values())

# Expresiones regulares para tokens simples

t\_PLUS = r'\+'

t\_MINUS = r'-'

t\_TIMES = r'\\*'

t\_DIVIDE = r'/'

t\_MODULO = r'%'

t\_ASSIGN = r'='

t\_LT = r'<'

t\_GT = r'>'

t\_LE = r'<='

t\_GE = r'>='

t\_EQ = r'=='

t\_NE = r'!='

t\_AND = r'&&'

t\_OR = r'\|\|'

t\_NOT = r'!'

t\_LPAREN = r'\('

t\_RPAREN = r'\)'

t\_LBRACE = r'\{'

t\_RBRACE = r'\}'

t\_LBRACKET = r'\['

t\_RBRACKET = r'\]'

t\_COMMA = r','

t\_SEMICOLON = r';'

t\_DOT = r'\.'

# Manejo de caracteres no reconocidos

def t\_CHARACTER\_UNKNOWN(t):

    r'[^\w\s\+\-\\*/\(\)\[\]\{\},;=<>!\&\|\%\']'

    print(f"Error léxico: Carácter no reconocido '{t.value[0]}' en línea {t.lexer.lineno}")

    t.lexer.skip(1)  # Saltar el carácter no reconocido

# Manejo de identificadores mal formados

def t\_IDENTIFIER(t):

    r'[a-zA-Z\_][a-zA-Z0-9\_]\*'

    if t.value[0].isdigit():

        print(f"Error léxico: Identificador mal formado '{t.value}' en línea {t.lexer.lineno}")

        t.value = ""  # Limpiar el valor del token

    else:

        t.type = reserved.get(t.value, 'IDENTIFIER')

    return t

# Manejo de números flotantes incorrectos

def t\_FLOAT(t):

    r'\d+\.\.\d+'

    print(f"Error léxico: Número flotante incorrecto '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de cadenas mal formadas

def t\_STRING(t):

    r'"([^"\\]|\\.)\*"'

    if t.value.count('"') % 2 == 1:  # Verifica si hay una comilla de cierre

        print(f"Error léxico: Cadena mal formada en línea {t.lexer.lineno}")

        t.value = ""  # Limpiar el valor del token

    else:

        t.value = t.value[1:-1]  # Remover comillas

    return t

# Manejo de comentarios mal cerrados

def t\_COMMENT\_SHARP(t):

    r'\#.\*'

    pass

def t\_COMMENT(t):

    r'//.\*|/\\*[\s\S]\*?\\*/'

    if t.value.startswith('/\*') and not t.value.endswith('\*/'):

        print(f"Error léxico: Comentario mal cerrado en línea {t.lexer.lineno}")

    pass

# Manejo de caracteres especiales no válidos (@)

def t\_INVALID\_CHAR(t):

    r'@'

    print(f"Error léxico: Uso de un carácter especial no válido '@' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de números con letras en medio

def t\_INVALID\_NUMBER(t):

    r'\d+[a-zA-Z]\d+'

    print(f"Error léxico: Uso de un número con una letra en medio '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de operadores desconocidos ($)

def t\_UNKNOWN\_OPERATOR(t):

    r'\$'

    print(f"Error léxico: Operador desconocido '$' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de punto y coma faltante

def t\_SEMICOLON\_MISSING(t):

    r'(?<=\w)\s\*$'

    print(f"Error léxico: Falta punto y coma al final de la declaración en línea {t.lexer.lineno}")

    return t

# Manejo de uso incorrecto de secuencias de escape

def t\_INVALID\_ESCAPE\_SEQUENCE(t):

    r'\\[^abfnrtv\\\'"]'

    print(f"Error léxico: Uso incorrecto de secuencias de escape en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de comillas simples en lugar de dobles en cadenas

def t\_SINGLE\_QUOTE\_IN\_STRING(t):

    r"'([^'\\]|\\.)\*'"

    print(f"Error léxico: Uso de comillas simples en lugar de dobles en cadena de texto en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de números hexadecimales mal formados

def t\_INVALID\_HEX(t):

    r'\b[^0x0-9A-Fa-f]\d+[A-Fa-f0-9]{2,}\b'

    print(f"Error léxico: Número hexadecimal mal formado '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de comentarios con una sola barra (inválido)

def t\_INVALID\_SINGLE\_SLASH\_COMMENT(t):

    r'[^/]/.\*'

    print(f"Error léxico: Comentario con una sola barra inclinada (inválido) en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Ignorar espacios y tabulaciones

t\_ignore = ' \t'

# Manejo de nuevas líneas

def t\_newline(t):

    r'\n+'

    t.lexer.lineno += len(t.value)

# Manejo de errores mejorado

def t\_error(t):

    print(f"Error léxico: Carácter no reconocido '{t.value[0]}' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Construir el lexer

lexer = lex.lex()

    return t

# Manejo de números flotantes incorrectos

def t\_FLOAT(t):

    r'\d+\.\.\d+'

    print(f"Error léxico: Número flotante incorrecto '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de cadenas mal formadas

def t\_STRING(t):

    r'"([^"\\]|\\.)\*"'

    if t.value.count('"') % 2 == 1:  # Verifica si hay una comilla de cierre

        print(f"Error léxico: Cadena mal formada en línea {t.lexer.lineno}")

        t.value = ""  # Limpiar el valor del token

    else:

        t.value = t.value[1:-1]  # Remover comillas

    return t

# Manejo de comentarios mal cerrados

def t\_COMMENT\_SHARP(t):

    r'\#.\*'

    pass

def t\_COMMENT(t):

    r'//.\*|/\\*[\s\S]\*?\\*/'

    if t.value.startswith('/\*') and not t.value.endswith('\*/'):

        print(f"Error léxico: Comentario mal cerrado en línea {t.lexer.lineno}")

    pass

# Manejo de caracteres especiales no válidos (@)

def t\_INVALID\_CHAR(t):

    r'@'

    print(f"Error léxico: Uso de un carácter especial no válido '@' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de números con letras en medio

def t\_INVALID\_NUMBER(t):

    r'\d+[a-zA-Z]\d+'

    print(f"Error léxico: Uso de un número con una letra en medio '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de operadores desconocidos ($)

def t\_UNKNOWN\_OPERATOR(t):

    r'\$'

    print(f"Error léxico: Operador desconocido '$' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de punto y coma faltante

def t\_SEMICOLON\_MISSING(t):

    r'(?<=\w)\s\*$'

    print(f"Error léxico: Falta punto y coma al final de la declaración en línea {t.lexer.lineno}")

    return t

# Manejo de uso incorrecto de secuencias de escape

def t\_INVALID\_ESCAPE\_SEQUENCE(t):

    r'\\[^abfnrtv\\\'"]'

    print(f"Error léxico: Uso incorrecto de secuencias de escape en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de comillas simples en lugar de dobles en cadenas

def t\_SINGLE\_QUOTE\_IN\_STRING(t):

    r"'([^'\\]|\\.)\*'"

    print(f"Error léxico: Uso de comillas simples en lugar de dobles en cadena de texto en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de números hexadecimales mal formados

def t\_INVALID\_HEX(t):

    r'\b[^0x0-9A-Fa-f]\d+[A-Fa-f0-9]{2,}\b'

    print(f"Error léxico: Número hexadecimal mal formado '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de comentarios con una sola barra (inválido)

def t\_INVALID\_SINGLE\_SLASH\_COMMENT(t):

    r'[^/]/.\*'

    print(f"Error léxico: Comentario con una sola barra inclinada (inválido) en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Ignorar espacios y tabulaciones

t\_ignore = ' \t'

# Manejo de nuevas líneas

def t\_newline(t):

    r'\n+'

    t.lexer.lineno += len(t.value)

# Manejo de errores mejorado

def t\_error(t):

    print(f"Error léxico: Carácter no reconocido '{t.value[0]}' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Construir el lexer

lexer = lex.lex()

ilizando la biblioteca **PLY** (Python Lex-Yacc). Este lexer está diseñado para procesar código fuente en el lenguaje **C** y reconocer los tokens básicos del lenguaje, como palabras clave, identificadores, números, operadores, símbolos, etc. A continuación, te explico en detalle cómo funciona este lexer.

**Estructura del Lexer**

# Manejo de operadores desconocidos ($)

def t\_UNKNOWN\_OPERATOR(t):

    r'\$'

    print(f"Error léxico: Operador desconocido '$' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de punto y coma faltante

def t\_SEMICOLON\_MISSING(t):

    r'(?<=\w)\s\*$'

    print(f"Error léxico: Falta punto y coma al final de la declaración en línea {t.lexer.lineno}")

    return t

# Manejo de uso incorrecto de secuencias de escape

def t\_INVALID\_ESCAPE\_SEQUENCE(t):

    r'\\[^abfnrtv\\\'"]'

    print(f"Error léxico: Uso incorrecto de secuencias de escape en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de comillas simples en lugar de dobles en cadenas

def t\_SINGLE\_QUOTE\_IN\_STRING(t):

    r"'([^'\\]|\\.)\*'"

    print(f"Error léxico: Uso de comillas simples en lugar de dobles en cadena de texto en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Manejo de números hexadecimales mal formados

def t\_INVALID\_HEX(t):

    r'\b[^0x0-9A-Fa-f]\d+[A-Fa-f0-9]{2,}\b'

    print(f"Error léxico: Número hexadecimal mal formado '{t.value}' en línea {t.lexer.lineno}")

    t.value = ""  # Limpiar el valor del token

    return t

# Manejo de comentarios con una sola barra (inválido)

def t\_INVALID\_SINGLE\_SLASH\_COMMENT(t):

    r'[^/]/.\*'

    print(f"Error léxico: Comentario con una sola barra inclinada (inválido) en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Ignorar espacios y tabulaciones

t\_ignore = ' \t'

# Manejo de nuevas líneas

def t\_newline(t):

    r'\n+'

    t.lexer.lineno += len(t.value)

# Manejo de errores mejorado

def t\_error(t):

    print(f"Error léxico: Carácter no reconocido '{t.value[0]}' en línea {t.lexer.lineno}")

    t.lexer.skip(1)

# Construir el lexer

lexer = lex.lex()

1. **t\_CHARACTER\_UNKNOWN**: Detecta cualquier carácter no reconocido.
2. **t\_IDENTIFIER**: Detecta identificadores mal formados (como los que empiezan con un número).
3. **t\_FLOAT**: Detecta números flotantes con un error en la cantidad de puntos decimales (3..14).
4. **t\_STRING**: Detecta cadenas mal formadas, donde falta la comilla de cierre.
5. **t\_COMMENT\_SHARP y t\_COMMENT**: Maneja comentarios de una sola línea y varios líneas.
6. **t\_INVALID\_CHAR**: Detecta el uso de caracteres especiales no válidos como @.
7. **t\_INVALID\_NUMBER**: Detecta números con letras intercaladas.
8. **t\_UNKNOWN\_OPERATOR**: Detecta operadores desconocidos como $.
9. **t\_SEMICOLON\_MISSING**: Detecta cuando falta un punto y coma al final de una declaración.
10. **t\_INVALID\_ESCAPE\_SEQUENCE**: Detecta secuencias de escape no válidas.
11. **t\_SINGLE\_QUOTE\_IN\_STRING**: Detecta el uso incorrecto de comillas simples en cadenas de texto.
12. **t\_INVALID\_HEX**: Detecta números hexadecimales mal formados.
13. **t\_INVALID\_SINGLE\_SLASH\_COMMENT**: Detecta comentarios con una sola barra.

**Prueba:**

Este lexer te mostrará los errores léxicos específicos para los casos mencionados en el código que proporcionaste.

**Salida**Tokens encontrados:

Error léxico: Carácter no reconocido '#' en línea 1

LexToken(IDENTIFIER,'nclude',1,2)

LexToken(LT,'<',1,9)

LexToken(IDENTIFIER,'stdio',1,10)

Error léxico: Carácter no reconocido '.' en línea 1

LexToken(GT,'>',1,17)

Error léxico: Comentario con una sola barra inclinada (inválido) en línea 3

LexToken(INT,'int',5,116)

LexToken(IDENTIFIER,'main',5,120)

LexToken(LPAREN,'(',5,124)

LexToken(RPAREN,')',5,125)

LexToken(LBRACE,'{',5,127)

LexToken(IDENTIFIER,'printf',7,188)

LexToken(LPAREN,'(',7,194)

Error léxico: Carácter no reconocido '"' en línea 7

LexToken(IDENTIFIER,'RADUCTORES',7,197)

LexToken(IDENTIFIER,'DE',7,208)

LexToken(IDENTIFIER,'LENGUAJES',7,211)

Error léxico: Carácter no reconocido ':' en línea 7

LexToken(IDENTIFIER,'n',7,222)

Error léxico: Carácter no reconocido '"' en línea 7

LexToken(SEMICOLON,';',7,225)

LexToken(IDENTIFIER,'printf',8,231)

LexToken(LPAREN,'(',8,237)

Error léxico: Carácter no reconocido '"' en línea 8

LexToken(IDENTIFIER,'ctividad',8,240)

Error léxico: Carácter no reconocido '2' en línea 8

Error léxico: Carácter no reconocido '\' en línea 8

Error léxico: Carácter no reconocido '"' en línea 8

LexToken(SEMICOLON,';',8,254)

LexToken(INT,'int',11,285)

LexToken(IDENTIFIER,'x',11,289)

LexToken(ASSIGN,'=',11,291)

Error léxico: Carácter no reconocido '1' en línea 11

Error léxico: Carácter no reconocido '0' en línea 11

LexToken(SEMICOLON,';',11,295)

LexToken(FLOAT\_K,'float',12,301)

LexToken(IDENTIFIER,'y',12,307)

LexToken(ASSIGN,'=',12,309)

Error léxico: Carácter no reconocido '3' en línea 12

Error léxico: Carácter no reconocido '.' en línea 12

Error léxico: Carácter no reconocido '4' en línea 12

LexToken(SEMICOLON,';',12,315)

LexToken(IF,'if',13,321)

LexToken(LPAREN,'(',13,324)

LexToken(IDENTIFIER,'x',13,325)

LexToken(GT,'>',13,327)

LexToken(IDENTIFIER,'y',13,329)

LexToken(RPAREN,')',13,330)

LexToken(LBRACE,'{',13,332)

LexToken(IDENTIFIER,'printf',14,342)

LexToken(LPAREN,'(',14,348)

Error léxico: Carácter no reconocido '"' en línea 14

LexToken(IDENTIFIER,'ola',14,351)

LexToken(COMMA,',',14,354)

LexToken(IDENTIFIER,'mundo',14,356)

LexToken(NOT,'!',14,361)

Error léxico: Carácter no reconocido '\' en línea 14

Error léxico: Carácter no reconocido '"' en línea 14

LexToken(SEMICOLON,';',14,366)

LexToken(RBRACE,'}',15,372)

LexToken(IDENTIFIER,'printf',18,443)

LexToken(LPAREN,'(',18,449)

Error léxico: Carácter no reconocido '"' en línea 18

LexToken(IDENTIFIER,'nIntegrantes',18,452)

LexToken(IDENTIFIER,'del',18,465)

LexToken(IDENTIFIER,'equipo',18,469)

Error léxico: Carácter no reconocido ':' en línea 18

LexToken(IDENTIFIER,'n',18,477)

Error léxico: Carácter no reconocido '"' en línea 18

LexToken(SEMICOLON,';',18,480)

LexToken(IDENTIFIER,'printf',19,486)

LexToken(LPAREN,'(',19,492)

Error léxico: Carácter no reconocido '"' en línea 19

Error léxico: Carácter no reconocido '.' en línea 19

LexToken(IDENTIFIER,'Celestino',19,497)

LexToken(IDENTIFIER,'Martinez',19,507)

LexToken(IDENTIFIER,'Cristopher',19,516)

Error léxico: Carácter no reconocido '\' en línea 19

Error léxico: Carácter no reconocido '"' en línea 19

LexToken(SEMICOLON,';',19,530)

LexToken(IDENTIFIER,'printf',20,536)

LexToken(LPAREN,'(',20,542)

Error léxico: Carácter no reconocido '"' en línea 20

Error léxico: Carácter no reconocido '.' en línea 20

LexToken(IDENTIFIER,'Ramirez',20,547)

LexToken(IDENTIFIER,'Rivas',20,555)

LexToken(IDENTIFIER,'Luis',20,561)

LexToken(IDENTIFIER,'Fernando',20,566)

Error léxico: Carácter no reconocido '\' en línea 20

Error léxico: Carácter no reconocido '"' en línea 20

LexToken(SEMICOLON,';',20,578)

LexToken(IDENTIFIER,'printf',21,584)

LexToken(LPAREN,'(',21,590)

Error léxico: Carácter no reconocido '"' en línea 21

Error léxico: Carácter no reconocido '.' en línea 21

LexToken(IDENTIFIER,'Rivas',21,595)

LexToken(IDENTIFIER,'Tejeda',21,601)

LexToken(IDENTIFIER,'Hector',21,608)

LexToken(IDENTIFIER,'Manuel',21,615)

Error léxico: Carácter no reconocido '\' en línea 21

Error léxico: Carácter no reconocido '"' en línea 21

LexToken(SEMICOLON,';',21,625)

LexToken(RETURN,'return',23,632)

Error léxico: Carácter no reconocido '0' en línea 23

LexToken(SEMICOLON,';',23,640)

LexToken(RBRACE,'}',24,642)

PS C:\Users\user\Desktop\traductores\Hoy\Tokkens-y-Exp.-Reguares\Traductores\Errores Lexicos> **Github**

[cristopherstrange25/Tokkens-y-Exp.-Reguares: Actividad 2: Definición de tokens y expresiones regulares.](https://github.com/cristopherstrange25/Tokkens-y-Exp.-Reguares/tree/main)

**Video**

<https://youtu.be/Zw6Ms9GDgak?si=nfkU-OzzXlQmzKFO>