**UNIVERSIDAD DE GUADALAJARA**

**CENTRO UNIVERSITARIO DE TONALÁ**

**Logotipo

Descripción generada automáticamente**

**Materia: Traductores de Lenguaje.**

**Tema: Tokens y expresiones regulares**

**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 alfabético

    printf("TRADUCTORES DE LENGUAJES:\n");

    printf("Actividad 2\n");

    // Código original

    int x = 10;

    float y = 3.14;

    if (x > y) {

        printf("Hola, mundo!\n");

    }

    // Imprimir nombres completos en orden alfabético al final

    printf("\nIntegrantes del equipo:\n");

    printf("1. Celestino Martinez Cristopher\n");

    printf("2. Ramirez Rivas Luis Fernando\n");

    printf("3. Rivas Tejeda Hector Manuel\n");

    return 0;

}

El código proporcionado es un programa simple escrito en C que realiza las siguientes acciones:

1. **Impresión de un encabezado**: El programa comienza imprimiendo un encabezado que indica el nombre de la actividad y el título "TRADUCTORES DE LENGUAJES".
2. **Declaración de variables**: Se declaran dos variables:
   * int x = 10;: Una variable entera x inicializada con el valor 10.
   * float y = 3.14;: Una variable flotante y inicializada con el valor 3.14.
3. **Condicional if**: Se utiliza una estructura condicional if para comparar el valor de x con el valor de y. Si x es mayor que y, se imprime el mensaje "Hola, mundo!".
4. **Impresión de nombres**: Finalmente, el programa imprime los nombres completos de los integrantes del equipo en orden alfabético.

**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**:

python

Copy

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**:

python

Copy

lexer.input(code)

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

* + **Mostrar los tokens generados**:

python

Copy

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**:

python

Copy

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', 'FLOAT\_KEYWORD', '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

]

# Palabras clave del lenguaje C

reserved = {

    'auto': 'AUTO', 'break': 'BREAK', 'case': 'CASE', 'char': 'CHAR',

    'const': 'CONST', 'continue': 'CONTINUE', 'default': 'DEFAULT', 'do': 'DO',

    'double': 'DOUBLE', 'else': 'ELSE', 'enum': 'ENUM', 'extern': 'EXTERN',

    'float': 'FLOAT', 'for': 'FOR', 'goto': 'GOTO', 'if': 'IF', 'int': 'INT',

    'long': 'LONG', 'register': 'REGISTER', 'return': 'RETURN', 'short': 'SHORT',

    'signed': 'SIGNED', 'sizeof': 'SIZEOF', 'static': 'STATIC', 'struct': 'STRUCT',

    'switch': 'SWITCH', 'typedef': 'TYPEDEF', 'union': 'UNION', 'unsigned': 'UNSIGNED',

    'void': 'VOID', 'volatile': 'VOLATILE', 'while': 'WHILE'

}

# Agregar las palabras clave como tokens

tokens += list(reserved.values())

# Reglas para los tokens

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'\.'

# Reglas complejas

def t\_IDENTIFIER(t):

    r'[a-zA-Z\_][a-zA-Z0-9\_]\*'

    t.type = reserved.get(t.value, 'IDENTIFIER')  # Verificar si es palabra clave

    return t

def t\_FLOAT(t):

    r'\d+\.\d+'

    t.value = float(t.value)

    return t

def t\_INTEGER(t):

    r'\d+'

    t.value = int(t.value)

    return t

def t\_STRING(t):

    r'"([^"\\]|\\.)\*"'

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

    return t

# Comentarios

def t\_COMMENT(t):

    r'//.\*|/\\*[\s\S]\*?\\*/'

    pass  # Ignorar comentarios

# Ignorar espacios y tabulaciones

t\_ignore = ' \t'

# Ignorar nuevas líneas (pero contar las líneas para depuración)

def t\_newline(t):

    r'\n+'

    t.lexer.lineno += len(t.value)

# Manejo de errores

def t\_error(t):

    print(f"Illegal character '{t.value[0]}' at line {t.lexer.lineno}")

    t.lexer.skip(1)

# Construir el lexer

lexer = lex.lex()

# Reglas para los tokens

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'\.'

# Reglas complejas

def t\_IDENTIFIER(t):

    r'[a-zA-Z\_][a-zA-Z0-9\_]\*'

    t.type = reserved.get(t.value, 'IDENTIFIER')  # Verificar si es palabra clave

    return t

def t\_FLOAT(t):

    r'\d+\.\d+'

    t.value = float(t.value)

    return t

def t\_INTEGER(t):

    r'\d+'

    t.value = int(t.value)

    return t

def t\_STRING(t):

    r'"([^"\\]|\\.)\*"'

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

    return t

# Comentarios

def t\_COMMENT(t):

    r'//.\*|/\\*[\s\S]\*?\\*/'

    pass  # Ignorar comentarios

# Ignorar espacios y tabulaciones

t\_ignore = ' \t'

# Ignorar nuevas líneas (pero contar las líneas para depuración)

def t\_newline(t):

    r'\n+'

    t.lexer.lineno += len(t.value)

# Manejo de errores

def t\_error(t):

    print(f"Illegal character '{t.value[0]}' at line {t.lexer.lineno}")

    t.lexer.skip(1)

# Construir el lexer

lexer = lex.lex()

def t\_STRING(t):

    r'"([^"\\]|\\.)\*"'

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

    return t

# Comentarios

def t\_COMMENT(t):

    r'//.\*|/\\*[\s\S]\*?\\*/'

    pass  # Ignorar comentarios

# Ignorar espacios y tabulaciones

t\_ignore = ' \t'

# Ignorar nuevas líneas (pero contar las líneas para depuración)

def t\_newline(t):

    r'\n+'

    t.lexer.lineno += len(t.value)

# Manejo de errores

def t\_error(t):

    print(f"Illegal character '{t.value[0]}' at line {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**

1. **Lista de Tokens**:

python

Copy

tokens = [

'IDENTIFIER', 'INTEGER', 'FLOAT', 'FLOAT\_KEYWORD', '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

]

Aquí se define una lista de tokens que el lexer puede reconocer. Estos tokens incluyen identificadores, literales (números, cadenas), operadores, símbolos, etc.

1. **Palabras Reservadas**:

python

Copy

reserved = {

'auto': 'AUTO', 'break': 'BREAK', 'case': 'CASE', 'char': 'CHAR',

'const': 'CONST', 'continue': 'CONTINUE', 'default': 'DEFAULT', 'do': 'DO',

'double': 'DOUBLE', 'else': 'ELSE', 'enum': 'ENUM', 'extern': 'EXTERN',

'float': 'FLOAT', 'for': 'FOR', 'goto': 'GOTO', 'if': 'IF', 'int': 'INT',

'long': 'LONG', 'register': 'REGISTER', 'return': 'RETURN', 'short': 'SHORT',

'signed': 'SIGNED', 'sizeof': 'SIZEOF', 'static': 'STATIC', 'struct': 'STRUCT',

'switch': 'SWITCH', 'typedef': 'TYPEDEF', 'union': 'UNION', 'unsigned': 'UNSIGNED',

'void': 'VOID', 'volatile': 'VOLATILE', 'while': 'WHILE'

}

Se define un diccionario de palabras reservadas del lenguaje C, como int, float, if, return, etc. Estas palabras clave se añaden a la lista de tokens.

1. **Reglas para Tokens Simples**:  
   Para los tokens simples (como operadores y símbolos), se definen expresiones regulares directamente. Por ejemplo:

python

Copy

t\_PLUS = r'\+'

t\_MINUS = r'-'

t\_TIMES = r'\\*'

t\_DIVIDE = r'/'

t\_MODULO = r'%'

t\_ASSIGN = r'='

Estas reglas indican cómo reconocer los operadores aritméticos y el operador de asignación.

1. **Reglas para Tokens Complejos**:  
   Para tokens más complejos (como identificadores, números y cadenas), se definen funciones. Por ejemplo:
   * **Identificadores**:

python

Copy

def t\_IDENTIFIER(t):

r'[a-zA-Z\_][a-zA-Z0-9\_]\*'

t.type = reserved.get(t.value, 'IDENTIFIER') # Verificar si es palabra clave

return t

Esta función reconoce identificadores y verifica si son palabras reservadas.

* + **Números Flotantes**:

python

Copy

def t\_FLOAT(t):

r'\d+\.\d+'

t.value = float(t.value)

return t

Esta función reconoce números flotantes y los convierte a tipo float.

* + **Números Enteros**:

python

Copy

def t\_INTEGER(t):

r'\d+'

t.value = int(t.value)

return t

Esta función reconoce números enteros y los convierte a tipo int.

* + **Cadenas**:

python

Copy

def t\_STRING(t):

r'"([^"\\]|\\.)\*"'

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

return t

Esta función reconoce cadenas de texto y elimina las comillas.

1. **Comentarios**:

python

Copy

def t\_COMMENT(t):

r'//.\*|/\\*[\s\S]\*?\\*/'

pass # Ignorar comentarios

Los comentarios (tanto de una línea como de múltiples líneas) se reconocen y se ignoran.

1. **Ignorar Espacios y Tabulaciones**:

python

Copy

t\_ignore = ' \t'

Los espacios y tabulaciones se ignoran en el análisis léxico.

1. **Manejo de Nuevas Líneas**:

python

Copy

def t\_newline(t):

r'\n+'

t.lexer.lineno += len(t.value)

Las nuevas líneas se reconocen y se utilizan para llevar un conteo de líneas (útil para mensajes de error).

1. **Manejo de Errores**:

python

Copy

def t\_error(t):

print(f"Illegal character '{t.value[0]}' at line {t.lexer.lineno}")

t.lexer.skip(1)

Si se encuentra un carácter no reconocido, se imprime un mensaje de error y se ignora el carácter.

1. **Construir el Lexer**:

python

Copy

lexer = lex.lex()

Finalmente, se construye el lexer utilizando la función lex.lex().

**Salida**Tokens encontrados:

Illegal character '#' at line 1

LexToken(IDENTIFIER,'include',1,1)

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

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

LexToken(DOT,'.',1,15)

LexToken(IDENTIFIER,'h',1,16)

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

LexToken(INT,'int',7,116)

LexToken(IDENTIFIER,'main',7,120)

LexToken(LPAREN,'(',7,124)

LexToken(RPAREN,')',7,125)

LexToken(LBRACE,'{',7,127)

LexToken(IDENTIFIER,'printf',9,188)

LexToken(LPAREN,'(',9,194)

LexToken(STRING,'TRADUCTORES DE LENGUAJES:\\n',9,195)

LexToken(RPAREN,')',9,224)

LexToken(SEMICOLON,';',9,225)

LexToken(IDENTIFIER,'printf',10,231)

LexToken(LPAREN,'(',10,237)

LexToken(STRING,'Actividad 2\\n',10,238)

LexToken(RPAREN,')',10,253)

LexToken(SEMICOLON,';',10,254)

LexToken(INT,'int',13,285)

LexToken(IDENTIFIER,'x',13,289)

LexToken(ASSIGN,'=',13,291)

LexToken(INTEGER,10,13,293)

LexToken(SEMICOLON,';',13,295)

LexToken(FLOAT,'float',14,301)

LexToken(IDENTIFIER,'y',14,307)

LexToken(ASSIGN,'=',14,309)

LexToken(FLOAT,3.14,14,311)

LexToken(SEMICOLON,';',14,315)

LexToken(IF,'if',15,321)

LexToken(LPAREN,'(',15,324)

LexToken(IDENTIFIER,'x',15,325)

LexToken(GT,'>',15,327)

LexToken(IDENTIFIER,'y',15,329)

LexToken(RPAREN,')',15,330)

LexToken(LBRACE,'{',15,332)

LexToken(IDENTIFIER,'printf',16,342)

LexToken(LPAREN,'(',16,348)

LexToken(STRING,'Hola, mundo!\\n',16,349)

LexToken(RPAREN,')',16,365)

LexToken(SEMICOLON,';',16,366)

LexToken(RBRACE,'}',17,372)

LexToken(IDENTIFIER,'printf',20,443)

LexToken(LPAREN,'(',20,449)

LexToken(STRING,'\\nIntegrantes del equipo:\\n',20,450)

LexToken(RPAREN,')',20,479)

LexToken(SEMICOLON,';',20,480)

LexToken(IDENTIFIER,'printf',21,486)

LexToken(LPAREN,'(',21,492)

LexToken(STRING,'1. Celestino Martinez Cristopher\\n',21,493)

LexToken(RPAREN,')',21,529)

LexToken(SEMICOLON,';',21,530)

LexToken(IDENTIFIER,'printf',22,536)

LexToken(LPAREN,'(',22,542)

LexToken(STRING,'2. Ramirez Rivas Luis Fernando\\n',22,543)

LexToken(RPAREN,')',22,577)

LexToken(SEMICOLON,';',22,578)

LexToken(IDENTIFIER,'printf',23,584)

LexToken(LPAREN,'(',23,590)

LexToken(STRING,'2. Ramirez Rivas Luis Fernando\\n',22,543)

LexToken(RPAREN,')',22,577)

LexToken(SEMICOLON,';',22,578)

LexToken(IDENTIFIER,'printf',23,584)

LexToken(LPAREN,'(',23,590)

LexToken(RPAREN,')',22,577)

LexToken(SEMICOLON,';',22,578)

LexToken(IDENTIFIER,'printf',23,584)

LexToken(LPAREN,'(',23,590)

LexToken(SEMICOLON,';',22,578)

LexToken(IDENTIFIER,'printf',23,584)

LexToken(LPAREN,'(',23,590)

LexToken(IDENTIFIER,'printf',23,584)

LexToken(LPAREN,'(',23,590)

LexToken(LPAREN,'(',23,590)

LexToken(STRING,'3. Rivas Tejeda Hector Manuel\\n',23,591)

LexToken(RPAREN,')',23,624)

LexToken(SEMICOLON,';',23,625)

LexToken(SEMICOLON,';',23,625)

LexToken(RETURN,'return',25,632)

LexToken(RETURN,'return',25,632)

LexToken(INTEGER,0,25,639)

LexToken(SEMICOLON,';',25,640)

LexToken(RBRACE,'}',26,642)

PS C:\Users\user\Desktop\traductores>  
 **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>