Aim: - Write a program to generate tokens for given lexeme

#include <iostream>

#include <string>

#include <vector>

#include <cctype> // for isalpha, isdigit, etc.

enum TokenType {

    IDENTIFIER,

    NUMBER,

    OPERATOR,

    PUNCTUATION,

    UNKNOWN

};

struct Token {

    TokenType type;

    std::string value;

};

std::vector<Token> tokenize(const std::string &lexeme) {

    std::vector<Token> tokens;

    size\_t i = 0;

    while (i < lexeme.size()) {

        if (std::isalpha(lexeme[i])) {

            // Process identifier (e.g., variable names, function names)

            std::string identifier;

            while (i < lexeme.size() && (std::isalnum(lexeme[i]) || lexeme[i] == '\_')) {

                identifier += lexeme[i++];

            }

            tokens.push\_back({IDENTIFIER, identifier});

        } else if (std::isdigit(lexeme[i])) {

            // Process number

            std::string number;

            while (i < lexeme.size() && std::isdigit(lexeme[i])) {

                number += lexeme[i++];

            }

            tokens.push\_back({NUMBER, number});

        } else if (std::ispunct(lexeme[i])) {

            // Process punctuation or operators

            std::string op(1, lexeme[i]);

            tokens.push\_back({OPERATOR, op});

            i++;

        } else {

            // Unknown token

            std::string unknown(1, lexeme[i]);

            tokens.push\_back({UNKNOWN, unknown});

            i++;

        }

    }

    return tokens;

}

void printTokens(const std::vector<Token> &tokens) {

    for (const auto &token : tokens) {

        std::string type;

        switch (token.type) {

            case IDENTIFIER: type = "IDENTIFIER"; break;

            case NUMBER: type = "NUMBER"; break;

            case OPERATOR: type = "OPERATOR"; break;

            case PUNCTUATION: type = "PUNCTUATION"; break;

            case UNKNOWN: type = "UNKNOWN"; break;

        }

        std::cout << "Token Type: " << type << ", Value: " << token.value << std::endl;

    }

}

int main() {

    std::string lexeme;

    std::cout << "Enter a lexeme: ";

    std::getline(std::cin, lexeme);

    std::vector<Token> tokens = tokenize(lexeme);

    printTokens(tokens);

    return 0;

}

Output: -

