#include <iostream>

#include <fstream>

#include <string>

#include <regex>

#include <unordered\_set>

using namespace std;

#define DEBUG\_MODE false  // Set to true for debugging output

void analyzeCode(const string& filename) {

    ifstream file(filename);

    if (!file) {

        cerr << "Error: Cannot open file." << endl;

        return;

    }

    string line;

    int lineNumber = 0;

    // Issue tracking flags

    bool hasDeepNesting = false;

    bool hasLongFunctions = false;

    bool hasPoorIndentation = false;

    regex semicolonCheck(R"(^\s\*(?!#|for|while|if|else|switch|return|do|case|default|namespace|class|struct|void|int|char|float|double|bool|template)[a-zA-Z0-9\_]+.\*[^;{}]\s\*$)");

    regex unusedVarCheck(R"(\b(int|float|double|char|bool)\s+([a-zA-Z\_][a-zA-Z0-9\_]\*)\s\*;)"); // Detects variable declarations

    regex varUsageCheck(R"(\b([a-zA-Z\_][a-zA-Z0-9\_]\*)\b)"); // Detects variable usage

    regex indentationCheck(R"(^\s{5,})");  // Checks if line starts with 5+ spaces

    regex deepNestingCheck(R"((?:\{[^{}]\*\{){3,})");  // Detects nesting beyond 2 levels

    regex longFunctionCheck(R"((?:\{[^{}]\*\}){16,})");  // Detects functions longer than 15 lines

    unordered\_set<string> declaredVars;

    unordered\_set<string> usedVars;

    int braceDepth = 0;  // Tracks nesting depth

    int functionLineCount = 0;  // Tracks function length

    cout << "{\n  \"issues\": [\n";

    bool firstIssue = true;

    while (getline(file, line)) {

        lineNumber++;

        // Detect missing semicolon (Refined)

        if (regex\_match(line, semicolonCheck) && !regex\_search(line, regex(R"(\b(if|for|while|switch|else|return|do)\b)"))) {

            if (!firstIssue) cout << ",\n";

            cout << "    {\"line\": " << lineNumber << ", \"message\": \"Possible missing semicolon.\"}";

            firstIssue = false;

        }

        // Detect variable declarations

        smatch match;

        if (regex\_search(line, match, unusedVarCheck)) {

            declaredVars.insert(match[2]);  // Store variable name

            if (DEBUG\_MODE) cout << "[DEBUG] Declared: " << match[2] << " at line " << lineNumber << endl;

        }

        // Detect variable usage

        smatch varMatch;

        unordered\_set<string> detectedUsageInLine;  // Prevent duplicate detections in the same line

        string tempLine = line;

        while (regex\_search(tempLine, varMatch, varUsageCheck)) {

            string var = varMatch.str();

            if (declaredVars.find(var) != declaredVars.end() && detectedUsageInLine.find(var) == detectedUsageInLine.end()) {

                usedVars.insert(var);

                detectedUsageInLine.insert(var);  // Avoid duplicate detections in a single line

                if (DEBUG\_MODE) cout << "[DEBUG] Used: " << var << " at line " << lineNumber << endl;

            }

            tempLine = varMatch.suffix().str();

        }

        // Detect indentation issues

        if (regex\_match(line, indentationCheck)) {

            hasPoorIndentation = true;

        }

        // Track nesting depth

        for (char c : line) {

            if (c == '{') braceDepth++;

            if (c == '}') braceDepth--;

        }

        if (braceDepth > 3) hasDeepNesting = true;  // Adjust threshold as needed

        // Track function length

        if (braceDepth > 0) {

            functionLineCount++;

        } else {

            if (functionLineCount > 15) {  // Adjust threshold for function size

                hasLongFunctions = true;

            }

            functionLineCount = 0;

        }

    }

    // Report unused variables

    for (const auto& var : declaredVars) {

        if (usedVars.find(var) == usedVars.end()) {

            if (!firstIssue) cout << ",\n";

            cout << "    {\"message\": \"Potential unused variable: " << var << "\"}";

            firstIssue = false;

        }

    }

    // Report other issues

    if (hasDeepNesting) {

        if (!firstIssue) cout << ",\n";

        cout << "    {\"message\": \"Code may be deeply nested. Consider refactoring.\"}";

        firstIssue = false;

    }

    if (hasLongFunctions) {

        if (!firstIssue) cout << ",\n";

        cout << "    {\"message\": \"Function might be too long. Consider breaking it into smaller functions.\"}";

        firstIssue = false;

    }

    if (hasPoorIndentation) {

        if (!firstIssue) cout << ",\n";

        cout << "    {\"message\": \"Poor indentation detected. Consider improving code formatting.\"}";

    }

    cout << "\n  ]\n}" << endl;

}

int main(int argc, char\* argv[]) {

    if (argc != 2) {

        cerr << "Usage: " << argv[0] << " <filename>" << endl;

        return 1;

    }

    analyzeCode(argv[1]);

    return 0;

}

/\*  g++ -std=c++17 cpp\_analyzer.cpp -o analyzer

./analyzer test.cpp                  \*/