#include <iostream>

#include <sstream>

#include <vector>

#include <string>

#include <map>

#include <set>

#include <cctype>

using namespace std;

// Function to remove comments from the code and return them separately

std::pair<std::string, std::set<std::string>> removeComments(const std::string& code) {

std::set<std::string> comments;

std::string noComments;

bool inComment = false;

std::string currentComment;

for (size\_t i = 0; i < code.size(); ++i) {

if (inComment) {

if (code[i] == '\n') {

inComment = false;

comments.insert(currentComment); // Add full comment to set

currentComment.clear(); // Clear for the next comment

} else {

currentComment += code[i]; // Build comment line

}

} else if (i < code.size() - 1 && code[i] == '/' && code[i + 1] == '/') {

inComment = true;

currentComment = "//"; // Start new comment

++i; // Skip the second '/'

} else {

noComments += code[i];

}

}

// If a comment is at the end without a newline, add it to the set

if (!currentComment.empty()) {

comments.insert(currentComment);

}

return {noComments, comments};

}

// Function to remove excess spaces from the code

std::string removeExcessSpaces(const std::string& code) {

std::string result;

bool inSpace = false;

for (char ch : code) {

if (isspace(ch)) {

if (!inSpace) {

result += ' ';

inSpace = true;

}

} else {

result += ch;

inSpace = false;

}

}

return result;

}

// Helper function to check if a string is a keyword

bool isKeyword(const std::string& token) {

return token == "void" || token == "int" || token == "return";

}

// Helper function to check if a string is an identifier

bool isIdentifier(const std::string& token) {

return token == "cout" || token == "endl" || token == "greet" || token == "main";

}

// Function to tokenize the code and categorize tokens

void categorizeTokens(const std::string& code, std::map<std::string, int>& keywords, std::map<std::string, int>& identifiers,

std::map<std::string, int>& literals, std::map<std::string, int>& operators, std::map<std::string, int>& separators) {

std::string token;

bool inLiteral = false;

for (size\_t i = 0; i < code.size(); ++i) {

char ch = code[i];

if (inLiteral) {

if (ch == '"') {

literals[token + '"']++;

token.clear();

inLiteral = false;

} else {

token += ch;

}

} else if (ch == '"') {

inLiteral = true;

token = '"';

} else if (isalnum(ch) || ch == '\_') {

token += ch;

} else {

if (!token.empty()) {

if (isKeyword(token)) {

keywords[token]++;

} else if (isIdentifier(token)) {

identifiers[token]++;

} else if (isdigit(token[0])) {

literals[token]++;

}

token.clear();

}

if (ch == '=' || ch == '<' || ch == '>' || ch == '+') {

if (i + 1 < code.size() && (code[i + 1] == '=' || code[i + 1] == '<' || code[i + 1] == '>')) {

operators[std::string(1, ch) + code[i + 1]]++;

++i;

} else {

operators[std::string(1, ch)]++;

}

} else if (ch == '(' || ch == ')' || ch == '{' || ch == '}' || ch == ';') {

separators[std::string(1, ch)]++;

}

}

}

}

int main() {

std::string code;

std::cout << "Enter code (end with an empty line):" << std::endl;

// Reading multiple lines of code from the user

std::string line;

while (std::getline(std::cin, line) && !line.empty()) {

code += line + "\n";

}

// Remove comments and store them

std::pair<std::string, std::set<std::string>> result = removeComments(code);

std::string codeWithoutComments = result.first;

std::set<std::string> allComments = result.second;

// Remove excess spaces for formatting

std::string processedCode = removeExcessSpaces(codeWithoutComments);

// Categorize tokens

std::map<std::string, int> keywords, identifiers, literals, operators, separators;

categorizeTokens(processedCode, keywords, identifiers, literals, operators, separators);

// Output tokens

std::cout << "Tokens:\n";

std::cout << "• Keywords: ";

for (const auto& kw : keywords) std::cout << kw.first << " (" << kw.second << "), ";

std::cout << "\n• Identifiers: ";

for (const auto& id : identifiers) std::cout << id.first << " (" << id.second << "), ";

std::cout << "\n• Literals: ";

for (const auto& lit : literals) std::cout << lit.first << " (" << lit.second << "), ";

std::cout << "\n• Operators: ";

for (const auto& op : operators) std::cout << op.first << " (" << op.second << "), ";

std::cout << "\n• Separators: ";

for (const auto& sep : separators) std::cout << sep.first << " (" << sep.second << "), ";

std::cout << "\n• Comments: ";

for (const auto& comment : allComments) std::cout << comment << ", ";

// Count total tokens

int totalTokens = 0;

for (const auto& kw : keywords) totalTokens += kw.second;

for (const auto& id : identifiers) totalTokens += id.second;

for (const auto& lit : literals) totalTokens += lit.second;

for (const auto& op : operators) totalTokens += op.second;

for (const auto& sep : separators) totalTokens += sep.second;

totalTokens += allComments.size();

std::cout << "\nTotal: " << totalTokens << std::endl;

return 0;

}