Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
По дисциплине «Надежность программного обеспечения»
Тема: «Оценка надежности с использованием метрик»

Выполнил:
Студент 3 курса
Группы ПО-11
Хведорец В. С.
Проверил:
Козик И. Д.

Брест 2025

**Цель работы:** Изучить метрики надежности программного обеспечения и применить их для оценки качества кода.

Рассчитать метрики надежности. Сравнить метрики до и после улучшения кода. Сделать выводы о влиянии изменений на надежность программы

**Код программы:**

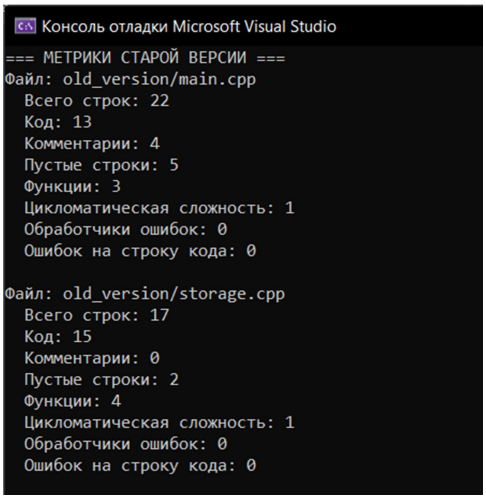
```
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <regex>
#include <cmath>
class CodeAnalyzer {
private:
    struct CodeMetrics {
        int totalLines = 0;
        int codeLines = 0;
        int commentLines = 0;
        int blankLines = 0;
        int functions = 0;
        int complexity = 0;
        int errorHandlers = 0;
    };
    std::map<std::string, CodeMetrics> metrics;
    void analyzeFile(const std::string& filename) {
        std::ifstream file(filename);
        if (!file.is_open()) throw std::runtime_error("Cannot open file: " + filename);
        CodeMetrics current;
        std::string line;
        bool inCommentBlock = false;
        while (getline(file, line)) {
            current.totalLines++;
            // удаляем лишние пробелы
            line = std::regex_replace(line, std::regex("^\\s+|\\s+$"), "");
            if (line.empty()) {
                current.blankLines++;
                continue;
            }
            // анализ комментариев
            if (std::regex_match(line, std::regex("^\\/\\/.*|^\\/\\*.*"))) {
                current.commentLines++;
                inCommentBlock = line.find("/*") != std::string::npos;
                continue;
            }
            if (inCommentBlock) {
                current.commentLines++;
                if (line.find("*/") != std::string::npos) inCommentBlock = false;
                continue;
            }
            current.codeLines++;
            // анализ функций
            if (std::regex_match(line, std::regex(".*\\s+\\w+\\(.*\\).*\\{?"))) {
                current.functions++;
            }
            // анализ сложности
            if (std::regex_match(line, std::regex(".*(if|else if|case|for|while|catch).*"))) {
                current.complexity++;
            }
            // анализ обработчиков ошибок
            if (std::regex_match(line, std::regex(".*(try|catch|throw|exception).*"))) {
                current.errorHandlers++;
            }
        }
        metrics[filename] = current;
    }
public:
    void analyze(const std::vector<std::string>& files) {
```

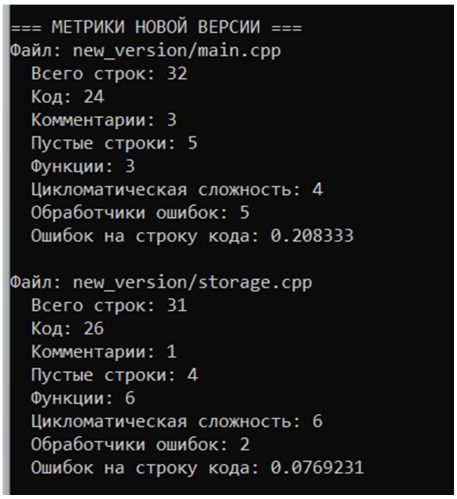
```
        for (const auto& file : files) {
            analyzeFile(file);
        }
    }
    void printMetrics() const {
        for (const auto& [file, metric] : metrics) {
            std::cout << "Файл: " << file << "\n"
                << "  Всего строк: " << metric.totalLines << "\n"
                << "  Код: " << metric.codeLines << "\n"
                << "  Комментарии: " << metric.commentLines << "\n"
                << "  Пустые строки: " << metric.blankLines << "\n"
                << "  Функции: " << metric.functions << "\n"
                << "  Цикломатическая сложность: " << metric.complexity << "\n"
                << "  Обработчики ошибок: " << metric.errorHandlers << "\n"
                << "  Ошибок на строку кода: "
                << (metric.errorHandlers / (float)metric.codeLines) << "\n\n";
        }
    }
    std::string extractFileName(const std::string& path) {
        size_t pos = path.find_last_of("/\\");
        return (pos == std::string::npos) ? path : path.substr(pos + 1);
    }
    void compareMetrics(const CodeAnalyzer& oldVersion) {
        std::cout << "=== Сравнение метрик ===\n";
        for (const auto& [newPath, newMetric] : metrics) {
            std::string fileName = extractFileName(newPath);
            // ищем файл с таким же именем в oldVersion
            const auto it = std::find_if(
                oldVersion.metrics.begin(),
                oldVersion.metrics.end(),
                [&](const auto& pair) {
                    return extractFileName(pair.first) == fileName;
                });
            if (it != oldVersion.metrics.end()) {
                const auto& oldMetric = it->second;
                std::cout << "Файл: " << fileName << "\n";
                printComparison("Строк кода", oldMetric.codeLines, newMetric.codeLines);
                printComparison("Цикломат. сложность", oldMetric.complexity,
newMetric.complexity);
                printComparison("Обработчики ошибок", oldMetric.errorHandlers,
newMetric.errorHandlers);
                float oldErrRate = oldMetric.codeLines ? (oldMetric.errorHandlers /
(float)oldMetric.codeLines) : 0;
                float newErrRate = newMetric.codeLines ? (newMetric.errorHandlers /
(float)newMetric.codeLines) : 0;
                printComparison("Ошибок/строку", oldErrRate, newErrRate);
                std::cout << "----------------------\n";
            }
        }
    }
private:
    void printComparison(const std::string& name, float oldVal, float newVal) const {
        std::cout << "  " << name << ": " << oldVal << " -> " << newVal
            << " (" << ((newVal - oldVal) / oldVal * 100) << "%)\n";
    }
};
int main() {
    setlocale(LC_ALL, "rus");
    // анализ старой версии
    CodeAnalyzer oldVersion;
    oldVersion.analyze({
        "old_version/main.cpp",
        "old_version/storage.cpp"
        });
    std::cout << "=== МЕТРИКИ СТАРОЙ ВЕРСИИ ===\n";
    oldVersion.printMetrics();
    // анализ новой версии
    CodeAnalyzer newVersion;
    newVersion.analyze({
        "new_version/main.cpp",
        "new_version/storage.cpp"
        });
```

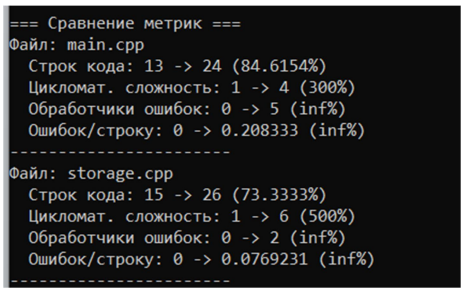
```
    std::cout << "\n=== МЕТРИКИ НОВОЙ ВЕРСИИ ===\n";
    newVersion.printMetrics();
    // сравнение метрик
    newVersion.compareMetrics(oldVersion);

    return 0;
}
```

**Результат работы:**




**Вывод:** Изучил метрики надежности программного обеспечения.