Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №12
«Обработка текстовых файлов»
по курсу: «Языки и методы программирования»

Выполнил:
Студент группы ИУ9-21Б
Старовойтов А. И.

Проверил:
Посевин Д. П.

Москва, 2022

Цели

Целью лабораторной работы является приобретение навыка разработки на языке C++ программ, осуществляющих анализ и преобразование текстовых файлов, записанных в различных форматах.

Задачи

В указанном каталоге лежат файлы тестов для сервера тестирования системы T-BMSTU. Каждый тест оформлен в виде двух файлов с именами «N» и «N.a», в которых содержатся входные данные для теста и правильный ответ, соответственно. Здесь N – это целое число, обозначающее номер теста. Требуется сгенерировать по набору файлов тестов таблицу в формате MarkDown. Таблица должна иметь следующий вид:

```
+--------------+-----+
|Входные данные|Ответ|
+==============+=====+
|abcd          |1    |
+--------------+-----+
|qwerty        |2    |
|abcd          |     |
+--------------+-----+
```

Решение

main.cpp

```cpp
#include <algorithm>
#include <filesystem>
#include <fstream>
#include <iostream>
#include <regex>
#include <vector>

namespace fs = std::filesystem;

struct Test {
```

```
    std::vector<std::string> input;
    std::vector<std::string> answer;
};

void printHelp(char* progName) {
    std::cout << "Usage: " << progName << " PATH\n";
}

std::vector<std::string> readTestFile(fs::path path) {
    std::ifstream fs(path);
    std::vector<std::string> res;
    std::string line;
    while (std::getline(fs, line)) {
        res.push_back(line);
    }
    return res;
}

std::vector<fs::path> getTestsPaths(fs::path dir) {
    std::vector<fs::path> res;
    for (const auto& entry : fs::directory_iterator(dir)) {
        if (!entry.is_regular_file()) {
            continue;
        }
        const auto& entryPath = entry.path();
        if (std::regex_match(entryPath.filename().string(),
                             std::regex{R"(\d+)"})) {
            res.push_back(entryPath);
        }
    }
    return res;
}

std::vector<Test> readTests(fs::path dir) {
    std::vector<Test> tests;
    auto paths = getTestsPaths(dir);
    std::sort(paths.begin(), paths.end());
    for (const auto& path : paths) {
        tests.push_back(
            Test{readTestFile(path),
 ↪  readTestFile(path.string() + ".a")});
```

```
    }
    return tests;
}

std::ostream& tableLineOutput(std::ostream& out, std::size_t
 ↪  inputWidth,
                              std::string_view inputLine,
                              std::size_t answerWidth,
                              std::string_view answerLine) {
    out << '|' << std::left <<
 ↪  std::setw(static_cast<int>(inputWidth))
        << inputLine << std::setw(0) << '|'
        << std::setw(static_cast<int>(answerWidth)) <<
         ↪  answerLine
        << std::setw(0) << '|';
    return out;
}

std::ostream& delimiterLineOutput(std::ostream& out,
 ↪  std::size_t inputWidth,
                                  std::size_t answerWidth,
                                   ↪  char c) {
    out << '+' << std::string(inputWidth, c) << '+'
        << std::string(answerWidth, c) << '+';
    return out;
}

std::pair<std::size_t, std::size_t>
countColoumnsWidths(std::size_t inputColTitleSize,
                    std::size_t answerColTitleSize,
                    const std::vector<Test>& tests) {
    std::size_t inputWidth = inputColTitleSize;
    std::size_t answerWidth = answerColTitleSize;
    for (const auto& [input, answer] : tests) {
        for (const auto& line : input) {
            inputWidth = std::max(inputWidth, line.size());
        }
        for (const auto& line : answer) {
            answerWidth = std::max(answerWidth,
 ↪  line.size());
```

```
        }
    }
    return {inputWidth, answerWidth};
}

std::ostream& testsMdTableOutput(std::ostream& out,
                                 const std::vector<Test>&
                                  ↪  tests) {
    constexpr std::string_view inputColName = "Input";
    constexpr std::string_view answerColName = "Answer";
    auto [inputWidth, answerWidth] =
        countColoumnsWidths(inputColName.size(),
 ↪  answerColName.size(), tests);
    delimiterLineOutput(out, inputWidth, answerWidth, '-')
 ↪  << '\n';
    tableLineOutput(out, inputWidth, inputColName,
 ↪  answerWidth, answerColName)
        << '\n';
    delimiterLineOutput(out, inputWidth, answerWidth, '=');
    for (const auto& [input, answer] : tests) {
        out << '\n';
        std::size_t i;
        for (i = 0; i < std::min(input.size(),
         ↪  answer.size()); i++) {
            tableLineOutput(out, inputWidth, input[i],
 ↪  answerWidth, answer[i])
                << '\n';
        }
        for (; i < input.size(); i++) {
            tableLineOutput(out, inputWidth, input[i],
 ↪  answerWidth, "") << '\n';
        }
        for (; i < answer.size(); i++) {
            tableLineOutput(out, inputWidth, "",
 ↪  answerWidth, answer[i])
                << '\n';
        }
        delimiterLineOutput(out, inputWidth, answerWidth,
 ↪  '-');
    }
```

```
    return out;
}

int main(int argc, char** argv) {
    if (argc != 2) {
        printHelp(argv[0]);
        return 0;
    }
    std::string parameter = argv[1];
    if (parameter == "-h" || parameter == "--help") {
        printHelp(argv[0]);
        return 0;
    }
    fs::path path = parameter;
    auto tests = readTests(path);
    testsMdTableOutput(std::cout, tests) << '\n';
    return 0;
}
```

Пример вывода

```
+--------------+------+
|Input         |Answer|
+==============+======+
|abcd          |123   |
|              |123   |
|              |123   |
|              |123   |
+--------------+------+
|abcdefghijklmn|12345 |
|effefe        |      |
+--------------+------+
|aaaa          |0     |
+--------------+------+
```