**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

(национальный исследовательский университет)»

**ЛАБОРАТОРНАЯ РАБОТА**

**по курсу**

**«Программирование на языках высокого уровня»**

|  |
| --- |
| Вариант 18  Часть 2 |
|  |

|  |
| --- |
| **Выполнил:** |
| Студент группы  М7О-307С-22 |
| Бурцев Никита Романович |
| **Принял:** |

Барчев Николай  
 Борисович

Москва, 2025 г.

Оглавление

[1. Задание 3](#_Toc191392632)

[2. Псевдокод 3](#_Toc191392633)

[3. Сведения о программной реализации 6](#_Toc191392634)

[3.1. Язык программирования и среда разработки 6](#_Toc191392635)

[3.2. Описание входных и выходных данных 6](#_Toc191392636)

[3.3. Описание программных единиц 7](#_Toc191392637)

[4. Инструкция пользователю 10](#_Toc191392638)

[5. Листинг программной разработки 11](#_Toc191392639)

[6. Результаты тестирования программной разработки 20](#_Toc191392640)

# Задание

1. Подготовить программу, формирующую на основе информации, вводимой пользователем с клавиатуры, два внешних файла:

- файл отправляемой корреспонденции: состоит из записей, каждая из которых включает три поля вида корреспонденции, даты подготовки и названия организации;

- файл адресов организаций: состоит из записей, каждая из которых включает три поля названия организации, ее адреса и фамилии руководителя.

2. Подготовить вторую программу, формирующую сведения на основе данных из двух внешних файлов, сформированных в результате работы программы, подготовленной в п. 1.

3. В процессе проектирования предусмотреть необходимые по смыслу задания проверки корректности данных, а также адекватное задаче взаимодействие с пользователем.

**Во второй программе реализовать возможность вывода итоговой информации в двух режимах: избирательно на экран (по запросу пользователем конкретных данных с клавиатуры) и полностью в отдельный текстовый файл. Оба режима вывода должны предоставлять СГРУППИРОВАННУЮ ПО СМЫСЛУ информацию ИЗ ОБОИХ ВХОДНЫХ ФАЙЛОВ в виде, УДОБНОМ ДЛЯ ВОСПРИЯТИЯ пользователем.   
Учесть, что ограничения на размеры файлов отсутствуют длины, файлы в общем случае могут быть различной длины.**

**В обязательном порядке использовать языковые средства организации программных единиц.   
Не использовать программные средства, поддерживающие работу с базами данных.**  
4. Программы подготовить с использованием средств одной из реализаций языка программирования С++, в первой программе ввод и вывод реализовать при помощи средств языка программирования С. Выполнить тестирование обеих программ согласно стратегии черного ящика, используя соответствующие критерии тестирования.

# Псевдокод

Глобальная переменная

currentFolderPath = ""

Функция is\_leap(year)

Вернуть (year кратно 4 И НЕ кратно 100) ИЛИ (year кратно 400)

Функция isValidFio(fio)

Вернуть результат проверки строки fio

Функция isValidDate(dateStr)

Вернуть результат проверки строки dateStr

Функция isValidFileName(fileName)

Вернуть результат проверки строки fileName

Функция readInstructionsFromFile(filename)

Вывести содержимое файла filename или ошибку

Функция getLineWithEsc(instruction)

Вывести instruction

Пока истина

Считать символ

Если Esc то Вернуть ""

Если Enter то Вернуть строку

Если Backspace то Удалить символ

Иначе Если символ допустимый то Добавить к строке

Вернуть строку

Функция processOrganization(orgName, corrFilename, addrFilename, selectiveOutput, outfile, printedOrganizations, outputBuffer)

Открыть файлы

addresses = []

orgFoundInAddress = ложь

Если selectiveOutput то

writeToBuffer = добавить в outputBuffer

Иначе

writeToBuffer = записать в outfile

Если файл адресов открыт то

Пока читать строку из файла адресов

Разобрать строку

Удалить пробелы

Если орг. совпадает то

Добавить в addresses

orgFoundInAddress = истина

Закрыть файл

Иначе

writeToBuffer ошибка

Вернуться

Если НЕ orgFoundInAddress И selectiveOutput то

writeToBuffer орг. не найдена

Вернуться

Если НЕ selectiveOutput

Если orgName уже выводилась то

Вернуться

Добавить orgName в printedOrganizations

writeToBuffer орг.

Для каждого адреса

writeToBuffer адрес

writeToBuffer фио

Если файл корр. открыт

foundCorrespondence = ложь

Пока читать строку

Разобрать

Удалить пробелы

Если орг. совпадает то

foundCorrespondence = истина

writeToBuffer

Если корр. не найдена то

writeToBuffer

Закрыть файл

Иначе writeToBuffer ошибка

Если selectiveOutput И orgFoundInAddress то

writeToBuffer орг.

Если addresses не пуст то

writeToBuffer адрес

writeToBuffer фио

Если есть другие адреса то

writeToBuffer другие данные

Для каждого адреса со второго

writeToBuffer адрес

writeToBuffer фио

Если файл корр. открыт

foundCorrespondence = ложь

Пока читать строку

Разобрать

Удалить пробелы

Если орг. совп. то

foundCorrespondence = истина

writeToBuffer

Если корр. не найдена то

writeToBuffer

Закрыть файл

Иначе writeToBuffer ошибка

writeToBuffer

Функция getFilenamesFromUser(folderPath)

Пока истина

corrFilename = getLineWithEsc ввод имени файла корр.

Если corrFilename пусто то Вернуть "", ""

fullCorrFilename = полный путь

Если файл не существует то Ошибка, Продолжить

Прервать цикл

Пока истина

addrFilename = getLineWithEsc ввод имени файла адр.

Если addrFilename пусто то Вернуть "", ""

fullAddrFilename = полный путь

Если файл не существует то Ошибка, Продолжить

Прервать цикл

Вернуть fullCorrFilename, fullAddrFilename

Функция runProgram(folderPath, correspondenceFilename, addressesFilename, outputFilename)

Пока истина

Вывести меню режима вывода

Ввод = считать символ

Если не '1', не '2', не Esc, то считать заново

Если Ввод = '1' то

orgName = getLineWithEsc ввод орг.

Если orgName пусто то Продолжить

outfile = nullptr

printedOrganizations = {}

outputBuffer = {}

processOrganization

Вывести outputBuffer

Иначе Если Ввод = '2' то

corrFileNameOnly = имя файла

addrFileNameOnly = имя файла

Удалить .txt

baseFilename = ...

filename = baseFilename + .txt

i = 1

Пока файл существует

filename = ... + (i) + .txt

i++

Попытка

Открыть файл для записи

Если не открылся то Ошибка, Продолжить

printedOrganizations = {}

outputBuffer = {}

Открыть файл адресов

Если открыт то

Пока читать строку

Разобрать

Удалить пробелы

Если орг. не выводилась то

processOrganization

Закрыть файл

Иначе Ошибка, Продолжить

Закрыть файл

Сообщение об успехе, Прервать

Исключение

Ошибка

Иначе Если Ввод = Esc то Вернуть

Функция menu()

correspondenceFilename = ""

addressesFilename = ""

outputFilename = ""

folderPathSet = true

currentFolderPath = норм. путь + "\"

instructionsFile = путь + имя файла

Приветствие

Пока истина

Очистить экран

Вывести меню

Вывести путь

Ввод = считать символ

Если Ввод = '1' то

filenames = getFilenamesFromUser

Если filenames пуст то Продолжить

corrFilename, addrFilename = filenames

runProgram

Иначе Если Ввод = '2' то

newPath = getLineWithEsc ввод пути

Если newPath пуст то Продолжить

newPath = нормализовать

Если папка существует то

currentFolderPath = newPath + "\"

Сообщение

Иначе Ошибка

Ждать

Иначе Если Ввод = '3' то

readInstructionsFromFile

Иначе Если Ввод = Esc то Вернуть

Функция normalizePath(path)

Вернуть абс. путь

Функция main()

Кодировка

Вызвать menu

Вернуть 0

# Сведения о программной реализации

# Язык программирования и среда разработки

Код написан на языке C++ (стандарт ISO C++ 17) в среде разработки Microsoft Visual Studio 2022 v17.0.0.

# Описание входных и выходных данных

Входные данные:

* **Путь к папке:** Задаётся пользователем через главное меню. Если не задан, используется путь к исполняемому файлу.
* **Два файла, расположенные в указанной папке:**
  + **Файл исходящей корреспонденции:** Имя: IC\_<имя>.txt. Содержит данные об исходящей корреспонденции.
  + **Файл адресов организаций:** Имя: AO\_<имя>.txt. Содержит данные об адресах организаций.

Пользователь вводит *только* <имя> (без IC\_/AO\_ и .txt).

Выходные данные (в зависимости от режима):

* **Режим поиска:** Вывод в консоль информации о запрошенной организации.
* **Режим полного вывода:** Вывод в файл Отчет\_<имя\_корр>\_<имя\_адр>[(N)].txt (в той же папке, что и входные файлы) информации обо всех организациях. <имя\_корр> и <имя\_адр> — имена файлов, введённые пользователем (без префиксов и расширений). (N) — порядковый номер, если файл с таким именем уже есть.

# Описание программных единиц

Описание глобальных переменных:

* std::string currentFolderPath: Глобальная переменная, хранящая текущий путь к папке с входными и выходными файлами.

Описание используемых констант:

* Constants::INSTRUCTIONS\_FILE = "instructions.txt": Имя файла, содержащего инструкцию для пользователя.

Описание используемых структур:

* Correspondence: Структура, содержащая информацию об исходящей корреспонденции.
  + std::string type: Вид корреспонденции.
  + std::string date: Дата подготовки корреспонденции.
  + std::string organization: Название организации.
* Address: Структура, содержащая информацию об адресе организации.
  + std::string organization: Название организации.
  + std::string address: Адрес организации.
  + std::string contactPerson: ФИО организатора.

**Описание функций программы**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название функции** | **Назначение функции** | **Параметры функции** | **Возвращаемое значение функции** |
| is\_leap(int year) | Проверяет, является ли год високосным. | int year – год для проверки. | bool – true, если год високосный, иначе false. |
| isValidFio(const std::string& fio) | Проверяет, является ли строка допустимым ФИО. | const std::string& fio – строка с ФИО. | bool – true, если ФИО допустимо, иначе false. |
| isValidDate(const std::string& dateStr) | Проверяет, является ли строка допустимой датой в формате ДД.ММ.ГГГГ или строкой "нет данных". | const std::string& dateStr – строка с датой. | bool – true, если дата допустима, иначе false. |
| isValidFileName(const std::string& fileName) | Проверяет, содержит ли имя файла недопустимые символы. | const std::string& fileName – строка с именем файла. | bool – true, если имя файла допустимо, иначе false. |
| readInstructionsFromFile(const std::string& filename) | Отображает инструкцию по использованию программы (из файла). | const std::string& filename – имя файла с инструкцией. | void |
| processOrganization(const std::string& orgName, const std::string& corrFilename, const std::string& addrFilename, bool selectiveOutput, std::ofstream\* outfile, std::set<std::string>& printedOrganizations, std::vector<std::string>& outputBuffer) | Обрабатывает информацию об организации, находя и выводя данные из файлов корреспонденции и адресов в зависимости от режима вывода. | const std::string& orgName – название организации.<br>const std::string& corrFilename – имя файла корреспонденции.<br>const std::string& addrFilename – имя файла адресов.<br>bool selectiveOutput – режим вывода (true - поиск, false - полный вывод).<br>std::ofstream\* outfile – указатель на файл для вывода (только для полного вывода).<br>std::set<std::string>& printedOrganizations - множество уже обработанных организаций, <br>std::vector<std::string>& outputBuffer - Буфер для вывода результата поиска. | void |
| getFilenamesFromUser(const std::string& folderPath) | Запрашивает у пользователя имена файлов корреспонденции и адресов. | const std::string& folderPath – путь к папке. | std::pair<std::string, std::string> – пара строк: полное имя файла корреспонденции и полное имя файла адресов. |
| runProgram(const std::string& folderPath, const std::string& correspondenceFilename, const std::string& addressesFilename, std::string& outputFilename) | Запускает основной процесс программы: выбор режима вывода, обработка данных. | const std::string& folderPath – путь к папке.<br>const std::string& correspondenceFilename – имя файла корреспонденции.<br>const std::string& addressesFilename – имя файла адресов.<br> std::string& outputFilename | void |
| menu() | Отображает главное меню программы и управляет её основным циклом. | (нет) | void |
| normalizePath(const std::string& path) | Преобразует путь к абсолютному. | const std::string& path - путь | std::string - абсолютный путь |
| getLineWithEsc(const std::string& instruction) | Получает строку ввода от пользователя с обработкой Esc и Backspace. | const std::string& instruction - строка | std::string - введенная строка |

# Инструкция пользователю

1. **Запуск программы:**
   * Запустите исполняемый файл программы.
2. **Главное меню:**
   * **Выбор пути к папке (необязательно):**
     + По умолчанию программа использует папку, где находится исполняемый файл.
     + Чтобы изменить папку, выберите пункт "2. Изменить путь к папке".
     + Введите путь к папке, в которой находятся файлы IC\_\*.txt и AO\_\*.txt, и нажмите Enter.
     + Если путь не указан, программа будет использовать текущий.
   * **Начало работы:**
     + Выберите пункт "1. Начать работу".
3. **Ввод имён файлов:**
   * Программа запросит имена файлов (без префиксов IC\_/AO\_ и расширения .txt):
     + Введите имя файла с исходящей корреспонденцией.
     + Нажмите Enter.
     + Введите имя файла с адресами организаций.
     + Нажмите Enter.
   * Убедитесь, что файлы с указанными именами (с добавленными префиксами и расширением) существуют в выбранной папке.
4. **Выбор режима вывода:**
   * **1. Поиск по компании и вывод в консоль:**
     + Введите название организации для поиска.
     + Нажмите Enter.
     + Результаты поиска будут выведены на консоль.
   * **2. Полный вывод в файл:**
     + Программа создаст файл отчёта Отчет\_<имя\_корр>\_<имя\_адр>[(N)].txt в той же папке, где находятся входные файлы.
       - <имя\_корр> - имя файла корреспонденции, введённое пользователем (без префикса и расширения).
       - <имя\_адр> - имя файла адресов, введённое пользователем (без префикса и расширения).
       - (N) - порядковый номер, добавляемый, если файл с таким именем уже существует.
     + В файл будет записана информация обо всех организациях из файла адресов, а также связанная с ними корреспонденция из файла корреспонденции.
   * **Esc - вернуться в главное меню:** Нажмите Esc, чтобы вернуться в главное меню из режима выбора вывода.
5. **Выход:**

* Чтобы выйти из программы нажмите Esc в главном меню.

Так же программа дополнительно сопровождается инструкцией по умолчанию, доступной из главного меню по нажатию клавиши “3”:

Программа для обработки корреспонденции и адресов организаций и вывода обобщенной информации.

Соблюдайте требования для корректной работы:

Путь к папке: Указывайте только существующие пути к папкам.

Имена файлов: Используйте латинские буквы или кириллицу, цифры,

символы подчеркивания (\_) и дефиса (-). Не используйте иные специальные символы.

Искомые файлы должны существовать.

-----Порядок работы-----

Перед началом работы: Если путь к папке не совпадает с необходимым, укажите верный.

Главное меню:

"Начать работу": Переход к обработке файлов.

"Изменить путь к папке": Ввод пути к папке, где располагаются файлы с данными.

"Инструкция": Отображение данного руководства.

"Выход": Завершение работы программы.

Процесс обработки данных:

После выбора пункта "Начать работу" программа последовательно запросит:

Название файла исходящей корреспонденции (без префикса IC\_ и расширения .txt).

Название файла адресов организаций (без префикса AO\_ и расширения .txt).

Выбор режима вывода данных:

Поиск по организации и вывод в консоль: Требуется ввод названия организации с учетом региста,

для которой необходимо отобразить информацию.

Полный вывод в файд: Информация по всем организациям будет записана в файл

"Отчет\_название файла исходящей корреспонденции\_название файла адресов организаций.txt".

Esc: Возврат в главное меню.

-----Результат-----

Поиск по организации и вывод в консоль: Информация отображается непосредственно в окне консоли.

Полный вывод в файл: Информация сохраняется в каталоге, указанном в пути в текстовый файл с именем

"Отчет\_название файла исходящей корреспонденции\_название файла адресов организаций.txt".

Путь к данному файлу будет выведет на экран.

# Листинг программной разработки

Главный файл

// Lab\_prog.cpp

#include "Lab\_prog\_2.h"

#include <iostream>

#include <windows.h>

#include <filesystem>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

system("mode con cols=120 lines=30");

// Запуск главного меню программы

menu();

return 0;

}

Файл заголовка

// Lab\_prog\_2.h

#pragma once

#include <algorithm>

#include <cctype>

#include <conio.h>

#include <fstream>

#include <iostream>

#include <limits>

#include <locale>

#include <map>

#include <set>

#include <sstream>

#include <string>

#include <sys/stat.h>

#include <vector>

#include <Windows.h>

#include <filesystem>

#ifdef max // Нужно для Visual Studio, чтобы не было конфликтов с <limits>

#undef max

#endif

// Глобальная переменная для хранения текущего пути к папке с данными.

extern std::string currentFolderPath;

// Объявление констант.

namespace Constants {

const std::string INSTRUCTIONS\_FILE = "instructions.txt"; // Имя файла с инструкциями.

}

// Структура для хранения данных о корреспонденции.

struct Correspondence {

std::string type; // Вид корреспонденции.

std::string date; // Дата корреспонденции.

std::string organization; // Название организации.

};

// Структура для хранения данных об адресе организации.

struct Address {

std::string organization; // Название организации.

std::string address; // Адрес организации.

std::string contactPerson; // ФИО организатора.

};

/\*\* @brief Проверяет, является ли год високосным. \*/

bool is\_leap(int year);

/\*\* @brief Проверяет допустимость ФИО. \*/

bool isValidFio(const std::string& fio);

/\*\* @brief Проверяет допустимость формата даты. \*/

bool isValidDate(const std::string& dateStr);

/\*\* @brief Проверяет допустимость имени файла. \*/

bool isValidFileName(const std::string& fileName);

/\*\* @brief Читает инструкции из файла. \*/

void readInstructionsFromFile(const std::string& filename);

/\*\* @brief Обрабатывает информацию об организации.

\* Находит и выводит данные организации из файлов корреспонденции и адресов.

\*/

void processOrganization(const std::string& orgName, const std::string& corrFilename,

const std::string& addrFilename, bool selectiveOutput, std::ofstream\* outfile,

std::set<std::string>& printedOrganizations, std::vector<std::string>& outputBuffer);

/\*\* @brief Запрашивает имена файлов у пользователя. \*/

std::pair<std::string, std::string> getFilenamesFromUser(const std::string& folderPath);

/\*\* @brief Запускает основной процесс программы. \*/

void runProgram(const std::string& folderPath, const std::string& correspondenceFilename,

const std::string& addressesFilename, std::string& outputFilename);

/\*\* @brief Отображает главное меню. \*/

void menu();

/\*\* @brief Нормализует путь к файлу, делая его абсолютным. \*/

std::string normalizePath(const std::string& path);

/\*\* @brief Выполняет чтение строки из консоли с поддержкой Esc и Backspace, а также фильтрацией.

\* @param instruction Сообщение, отображаемое пользователю перед вводом.

\* @return Введенная строка, или пустая строка, если был нажат Esc.

\*/

std::string getLineWithEsc(const std::string& instruction);

Файл реализации библиотеки

//Lab\_prog\_2.cpp

#include "Lab\_prog\_2.h"

#include <algorithm>

#include <fstream>

#include <iostream>

#include <limits>

#include <sstream>

#include <vector>

#include <iomanip>

#include <filesystem>

#include <conio.h>

using namespace std;

// ОПРЕДЕЛЕНИЕ глобальной переменной

string currentFolderPath = "";

string getLineWithEsc(const string& instruction) {

string input;

cout << instruction;

while (true) {

int key = \_getch();

char ch = static\_cast<char>(key);

if (key == 27) { // ESC

cin.clear();

input = "";

cout << endl;

return input;

}

else if (key == '\r') { // Enter

cout << endl;

return input;

}

else if (key == 8) { // Backspace

if (!input.empty()) {

input.pop\_back();

cout << "\b \b";

}

}

// Разрешаем печатные символы (включая кавычки)

else if (ch >= 32 && ch <= 126) {

input += ch;

cout << ch;

}

// Кириллица

else if ((ch >= 'а' && ch <= 'я') || (ch >= 'А' && ch <= 'Я'))

{

input += ch;

cout << ch;

}

}

}

void readInstructionsFromFile(const string& filename) {

system("mode con cols=150 lines=36");

ifstream file(filename);

if (file.is\_open()) {

string line;

while (getline(file, line))

cout << line << endl;

file.close();

}

else {

cerr << "Не удалось открыть файл инструкции: " << filename << endl;

cerr << "Пожалуйста, поместите файл instructions.txt в ту же папку, где находится исполняемый файл программы.\n";

}

cout << "\nНажмите любую клавишу для возврата в главное меню...\n";

\_getch();

system("mode con cols=120 lines=30");

}

void processOrganization(const string& orgName, const string& corrFilename, const string& addrFilename, bool selectiveOutput, ofstream\* outfile, set<string>& printedOrganizations, vector<string>& outputBuffer) {

ifstream corrFile(corrFilename);

ifstream addrFile(addrFilename);

vector<Address> addresses;

bool orgFoundInAddress = false;

auto writeToBuffer = [&](const string& str) {

if (selectiveOutput) {

outputBuffer.push\_back(str); // Для консольного вывода

}

else if (outfile) {

\*outfile << str << endl; // Для файлового вывода

}

};

// Сначала ищем организацию в файле адресов. Это нужно для обоих режимов вывода.

if (addrFile.is\_open()) {

string line;

while (getline(addrFile, line)) {

stringstream ss(line);

string currentOrg, address, contactPerson;

getline(ss, currentOrg, ';');

getline(ss, address, ';');

getline(ss, contactPerson, ';');

currentOrg.erase(0, currentOrg.find\_first\_not\_of(" \t")); //Удаляем пробелы

currentOrg.erase(currentOrg.find\_last\_not\_of(" \t") + 1);

if (currentOrg == orgName) {

addresses.push\_back({ currentOrg, address, contactPerson });

orgFoundInAddress = true;

}

}

addrFile.close();

}

else {

writeToBuffer("Не удалось открыть файл адресов: " + addrFilename);

return; // Если не удалось открыть файл адресов, дальше не идём

}

if (!orgFoundInAddress && selectiveOutput) {

writeToBuffer("Организация \"" + orgName + "\" не найдена в файле адресов.");

return;

}

//Если это полный вывод, выводим данные для каждой записи

if (!selectiveOutput)

{

if (printedOrganizations.count(orgName)) {

return; // Организация уже была обработана

}

printedOrganizations.insert(orgName); // Помечаем организацию как обработанную

// Вывод информации об организации (из файла адресов)

writeToBuffer("Организация: " + orgName);

for (const auto& addr : addresses) {

writeToBuffer("Адрес: " + addr.address);

writeToBuffer("ФИО Организатора: " + addr.contactPerson);

}

// Поиск и вывод корреспонденции для организации

if (corrFile.is\_open()) {

string line;

bool foundCorrespondence = false;

while (getline(corrFile, line)) {

stringstream ss(line);

string type, date, currentOrg;

getline(ss, type, ';');

getline(ss, date, ';');

getline(ss, currentOrg, ';');

currentOrg.erase(0, currentOrg.find\_first\_not\_of(" \t")); //Удаляем пробелы

currentOrg.erase(currentOrg.find\_last\_not\_of(" \t") + 1);

if (currentOrg == orgName) {

foundCorrespondence = true;

writeToBuffer("\tВид корреспонденции: " + type);

writeToBuffer("\tДата: " + date);

}

}

if (!foundCorrespondence) {

writeToBuffer("\tКорреспонденция не найдена."); // Для полного вывода

}

corrFile.close();

}

else {

writeToBuffer("Не удалось открыть файл корреспонденции: " + corrFilename);

}

writeToBuffer("--------------------");

}

//Если это поиск

if (selectiveOutput && orgFoundInAddress)

{

writeToBuffer("Организация: " + orgName);

if (!addresses.empty()) {

writeToBuffer("Адрес: " + addresses[0].address); // Выводим первый адрес

writeToBuffer("ФИО организатора: " + addresses[0].contactPerson); // Первое ФИО организатора

if (addresses.size() > 1) {

writeToBuffer("\nДругие найденные данные компании:\n");

for (size\_t i = 1; i < addresses.size(); ++i) {

writeToBuffer("Адрес: " + addresses[i].address);

writeToBuffer("ФИО организатора: " + addresses[i].contactPerson);

}

}

}

if (corrFile.is\_open()) {

string line;

bool foundCorrespondence = false;

while (getline(corrFile, line)) {

stringstream ss(line);

string type, date, currentOrg;

getline(ss, type, ';');

getline(ss, date, ';');

getline(ss, currentOrg, ';');

currentOrg.erase(0, currentOrg.find\_first\_not\_of(" \t")); //Удаляем пробелы

currentOrg.erase(currentOrg.find\_last\_not\_of(" \t") + 1); //

if (currentOrg == orgName) {

foundCorrespondence = true;

writeToBuffer("\tВид корреспонденции: " + type);

writeToBuffer("\tДата: " + date);

}

}

if (!foundCorrespondence) {

writeToBuffer("\tКорреспонденция не найдена."); // Сообщение, если не найдено

}

corrFile.close();

}

else {

writeToBuffer("Не удалось открыть файл корреспонденции: " + corrFilename);

}

writeToBuffer("--------------------"); // Добавляем разделитель

}

}

pair<string, string> getFilenamesFromUser(const string& folderPath) {

string correspondenceFilename, addressesFilename;

while (true) {

correspondenceFilename = getLineWithEsc("Введите имя файла с исходящей корреспонденцией (префикса IC\_ и без расширения .txt) или нажмите Esc для отмены: ");

if (correspondenceFilename.empty()) {

return { "", "" };

}

string fullCorrFilename = folderPath + "IC\_" + correspondenceFilename + ".txt";

if (!filesystem::exists(fullCorrFilename)) {

cerr << "Ошибка: Файл " << fullCorrFilename << " не существует.\n";

cout << "Пожалуйста, попробуйте еще раз.\n";

continue;

}

break;

}

while (true) {

addressesFilename = getLineWithEsc("Введите имя файла с адресами организаций (префикса AO\_ и без расширения .txt) или нажмите Esc для отмены: ");

if (addressesFilename.empty()) {

return { "", "" };

}

string fullAddressesFilename = folderPath + "AO\_" + addressesFilename + ".txt";

if (!filesystem::exists(fullAddressesFilename)) {

cerr << "Ошибка: Файл " << fullAddressesFilename << " не существует.\n";

cout << "Пожалуйста, попробуйте еще раз.\n";

continue;

}

break;

}

return { folderPath + "IC\_" + correspondenceFilename + ".txt",

folderPath + "AO\_" + addressesFilename + ".txt" };

}

void runProgram(const string& folderPath, const string& correspondenceFilename,

const string& addressesFilename, string& outputFilename) {

while (true) {

cout << "\nВыберите режим вывода:\n";

cout << "1. Поиск по компании и вывод в консоль\n";

cout << "2. Полный вывод в файл\n";

cout << "Esc - вернуться в главное меню\n";

cout << "\nВаш выбор: ";

char choice = \_getch();

char choiceChar = static\_cast<char>(choice);

while (choiceChar != '1' && choiceChar != '2' && choice != 27) {

choice = \_getch();

choiceChar = static\_cast<char>(choice);

}

cout << endl;

switch (choice) {

case '1': {

string orgName = getLineWithEsc("Введите название организации для поиска или нажмите Esc для отмены: ");

if (orgName.empty()) {

break;

}

ofstream\* outfile = nullptr; // Указатель на ofstream (не создаём объект!)

set<string> printedOrganizations;

vector<string> outputBuffer; // Буфер для вывода

processOrganization(orgName, correspondenceFilename, addressesFilename,

true, outfile, printedOrganizations, outputBuffer);

// Вывод буфера на консоль

for (const string& line : outputBuffer) {

cout << line << endl;

}

break;

}

case '2': {

// Получаем имена файлов без путей и расширений для формирования имени выходного файла

string corrFileNameOnly = filesystem::path(correspondenceFilename).filename().string();

string addrFileNameOnly = filesystem::path(addressesFilename).filename().string();

size\_t pos = corrFileNameOnly.rfind(".txt");

if (pos != string::npos) {

corrFileNameOnly = corrFileNameOnly.substr(0, pos);

}

pos = addrFileNameOnly.rfind(".txt");

if (pos != string::npos) {

addrFileNameOnly = addrFileNameOnly.substr(0, pos);

}

// Формируем имя выходного файла (с путём)

string baseFilename = folderPath + "Отчет\_" + corrFileNameOnly + "\_" + addrFileNameOnly;

string filename = baseFilename + ".txt";

int i = 1;

while (filesystem::exists(filename)) { // Проверяем, существует ли файл

filename = baseFilename + "(" + to\_string(i++) + ")" + ".txt"; // Добавляем (1), (2) и т.д.

}

try { //Добавил обработку исключений

ofstream outfile(filename); // Создаём ofstream \*после\* формирования имени файла

if (!outfile.is\_open()) {

cerr << "Ошибка: не удалось открыть файл для записи.\n";

break;

}

set<string> printedOrganizations; // Множество для отслеживания уже выведенных организаций

vector<string> outputBuffer; //Вектор для буферизации вывода

ifstream addrFile(addressesFilename); // Открываем файл адресов

if (addrFile.is\_open()) {

string line;

while (getline(addrFile, line)) {

stringstream ss(line);

string orgName, address, contactPerson;

getline(ss, orgName, ';');

getline(ss, address, ';');

getline(ss, contactPerson, ';');

orgName.erase(0, orgName.find\_first\_not\_of(" \t")); //Удаляем пробелы

orgName.erase(orgName.find\_last\_not\_of(" \t") + 1);

//Если орагнизация не встречалась, то выводим о ней информацию

if (printedOrganizations.find(orgName) == printedOrganizations.end()) {

processOrganization(orgName, correspondenceFilename, addressesFilename, false, &outfile, printedOrganizations, outputBuffer);

}

}

addrFile.close();

}

else {

cerr << "Ошибка: не удалось открыть файл адресов: " << addressesFilename << ".\n";

break;

}

outfile.close();

cout << "Информация успешно записана в файл " << filename << ".\n";

break;

}

catch (const exception& e)

{

cerr << "Ошибка: не удалось создать или записать файл.\n";

cerr << e.what() << endl;

break;

}

}

case 27:

return; // Выход из runProgram

}

}

}

void menu() {

string correspondenceFilename, addressesFilename, outputFilename;

string escPressed;

bool folderPathSet = true; // Флаг, что путь к папке установлен

currentFolderPath = normalizePath(filesystem::current\_path().string()) + "\\"; // Сразу нормализуем

const string instructionsFile = currentFolderPath + "instructions.txt";

cout << "\nПрограмма для обработки данных о корреспонденции и адресах организаций.\n";

cout << "Она позволяет искать информацию об организациях и их корреспонденции, а также формировать отчеты.\n";

while (true) {

system("cls"); // Очищаем экран

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\* Программа для обработки данных корреспонденции и адресов \*\n";

cout << "\* Программа позволяет искать и обрабатывать информацию об \*\n";

cout << "\* организациях на основе двух файлов - \*\n";

cout << "\* Исходящей корреспонденции (IC\_\*\*\*.txt) \*\n";

cout << "\* Адресов организаций (AO\_\*\*\*.txt) \*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

cout << "Главное меню:\n";

cout << "1. Начать работу\n";

cout << "2. Изменить путь к папке\n";

cout << "3. Инструкция\n\n";

cout << "Текущий путь: " << currentFolderPath << "\n\n";

cout << "Esc - выход\n\n";

int mainChoice = \_getch(); // Считываем символ

char mainChoiceChar = static\_cast<char>(mainChoice); // char

switch (mainChoice) {

case '1': { // "Начать работу"

// Получаем имена файлов от пользователя

pair<string, string> filenames = getFilenamesFromUser(currentFolderPath);

if (filenames.first.empty() && filenames.second.empty()) {

break; // Если пользователь нажал Esc, выходим из case '1'

}

correspondenceFilename = filenames.first;

addressesFilename = filenames.second;

runProgram(currentFolderPath, correspondenceFilename, addressesFilename, outputFilename);

break;

}

case '2': { // "Изменить путь к папке"

string newPath = getLineWithEsc("Введите новый путь к папке: ");

if (newPath.empty()) break;

newPath = normalizePath(newPath); // Нормализуем путь

// Проверяем, существует ли папка и является ли она директорией

struct stat buffer;

if (stat(newPath.c\_str(), &buffer) == 0 && (buffer.st\_mode & S\_IFDIR)) {

currentFolderPath = newPath + "\\"; // Добавляем разделитель

cout << "Путь успешно изменен.\n";

}

else {

cerr << "Ошибка: указанный путь не существует или не доступен.\n";

}

cout << "Нажмите любую клавишу для продолжения...\n"; // Добавлено сообщение

\_getch(); // Ждём нажатия клавиши

break;

}

case '3': // Инструкция

readInstructionsFromFile(instructionsFile);

break;

case 27: // Esc - выход

return;

default:

continue; // Некорректный ввод - продолжаем цикл

}

}

}

string normalizePath(const string& path) {

namespace fs = std::filesystem;

fs::path normalizedPath = fs::absolute(path); // Получаем абсолютный путь

return normalizedPath.string(); // Возвращаем как строку

}

# Результаты тестирования программной разработки

При запуске программы выводится главное меню (рис. 1). Для нажатия доступны только цифры 1, 2, 3, esc. Остальные символы при нажатии игнорируются.

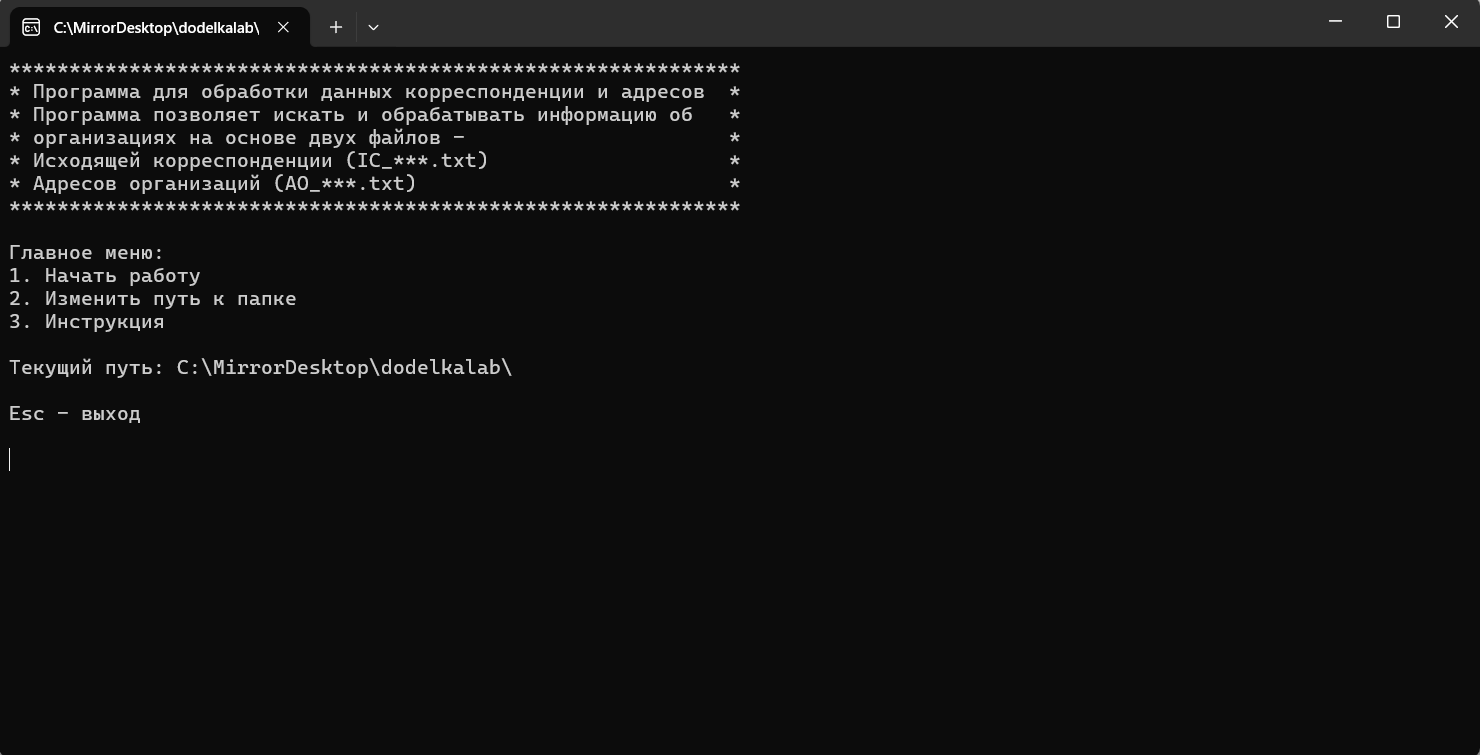


Рисунок Главное меню.

**Пример работы программы при выборе пути к папке хранения (Нажатие “2” в главном меню):**

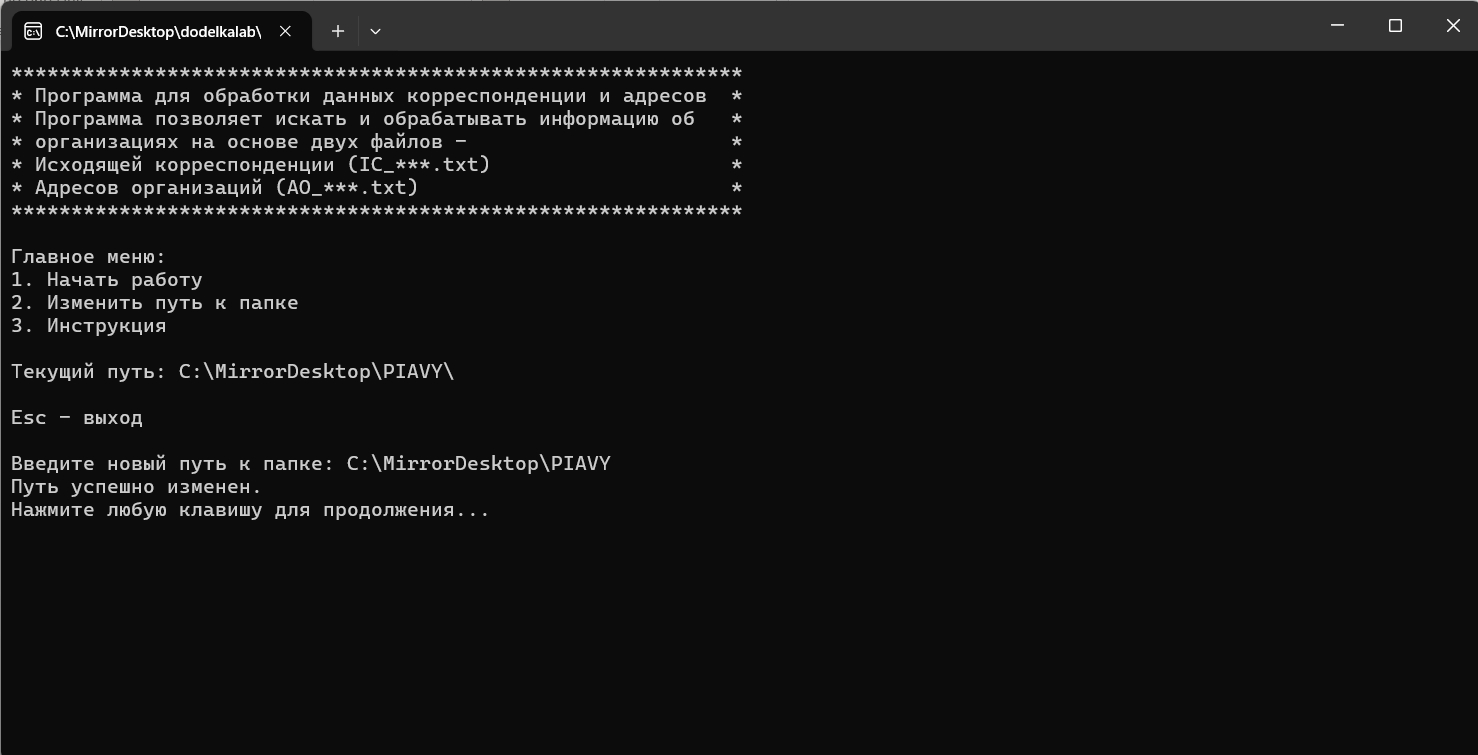
****

Рисунок Корректный ввод пути к папке хранения файлов.

При корректном вводе выбранный путь будет далее выведен на последующих этапах работы для удобства пользователя, при некорректном вводе программа запрашивает у пользователя повторный ввод и очищает вывод выбранного пути к папке(рис. 3).

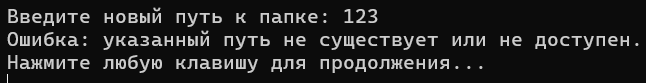


Рисунок Некорректный ввод пути к папке хранения файлов.

**Пример работы программы при открытии иснструкции (нажатие “3” в главном меню):**

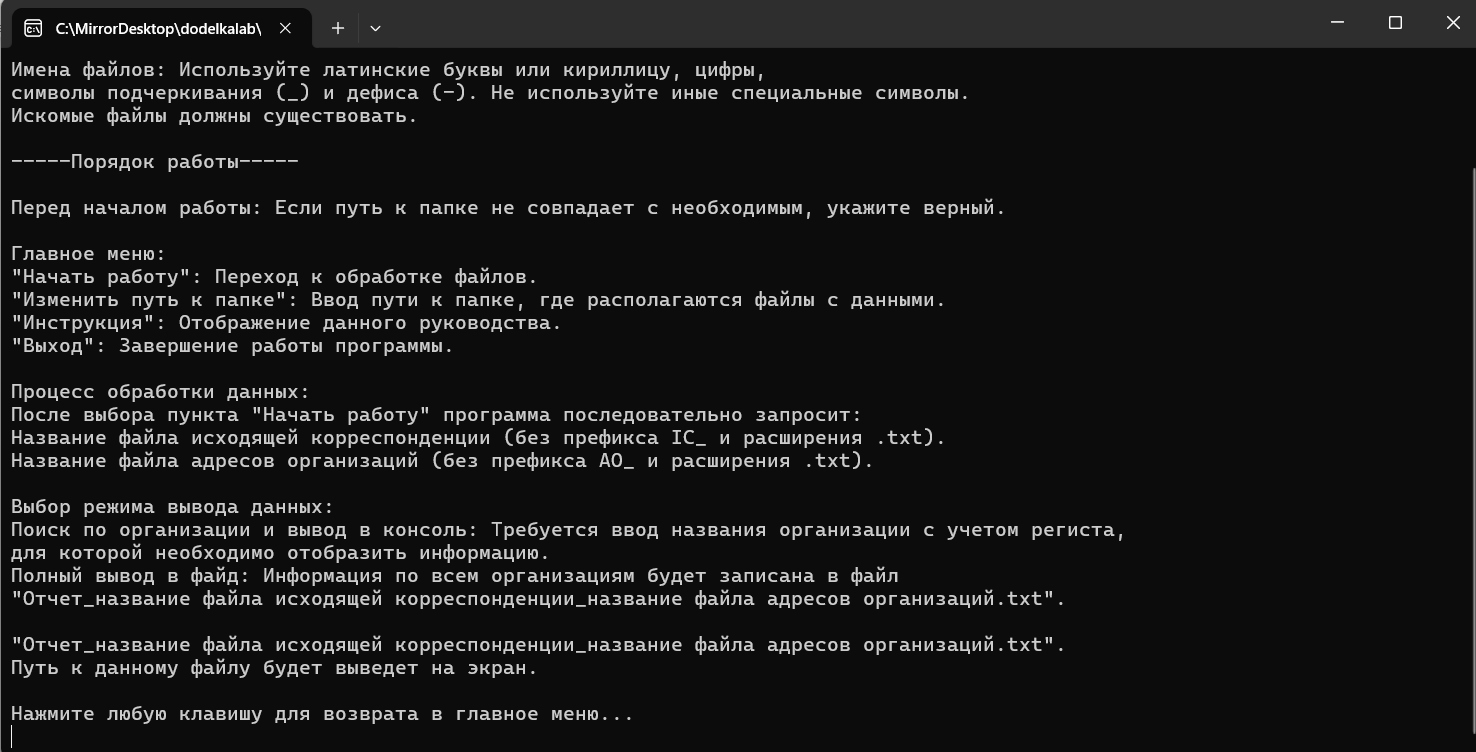


Рисунок При выборе инструкции.

**Пример работы программы по записи данных в файлы (нажатие “1” в главном меню):**

При выборе начать работу программа запрашивает название файлов без префиксов и расширений (рис. 5). Далее программа запрашивает какой тип вывода необходим (рис. 6). При выборе избирательного выбора пользователь должен ввести название организации, по которой он хочет совершить поиск. Программа выведет доступную информацию в консоль (рис. 7). При выборе полного вывода в файл программа запивсывает информацию в файл Отчет\_<имя\_корр>\_<имя\_адр>[(N)].txt и выводит путь до него (рис. 8).

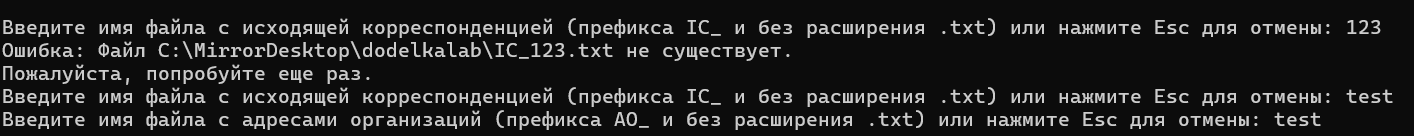


Рисунок Пример заполнения имен файлов.

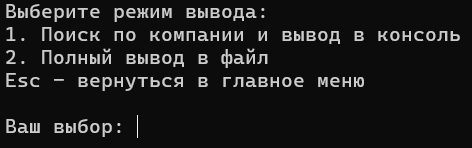


Рисунок Выбор режима вывода.

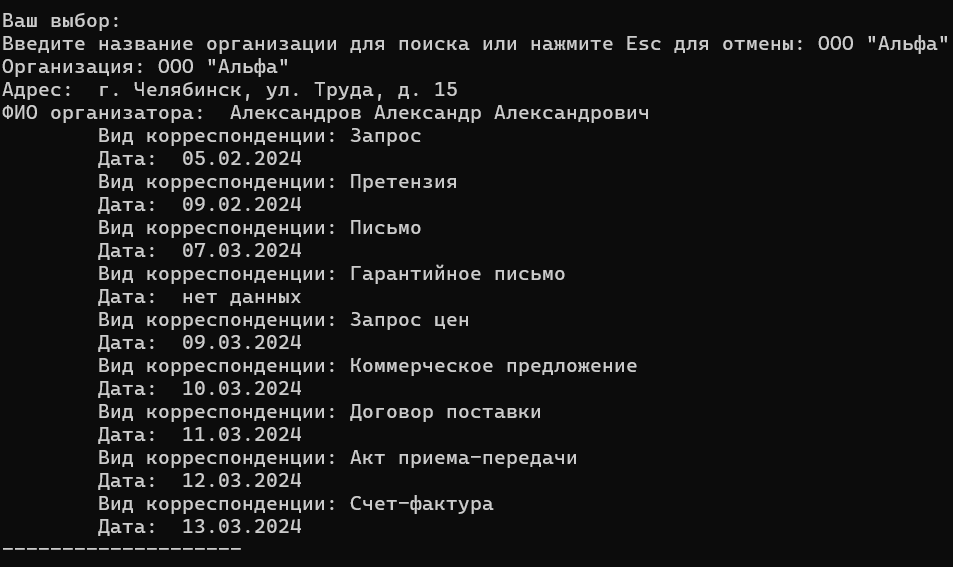


Рисунок 7 Пример вывода в консоль избирательного вывода.

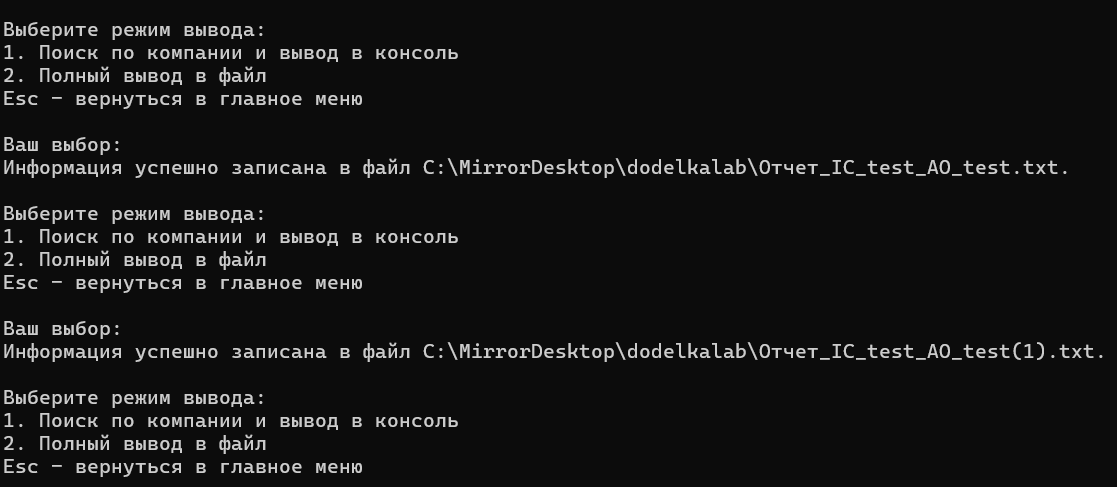


Рисунок 8 Пример использования полного вывода, с повтором, в файл.

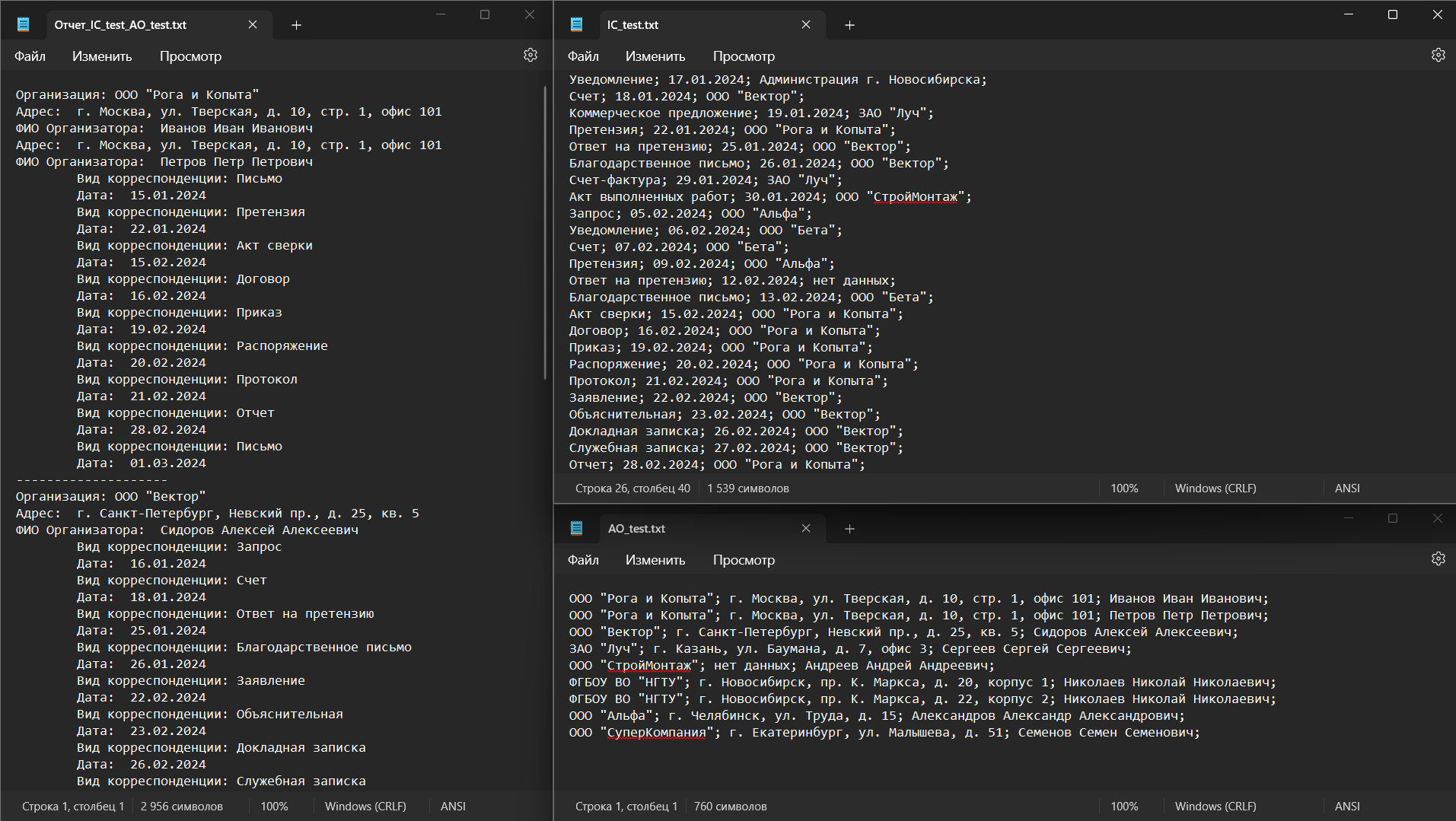


Рисунок 9 Пример преобразования входных файлов (слева) в выходной файл (справа).