**«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](#_Toc194048774)

[2. Псевдокод 3](#_Toc194048775)

[3. Сведения о программной реализации 8](#_Toc194048776)

[3.1. Язык программирования и среда разработки 8](#_Toc194048777)

[3.2. Описание входных и выходных данных 8](#_Toc194048778)

[3.3. Описание программных единиц 8](#_Toc194048779)

[4. Инструкция пользователю 11](#_Toc194048780)

[5. Листинг программной разработки 14](#_Toc194048781)

[6. Результаты тестирования программной разработки 36](#_Toc194048782)

# 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

Подготовка: Программа настраивает консоль для русского языка и определяет папку для работы (изначально – текущая папка).

Главное меню: Показывает пользователю опции:

Начать работу: Перейти к выбору файлов и действиям с ними.

Выбрать папку: Позволяет указать другую папку для файлов данных. Программа проверит, существует ли папка и можно ли в ней работать.

Инструкция: Показывает текст из файла инструкций.

Выход: Завершает программу.

Выбор файлов (если выбрано "Начать работу"):

Программа спрашивает часть имени файла с корреспонденцией и часть имени файла с адресами.

Она проверяет, существуют ли в выбранной папке файлы с нужными префиксами (IC\_ и AO\_) и введенными именами. Если нет – просит ввести снова.

Если файлы найдены, переходит к выбору действия.

Выбор действия (после выбора файлов): Показывает меню:

Поиск данных: Найти и показать информацию по определенному критерию.

Создать полный отчет: Собрать информацию по всем организациям в один файл.

Назад: Вернуться в главное меню.

Поиск данных (если выбран):

Спрашивает, по какому полю искать (Название организации, Адрес, Руководитель, Вид корреспонденции, Дата).

Спрашивает, какой текст искать в этом поле.

Находит все организации, у которых указанное поле соответствует поисковому тексту (для адреса – содержит текст, для остальных – точное совпадение).

Для каждой найденной организации собирает *всю* информацию о ней (все адреса, всю корреспонденцию) из обоих исходных файлов.

Выводит результаты в консоль:

Если искали по *Виду* или *Дате*, сначала показывает строки, где было точное совпадение, а затем полную информацию по связанным организациям.

Если искали по *Адресу* или *Руководителю*, сразу показывает полную информацию по найденным организациям.

Если искали по *Названию*, просто показывает полную информацию по этой одной организации.

Если ничего не найдено – сообщает об этом.

Создание полного отчета (если выбрано):

Определяет имя для нового файла-отчета в рабочей папке (добавляя номер, если файл уже есть).

Собирает список *всех уникальных* названий организаций, упомянутых *хотя бы в одном* из исходных файлов (корреспонденции или адресов).

Сортирует этот список по алфавиту.

Создает файл отчета.

Для каждой организации из отсортированного списка: находит *всю* информацию о ней (все адреса, всю корреспонденцию) из обоих исходных файлов и записывает в файл отчета. Если при этом какой-то из исходных файлов недоступен, в отчет пишется сообщение об ошибке.

Сообщает пользователю имя созданного файла отчета.

Возврат: После выполнения поиска или создания отчета программа возвращается к меню выбора действия (Поиск/Отчет/Назад). При выходе из этого меню – возвращается в главное меню.

# 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.
* **Название организации если пользователь выбрал поиск по ней**

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: Фамилия руководителя.

**Описание функций программы**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название функции** | **Назначение функции** | **Параметры функции** | **Возвращаемое значение функции** |
| isvalidfilenamechar(c) | Проверяет, допустим ли символ c в имени файла. | char c – символ для проверки. | boolean – истина, если символ допустим, иначе ложь |
| isvalidpathchar(c) | Проверяет, допустим ли символ c в пути. | char c – символ для проверки. | boolean – истина, если символ допустим, иначе ложь |
| isvalidorgnamechar(c) | Проверяет, допустим ли символ c в названии организации. | char c – символ для проверки. | boolean – истина, если символ допустим, иначе ложь |
| getlinewithrestrictedchars(instruction, buffer, buffer\_size, isvalidcharfunc) | Получает строку ввода от пользователя, фильтруя недопустимые символы, обрабатывает ESC, Enter, Backspace, Ctrl+V. | строка instruction – приглашение для ввода.<br>массив символов buffer – буфер для ввода.<br>целое число buffer\_size – размер буфера.<br>указатель на функцию isvalidcharfunc – функция проверки символа. | нет |
| normalizepath(path, normalized\_path, normalized\_path\_size) | Преобразует путь к абсолютному, удаляет лишние слеши, добавляет завершающий, если нужно. | строка path – исходный путь.<br>массив символов normalized\_path – буфер для результата.<br>целое число normalized\_path\_size – размер буфера. | нет |
| selectfolderpath() | Запрашивает у пользователя путь к папке, проверяет его существование и права доступа. | (нет) | нет |
| readinstructionsfromfile(filename) | Отображает инструкцию из файла filename на экране. | строка filename – имя файла с инструкцией. | нет |
| processorganization(orgname, corrfilename, addrfilename, selectiveoutput, outfile, printedorganizations, outputbuffer) | Обрабатывает информацию об организации, ищет и выводит данные. | строка orgname – название организации.<br>строка corrfilename – файл корреспонденции.<br>строка addrfilename – файл адресов.<br>boolean selectiveoutput – режим вывода (поиск/отчет).<br>указатель на файл outfile – файл для отчета (или null).<br>множество printedorganizations – список уже обработанных организаций (для отчета).<br>список строк outputbuffer – список для результатов поиска. | нет |
| getfilenamesfromuser(folderpath) | Запрашивает у пользователя имена файлов корреспонденции и адресов (базовые имена). | строка folderpath – путь к папке. | пара строк – полное имя файла корреспонденции и адресов. |
| runprogram(folderpath, corrfilename, addrfilename, outputfilename) | Запускает основной процесс: выбор режима (поиск/отчет), обработка данных. | строка folderpath – путь к папке.<br>строка corrfilename – файл корреспонденции.<br>строка addrfilename – файл адресов.<br>строка outputfilename – имя файла отчета (используется ли?). | нет |
| menu() | Отображает главное меню, управляет основным циклом программы. | (нет) | нет |

# 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);

// Запуск главного меню программы

menu();

return 0;

}

Файл заголовка

//Lab\_prog\_2.h

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <set>

#include <sstream>

#include <algorithm>

#include <filesystem> // Для filesystem::exists и filesystem::path

#include <cstdio>

#include <cstring>

#include <conio.h> // Для \_getch, \_kbhit

#include <windows.h> // Для SetConsoleCP, SetConsoleOutputCP, OpenClipboard, GetClipboardData, GlobalLock, GlobalUnlock, CloseClipboard

#include <direct.h> // Для \_getcwd

#include <sys/stat.h> // Для stat

#include <cstdlib> // Для \_fullpath

#include <cctype> // Для isdigit, tolower

#ifdef max

#undef max

#endif

extern char folder\_way[256]; // Путь к папке (как из первой программы)

extern const char\* file\_extension;

extern const char\* ocfe; // Префикс для файлов корреспонденции

extern const char\* oa; // Префикс для файлов адресов

// Макрос для безопасного копирования строк

#define SAFE\_STRCPY(dest, src, size) strcpy\_s(dest, size, src)

#ifndef MAX\_PATH

#define MAX\_PATH 260

#endif

namespace Constants {

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

}

struct Correspondence {

std::string type;

std::string date;

std::string organization;

};

struct Address {

std::string organization;

std::string address;

std::string contactPerson;

};

/\*\* @brief Функция проверки символа для имени файла \*/

bool isValidFileNameChar(char c);

/\*\* @brief Функция проверки символа для пути \*/

bool isValidPathChar(char c);

/\*\* @brief Функция для получения строки с консоли с фильтрацией символов \*/

void getLineWithRestrictedChars(const char\* instruction, char\* buffer, int buffer\_size, bool (\*isValidCharFunc)(char));

/\*\* @brief Функция нормализации пути (упрощенная для Windows) \*/

void normalizePath(const char\* path, char\* normalized\_path, size\_t normalized\_path\_size);

/\*\* @brief Функция выбора пути к папке (аналог program\_way) \*/

void selectFolderPath();

// --- Функции из второй программы (основная логика) ---

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

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

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

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 (адаптировано под C-стиль пути) \*/

std::pair<std::string, std::string> getFilenamesFromUser(const char\* folderPath);

/\*\* @brief Запускает основной процесс программы (адаптировано под C-стиль пути) \*/

void runProgram(const char\* folderPath, const std::string& correspondenceFilename,

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

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

void menu();

Файл реализации библиотеки

//Lab\_prog\_2.cpp

#include "Lab\_prog\_2.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

using namespace std;

char folder\_way[256] = { 0 };

const char\* file\_extension = ".txt";

const char\* ocfe = "IC\_";

const char\* oa = "AO\_";

// Функция проверки символа для имени файла

bool isValidFileNameChar(char c) {

unsigned char uc = (unsigned char)c;

return (uc != '\\' && uc != '/' && uc != ':' && uc != '\*' && uc != '?' && uc != '"' && uc != '<' && uc != '>' && uc != '|');

}

// Функция проверки символа для пути

bool isValidPathChar(char c) {

unsigned char uc = (unsigned char)c;

return (uc != '\*' && uc != '?' && uc != '"' && uc != '<' && uc != '>' && uc != '|');

}

// Функция проверки символа для названия организации

bool isValidOrgNameChar(char c) { //

unsigned char uc = (unsigned char)c;

bool is\_letter = (uc >= 'A' && uc <= 'Z') || (uc >= 'a' && uc <= 'z') || (uc >= 192); // Латиница + Кириллица

bool is\_digit = (uc >= '0' && uc <= '9');

bool is\_space = (uc == ' ');

// Точная логика из openFileForAppend для поля "Название организации"

return (is\_letter || is\_digit || is\_space || uc == '-' || uc == '\_' || uc == '.' || uc == ',' || uc == '"' || uc == '(' || uc == ')' || uc == 211 || uc == '+' || uc == '!' || uc == '&' || uc == ':' || uc == 171 || uc == 187 || uc == '#'); // №(211), «(171), »(187)

}

// Функция для получения строки с консоли с фильтрацией символов

// Принимает указатель на функцию проверки isValidCharFunc

void getLineWithRestrictedChars(const char\* instruction, char\* buffer, int buffer\_size, bool (\*isValidCharFunc)(char)) {

printf("%s", instruction);

int i = 0;

buffer[0] = '\0';

while (true) {

int key = \_getch();

if (key == 22) {

if (OpenClipboard(NULL)) {

HANDLE hData = GetClipboardData(CF\_TEXT);

if (hData != NULL) {

char\* pszText = (char\*)GlobalLock(hData);

if (pszText != NULL) {

// Используем переданную функцию isValidCharFunc для проверки вставляемых символов

for (int j = 0; pszText[j] != '\0' && i < buffer\_size - 1; ++j) {

if (isValidCharFunc(pszText[j])) { // Проверка здесь

buffer[i++] = pszText[j];

printf("%c", pszText[j]);

}

}

GlobalUnlock(hData);

}

}

CloseClipboard();

}

}

else if (key == 0 || key == 224) {

if (\_kbhit()) {

\_getch();

continue;

}

}

if (key == 27) {

buffer[0] = '\0';

printf("\n");

return;

}

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

buffer[i] = '\0';

printf("\n");

return;

}

else if (key == 8) {

if (i > 0) {

i--;

printf("\b \b");

buffer[i] = '\0';

}

}

else {

char c = (char)key;

// Используем переданную функцию isValidCharFunc для проверки вводимых символов

if (isValidCharFunc(c)) { // Проверка здесь

if (i < buffer\_size - 1) {

buffer[i] = c;

printf("%c", c);

i++;

}

}

}

}

}

// Функция нормализации пути

void normalizePath(const char\* path, char\* normalized\_path, size\_t normalized\_path\_size) {

if (path[0] == '\0') {

if (\_getcwd(normalized\_path, normalized\_path\_size) == NULL) {

fprintf(stderr, "Не удалось определить текущую папку.\n");

normalized\_path[0] = '\0';

}

size\_t len = strlen(normalized\_path);

if (len > 0 && normalized\_path[len - 1] != '\\' && len < normalized\_path\_size - 1) {

normalized\_path[len] = '\\';

normalized\_path[len + 1] = '\0';

}

return;

}

if (\_fullpath(normalized\_path, path, normalized\_path\_size) == NULL) {

SAFE\_STRCPY(normalized\_path, path, normalized\_path\_size);

}

int j = 0;

bool last\_was\_slash = false;

for (int i = 0; normalized\_path[i] != '\0'; ++i) {

if (normalized\_path[i] == '\\') {

if (!last\_was\_slash) {

normalized\_path[j++] = normalized\_path[i];

}

last\_was\_slash = true;

}

else {

normalized\_path[j++] = normalized\_path[i];

last\_was\_slash = false;

}

}

normalized\_path[j] = '\0';

size\_t len = strlen(normalized\_path);

if (len > 0 && normalized\_path[len - 1] != '\\') {

if (len < normalized\_path\_size - 1) {

normalized\_path[len] = '\\';

normalized\_path[len + 1] = '\0';

}

else {

fprintf(stderr, "Предупреждение: Путь слишком длинный для добавления завершающего '\\'.\n");

}

}

}

// Функция выбора пути к папке

void selectFolderPath() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("\n");

char folder\_way\_new[256] = { 0 };

char normalized\_path\_temp[256];

// Используем isValidPathChar для проверки пути

getLineWithRestrictedChars("Введите путь к папке: ", folder\_way\_new, sizeof(folder\_way\_new), isValidPathChar); // Передача функции

if (folder\_way\_new[0] == '\0') {

return;

}

normalizePath(folder\_way\_new, normalized\_path\_temp, sizeof(normalized\_path\_temp));

struct stat path\_stat;

if (stat(normalized\_path\_temp, &path\_stat) == 0) {

if (path\_stat.st\_mode & S\_IFDIR) {

char test\_file\_path[512];

snprintf(test\_file\_path, sizeof(test\_file\_path), "%stest\_access.tmp", normalized\_path\_temp);

FILE\* test\_file = fopen(test\_file\_path, "w");

if (test\_file) {

fclose(test\_file);

remove(test\_file\_path);

printf("Путь к папке выбран: %s\n", normalized\_path\_temp);

SAFE\_STRCPY(folder\_way, normalized\_path\_temp, sizeof(folder\_way));

}

else {

printf("Нет прав для доступа к указанной папке. Выберите другую папку или обратитесь к администратору.\n");

if (folder\_way[0] != '\0') printf("Используется последний корректный путь: %s\n", folder\_way);

}

}

else {

printf("Указанный путь не является папкой.\n");

if (folder\_way[0] != '\0') printf("Используется последний корректный путь: %s\n", folder\_way);

}

}

else {

size\_t len = strlen(normalized\_path\_temp);

if (len > 0 && normalized\_path\_temp[len - 1] == '\\') {

normalized\_path\_temp[len - 1] = '\0';

if (stat(normalized\_path\_temp, &path\_stat) != 0 || !(path\_stat.st\_mode & S\_IFDIR)) {

printf("Указанный путь не существует или не является папкой. Проверьте правильность написания.\n");

if (folder\_way[0] != '\0') printf("Используется последний корректный путь: %s\n", folder\_way);

}

else {

normalized\_path\_temp[len - 1] = '\\'; // Возвращаем слеш обратно

printf("Путь к папке выбран: %s\n", normalized\_path\_temp);

SAFE\_STRCPY(folder\_way, normalized\_path\_temp, sizeof(folder\_way));

}

}

else {

printf("Указанный путь не существует. Проверьте правильность написания.\n");

if (folder\_way[0] != '\0') printf("Используется последний корректный путь: %s\n", folder\_way);

}

}

cout << "Нажмите любую клавишу для продолжения..." << endl; \_getch();

}

// readInstructionsFromFile

void readInstructionsFromFile(const string& filename) {

system("cls");

FILE\* file = fopen("instructions.txt", "r");

if (file) {

char line[256];

while (fgets(line, sizeof(line), file)) {

printf("%s", line);

}

fclose(file);

}

else {

printf("Не удалось открыть файл с инструкцией. Убедитесь, что файл 'instructions.txt' находится в той же папке, что и программа.\n");

char current\_path[256];

if (\_getcwd(current\_path, sizeof(current\_path)) != NULL) {

printf("Текущая папка: %s\n", current\_path);

}

else

{

printf("Не удалось определить текущую папку.\n");

}

}

printf("\nНажмите любую клавишу для продолжения...\n");

\_getch();

system("cls");

}

// processOrganization

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

ifstream corrFile;

ifstream addrFile;

vector<Address> addresses;

vector<Correspondence> correspondences;

bool addressFound = false;

bool correspondenceFound = false;

string addrFileError = "";

string corrFileError = "";

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

if (selectiveOutput) {

outputBuffer.push\_back(str);

}

else if (outfile) {

\*outfile << str << endl;

}

};

// --- Этап 1: Поиск в файле адресов ---

addrFile.open(addrFilename);

if (addrFile.is\_open()) {

string line;

while (getline(addrFile, line)) {

stringstream ss(line);

string currentOrg, address, contactPerson;

if (!(getline(ss, currentOrg, ';') && getline(ss, address, ';') && getline(ss, contactPerson, ';'))) {

continue;

}

size\_t first = currentOrg.find\_first\_not\_of(" \t");

if (string::npos != first) {

size\_t last = currentOrg.find\_last\_not\_of(" \t");

currentOrg = currentOrg.substr(first, (last - first + 1));

}

else {

currentOrg.clear();

}

if (!currentOrg.empty() && currentOrg == orgName) {

first = address.find\_first\_not\_of(" \t"); address = (first == string::npos) ? "" : address.substr(first);

first = contactPerson.find\_first\_not\_of(" \t"); contactPerson = (first == string::npos) ? "" : contactPerson.substr(first);

if (!address.empty() && address.back() == ';') address.pop\_back();

if (!contactPerson.empty() && contactPerson.back() == ';') contactPerson.pop\_back();

addresses.push\_back({ currentOrg, address.empty() ? "нет данных" : address, contactPerson.empty() ? "нет данных" : contactPerson }); // Сразу ставим заглушку при необходимости

addressFound = true;

}

}

addrFile.close();

}

else {

addrFileError = "Не удалось открыть файл адресов: " + addrFilename;

}

// --- Этап 2: Поиск в файле корреспонденции ---

corrFile.open(corrFilename);

if (corrFile.is\_open()) {

string line;

while (getline(corrFile, line)) {

stringstream ss(line);

string type, date, currentOrg;

if (!(getline(ss, type, ';') && getline(ss, date, ';') && getline(ss, currentOrg, ';'))) {

continue;

}

size\_t first = currentOrg.find\_first\_not\_of(" \t");

if (string::npos != first) {

size\_t last = currentOrg.find\_last\_not\_of(" \t");

currentOrg = currentOrg.substr(first, (last - first + 1));

}

else {

currentOrg.clear();

}

if (!currentOrg.empty() && currentOrg == orgName) {

first = type.find\_first\_not\_of(" \t"); type = (first == string::npos) ? "" : type.substr(first);

first = date.find\_first\_not\_of(" \t"); date = (first == string::npos) ? "" : date.substr(first);

if (!type.empty() && type.back() == ';') type.pop\_back();

if (!date.empty() && date.back() == ';') date.pop\_back();

correspondences.push\_back({ type.empty() ? "нет данных" : type, date.empty() ? "нет данных" : date, currentOrg }); // Сразу ставим заглушку при необходимости

correspondenceFound = true;

}

}

corrFile.close();

}

else {

corrFileError = "Не удалось открыть файл корреспонденции: " + corrFilename;

}

// --- Этап 3: Вывод результата ---

if (selectiveOutput) { // Режим поиска (в консоль)

// Сначала выводим ошибки открытия файлов, если были

if (!addrFileError.empty()) writeToTarget(addrFileError);

if (!corrFileError.empty()) writeToTarget(corrFileError);

// Проверяем, есть ли хоть какая-то информация или ошибки для вывода

if (!addressFound && !correspondenceFound && addrFileError.empty() && corrFileError.empty()) {

// Ничего не найдено и ошибок не было

writeToTarget("Организация \"" + orgName + "\" не найдена ни в одном из файлов.");

}

else {

// Выводим найденную информацию или заглушки

writeToTarget("Название организации: " + orgName);

// Вывод информации об адресе (или заглушек)

if (addressFound) {

for (const auto& addr : addresses) { // Могут быть дубликаты, выводим все найденные

writeToTarget(" Адрес: " + addr.address); // Заглушка уже внутри структуры

writeToTarget(" Фамилия руководителя: " + addr.contactPerson); // Заглушка уже внутри структуры

}

}

else if (addrFileError.empty()) { // Файл читался, но совпадений нет

writeToTarget(" Адрес: нет данных");

writeToTarget(" Фамилия руководителя: нет данных");

}

// Если была ошибка addrFileError, она уже выведена

// Вывод информации о корреспонденции (или заглушек)

writeToTarget(" Корреспонденция:"); // Всегда выводим заголовок

if (correspondenceFound) {

for (const auto& corr : correspondences) { // Могут быть дубликаты, выводим все найденные

writeToTarget("\t- Вид: " + corr.type + ", Дата: " + corr.date); // Заглушки уже внутри структуры

}

}

else if (corrFileError.empty()) { // Файл читался, но совпадений нет

writeToTarget("\t- Вид: нет данных, Дата: нет данных");

}

// Если была ошибка corrFileError, она уже выведена

writeToTarget("--------------------");

}

}

else { // Режим полного вывода (в файл)

if (printedOrganizations.count(orgName)) {

return;

}

printedOrganizations.insert(orgName);

writeToTarget("Название организации: " + orgName);

if (addressFound) {

for (const auto& addr : addresses) {

writeToTarget(" Адрес: " + addr.address); // Заглушка уже внутри

writeToTarget(" Фамилия руководителя: " + addr.contactPerson); // Заглушка уже внутри

}

}

else if (addrFileError.empty()) {

writeToTarget(" Адрес: нет данных");

writeToTarget(" Фамилия руководителя: нет данных");

}

else {

// Выводим ошибку, если адрес не найден из-за нее

writeToTarget(" " + addrFileError);

}

if (!corrFileError.empty()) {

writeToTarget(" " + corrFileError);

}

writeToTarget(" Корреспонденция:"); // Выводим заголовок

if (correspondenceFound) {

for (const auto& corr : correspondences) {

writeToTarget("\t- Вид: " + corr.type + ", Дата: " + corr.date); // Заглушки уже внутри

}

}

else if (corrFileError.empty()) {

writeToTarget("\t- Вид: нет данных, Дата: нет данных");

}

// Если была ошибка corrFileError, она уже выведена

writeToTarget("--------------------");

}

}

// getFilenamesFromUser

pair<string, string> getFilenamesFromUser(const char\* folderPath) {

char correspondenceFilenameBase[256];

char addressesFilenameBase[256];

char fullCorrFilename[256];

char fullAddressesFilename[256];

cout << "Для отмены ввода оставьте поле пустым или нажмите Esc.\n\n";

while (true) {

// Используем isValidFileNameChar для имени файла

getLineWithRestrictedChars("Введите имя файла с исходящей корреспонденцией (без префикса IC\_ и расширения .txt): ",

correspondenceFilenameBase, sizeof(correspondenceFilenameBase), isValidFileNameChar); // Передача функции

if (strlen(correspondenceFilenameBase) == 0) {

return { string(""), string("") };

}

else {

snprintf(fullCorrFilename, sizeof(fullCorrFilename), "%s%s%s%s", folderPath, ocfe, correspondenceFilenameBase, file\_extension);

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

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

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

continue;

}

break;

}

}

while (true) {

// Используем isValidFileNameChar для имени файла

getLineWithRestrictedChars("Введите имя файла с адресами организаций (без префикса AO\_ и расширения .txt): ",

addressesFilenameBase, sizeof(addressesFilenameBase), isValidFileNameChar); // Передача функции

if (strlen(addressesFilenameBase) == 0) {

return { string(""), string("") };

}

else {

snprintf(fullAddressesFilename, sizeof(fullAddressesFilename), "%s%s%s%s", folderPath, oa, addressesFilenameBase, file\_extension);

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

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

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

continue;

}

break;

}

}

return { string(fullCorrFilename), string(fullAddressesFilename) };

}

// runProgram

void runProgram(const char\* folderPath, const string& correspondenceFilename,

const string& addressesFilename, string& outputFilename) {

while (true) {

cout << "\nВыберите режим вывода:\n";

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

cout << "2. Полный вывод в файл\n";

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

char choice = \_getch();

cout << endl;

switch (choice) {

case '1': {

char orgNameBuffer[256];

// Используем isValidOrgNameChar для названия организации

getLineWithRestrictedChars("Введите название организации для поиска или нажмите Esc для отмены: ",

orgNameBuffer, sizeof(orgNameBuffer), isValidOrgNameChar); // Передача функции

if (orgNameBuffer[0] == '\0') {

break;

}

if (strlen(orgNameBuffer) == 0) {

printf("Название организации не может быть пустым.\n");

cout << "Нажмите любую клавишу для продолжения..." << endl; \_getch();

break;

}

string orgName(orgNameBuffer);

ofstream\* outfile = nullptr;

set<string> printedOrganizations;

vector<string> outputBuffer;

processOrganization(orgName, correspondenceFilename, addressesFilename,

true, outfile, printedOrganizations, outputBuffer);

for (const string& line : outputBuffer) {

cout << line << endl;

}

//cout << "\nНажмите любую клавишу для продолжения..." << endl;

//\_getch();

break;

}

case '2': {

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);

}

string baseFilenameStr = string(folderPath) + "Отчет\_" + corrFileNameOnly + "\_" + addrFileNameOnly;

string filename = baseFilenameStr + ".txt";

int i = 1;

while (filesystem::exists(filename)) {

filename = baseFilenameStr + "(" + to\_string(i++) + ")" + ".txt";

}

try {

ofstream outfile(filename);

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

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

cout << "Нажмите любую клавишу для продолжения..." << endl; \_getch();

break;

}

set<string> printedOrganizations;

vector<string> outputBuffer; // Не используется

ifstream addrFile(addressesFilename);

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, ';');

size\_t first = orgName.find\_first\_not\_of(" \t");

if (string::npos != first)

{

size\_t last = orgName.find\_last\_not\_of(" \t");

orgName = orgName.substr(first, (last - first + 1));

}

else {

orgName.clear();

}

if (!orgName.empty() && 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";

outfile.close();

cout << "Нажмите любую клавишу для продолжения..." << endl; \_getch();

break;

}

outfile.close();

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

}

catch (const exception& e)

{

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

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

cout << "Нажмите любую клавишу для продолжения..." << endl; \_getch();

}

break;

}

case 27:

return;

}

}

}

// menu

void menu() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

if (folder\_way[0] == '\0') {

normalizePath("", folder\_way, sizeof(folder\_way));

}

printf("\nПрограмма для обработки данных о корреспонденции и адресах организаций.\n");

printf("Она позволяет искать информацию об организациях и их корреспонденции, а также формировать отчеты.\n");

while (true) {

system("cls");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Программа для обработки данных корреспонденции и адресов \*\n");

printf("\* Программа позволяет искать и обрабатывать информацию об \*\n");

printf("\* организациях на основе двух файлов - \*\n");

printf("\* Исходящей корреспонденции (%s\*\*\*%s) \*\n", ocfe, file\_extension);

printf("\* Адресов организаций (%s\*\*\*%s) \*\n", oa, file\_extension);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("Главное меню:\n");

printf("1. Начать работу\n");

printf("2. Изменить путь к папке\n");

printf("3. Инструкция\n\n");

printf("Текущий путь: %s\n\n", folder\_way);

printf("Esc - выход\n\n");

char mainChoice = \_getch();

switch (mainChoice) {

case '1': {

if (folder\_way[0] == '\0') {

printf("Сначала выберите путь к папке (пункт 2).\n");

printf("Нажмите любую клавишу для продолжения..."); \_getch();

break;

}

pair<string, string> filenames = getFilenamesFromUser(folder\_way);

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

break;

}

string outputFilenameDummy;

runProgram(folder\_way, filenames.first, filenames.second, outputFilenameDummy);

break;

}

case '2': {

selectFolderPath();

break;

}

case '3': {

string instructionsFilePath = string(folder\_way) + Constants::INSTRUCTIONS\_FILE;

readInstructionsFromFile(instructionsFilePath);

break;

}

case 27:

return;

default:

break;

}

}

}

# 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.

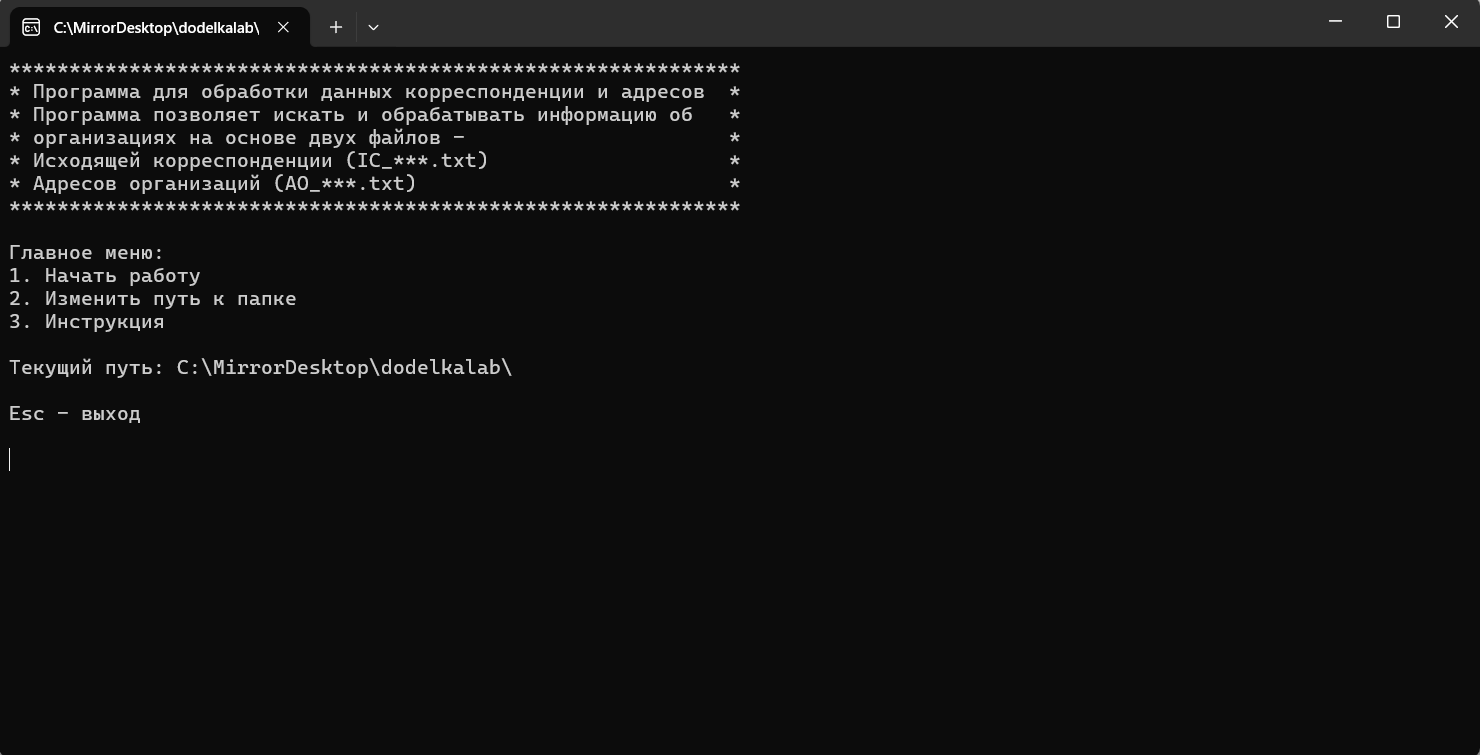


Рисунок 1 Главное меню.

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

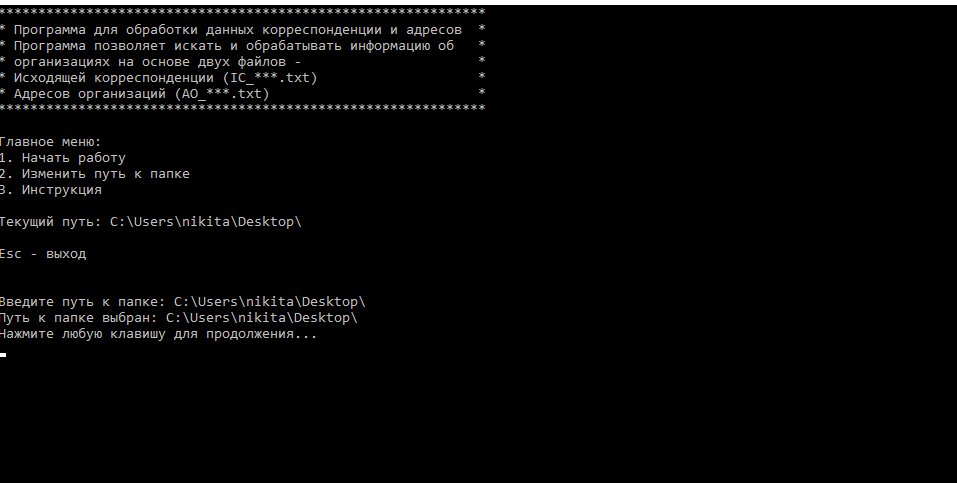
****

Рисунок 2 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).

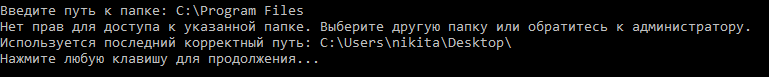
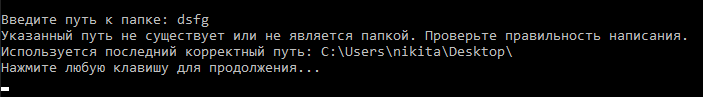


Рисунок 3 Некорректный ввод пути к папке хранения файлов.

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

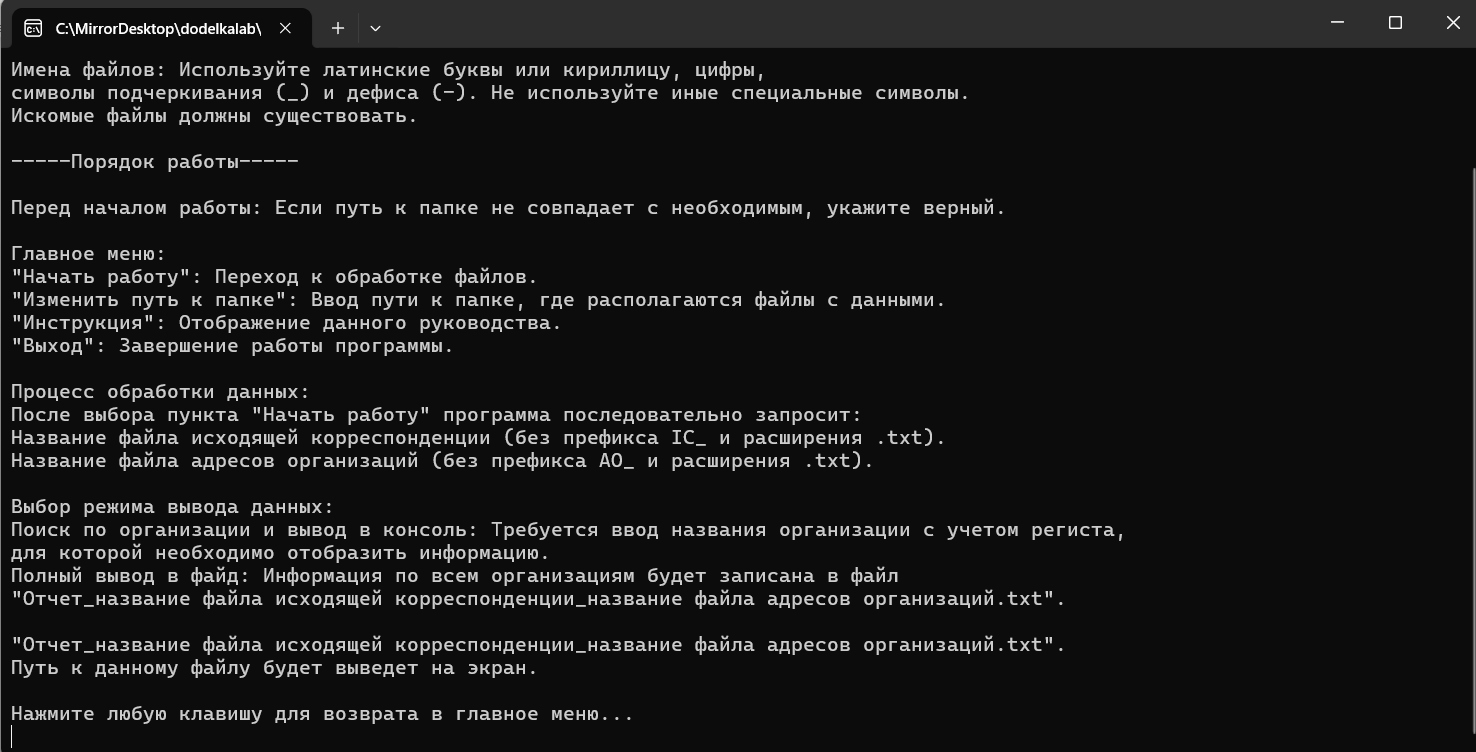


Рисунок 4 При выборе инструкции.

**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).

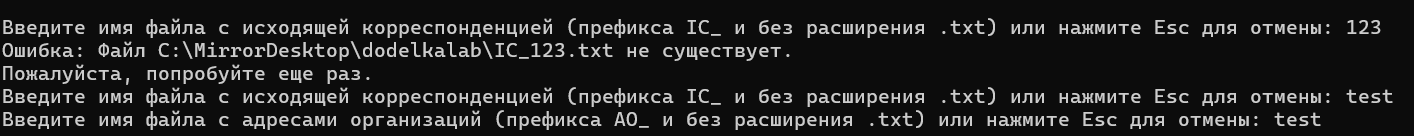


Рисунок 5 Пример заполнения имен файлов.

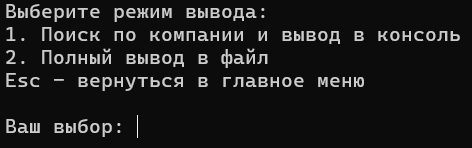


Рисунок 6 Выбор режима вывода.

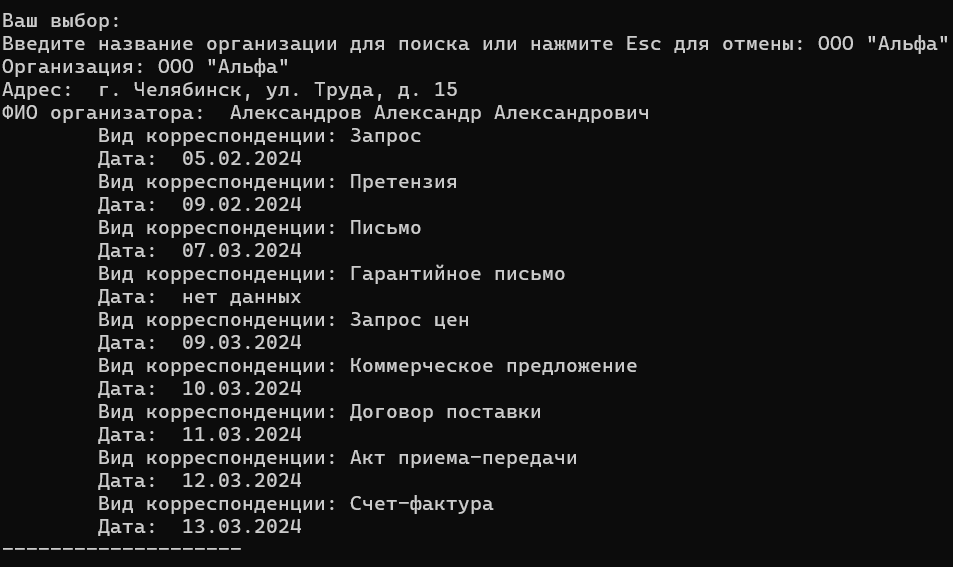


Рисунок 7 Пример вывода в консоль избирательного вывода.

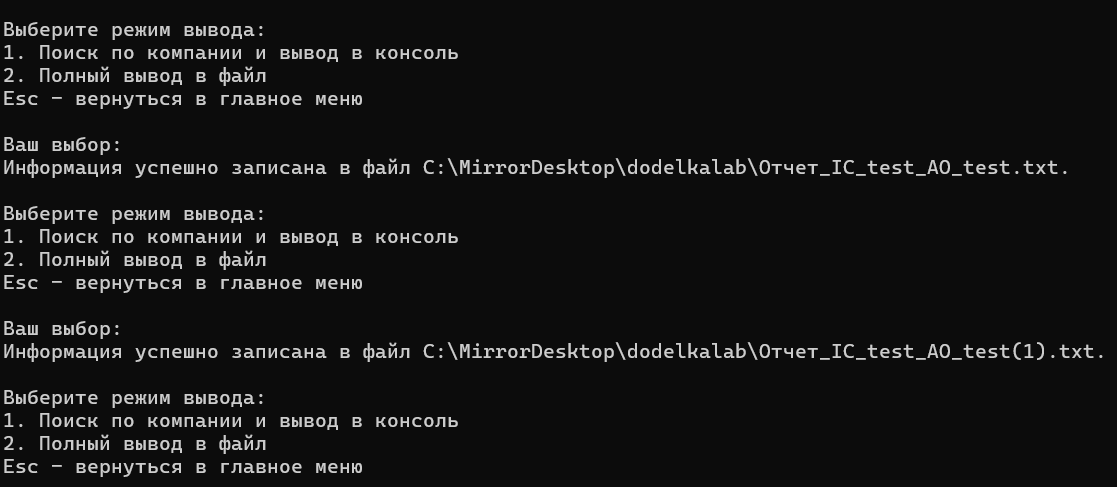


Рисунок 8 Пример использования полного вывода, с повтором, в файл.

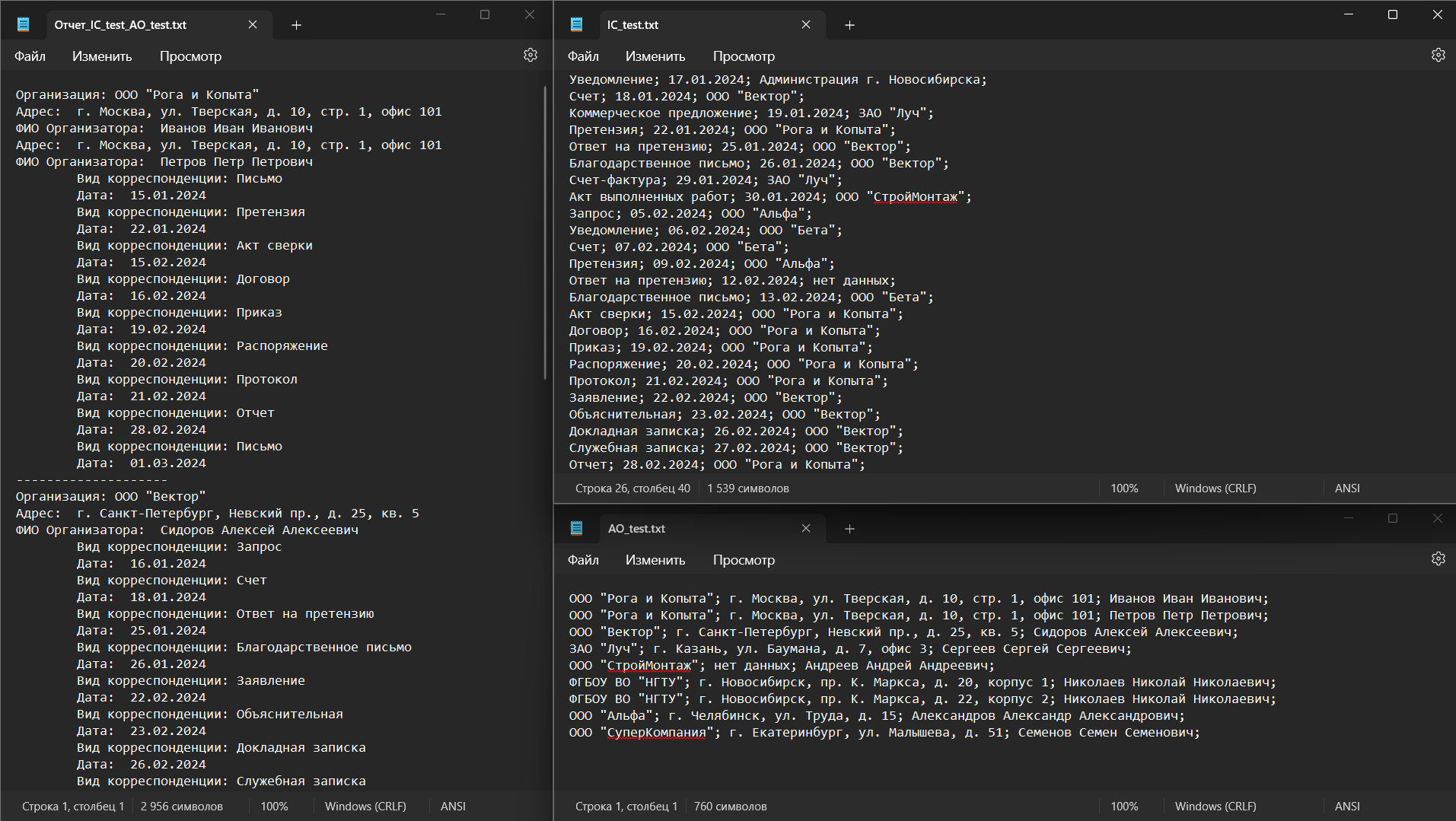


Рисунок 9 Пример преобразования входных файлов (слева) в выходной файл (справа).