**«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](#_Toc194062994)

[2. Псевдокод 3](#_Toc194062995)

[3. Сведения о программной реализации 6](#_Toc194062996)

[3.1. Язык программирования и среда разработки 6](#_Toc194062997)

[3.2. Описание входных и выходных данных 6](#_Toc194062998)

[3.3. Описание программных единиц 7](#_Toc194062999)

[4. Инструкция пользователю 9](#_Toc194063000)

[5. Листинг программной разработки 12](#_Toc194063001)

[6. Результаты тестирования программной разработки 26](#_Toc194063002)

# 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

// Константы

ПУТЬ\_К\_ПАПКЕ = пустая строка (256 символов)

РАСШИРЕНИЕ\_ФАЙЛА = ".txt"

ПРЕФИКС\_ФАЙЛА\_ИСХОДЯЩЕГО = "IC\_"

ПРЕФИКС\_ФАЙЛА\_АДРЕСА = "AO\_"

// Функция ввода строки с ограниченными символами

ФУНКЦИЯ вводСтрокиСОграниченнымиСимволами(инструкция, буфер, размер\_буфера, функция\_проверки):

ПОКА (истина):

ВВЕСТИ символ

ЕСЛИ символ = CTRL+V:

ВСТАВИТЬ ТЕКСТ ИЗ БУФЕРА ОБМЕНА, ПРОВЕРЯЯ КАЖДЫЙ СИМВОЛ

ИНАЧЕ ЕСЛИ символ = Esc:

ОЧИСТИТЬ БУФЕР И ВЕРНУТЬСЯ

ИНАЧЕ ЕСЛИ символ = Enter:

ЗАВЕРШИТЬ ВВОД И ВЕРНУТЬСЯ

ИНАЧЕ ЕСЛИ символ = Backspace:

УДАЛИТЬ ПОСЛЕДНИЙ СИМВОЛ ИЗ БУФЕРА

ИНАЧЕ:

ЕСЛИ функция\_проверки(символ) = истина:

ДОБАВИТЬ СИМВОЛ В БУФЕР

// Нормализация пути

ФУНКЦИЯ нормализоватьПуть(путь, нормализованный\_путь, размер\_нормализованного\_пути):

ЕСЛИ путь ПУСТОЙ:

ПОЛУЧИТЬ ТЕКУЩИЙ РАБОЧИЙ КАТАЛОГ

ИНАЧЕ:

ПОЛУЧИТЬ ПОЛНЫЙ ПУТЬ

УДАЛИТЬ ПОВТОРЯЮЩИЕСЯ СЛЕШИ

ДОБАВИТЬ СЛЕШ В КОНЕЦ, ЕСЛИ НЕТ

// Выбор пути к папке

ФУНКЦИЯ выбратьПутьКПапке():

ВВЕСТИ ПУТЬ К ПАПКЕ

НОРМАЛИЗОВАТЬ ПУТЬ

ПРОВЕРИТЬ СУЩЕСТВОВАНИЕ И ДОСТУПНОСТЬ ПАПКИ

ЕСЛИ ВСЕ ВЕРНО:

СОХРАНИТЬ ПУТЬ

// Чтение инструкций из файла

ФУНКЦИЯ прочитатьИнструкцииИзФайла(имя\_файла):

ОТКРЫТЬ ФАЙЛ

ЕСЛИ ФАЙЛ ОТКРЫТ:

ЧИТАТЬ И ВЫВОДИТЬ СТРОКИ

ИНАЧЕ:

ВЫВЕСТИ ОШИБКУ

// Удаление пробелов в начале и конце строки

ФУНКЦИЯ обрезать(строка):

УДАЛИТЬ ПРОБЕЛЫ В НАЧАЛЕ И КОНЦЕ СТРОКИ

ВОЗВРАТИТЬ ОБРЕЗАННУЮ СТРОКУ

// Обработка организации

ФУНКЦИЯ обработатьОрганизацию(имя\_организации, имя\_файла\_корреспонденции, имя\_файла\_адресов, выборочный\_вывод, файл\_вывода, напечатанные\_организации, буфер\_вывода):

СЧИТАТЬ ИНФОРМАЦИЮ ИЗ ФАЙЛА АДРЕСОВ

СЧИТАТЬ ИНФОРМАЦИЮ ИЗ ФАЙЛА КОРРЕСПОНДЕНЦИИ

ВЫВОДИТЬ ИНФОРМАЦИЮ В ЗАВИСИМОСТИ ОТ ТИПА ВЫВОДА

// Получение имен файлов от пользователя

ФУНКЦИЯ получитьИменаФайловОтПользователя(путь\_к\_папке):

ВВЕСТИ ИМЯ ФАЙЛА КОРРЕСПОНДЕНЦИИ

ВВЕСТИ ИМЯ ФАЙЛА АДРЕСОВ

ВОЗВРАТИТЬ ПОЛНЫЕ ПУТИ К ФАЙЛАМ

// Выбор типа ввода

ФУНКЦИЯ запуститьПрограмму(путь\_к\_папке, имя\_файла\_корреспонденции, имя\_файла\_адресов, имя\_файла\_вывода):

ПОКА (истина):

ВЫБРАТЬ ДЕЙСТВИЕ (1 - Поиск, 2 – Полный вывод)

ЕСЛИ Поиск ТО

ЗАПРОСИТЬ тип поля

ЗАПРОСИТЬ значения поля

ВЫПОЛНИТЬ ВЫБРАННЫЙ ВЫВОД

// Меню

ФУНКЦИЯ меню():

ПОКА (истина):

ОТОБРАЗИТЬ МЕНЮ

ВЫБРАТЬ ДЕЙСТВИЕ (1 - Запуск программы, 2 - Выбор папки, 3 - Инструкции)

ВЫПОЛНИТЬ ВЫБРАННОЕ ДЕЙСТВИЕ

ЕСЛИ Esc ТО:

ЗАВЕРШИТЬ РАБОТУ ПРОГРАММЫ

# 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 // Отключение предупреждений безопасности CRT

#pragma warning(disable:4996) // Отключение конкретного предупреждения C4996

using namespace std;

// --- Глобальные переменные ---

char folder\_way[256] = { 0 }; // Путь к рабочей папке

const char\* file\_extension = ".txt"; // Расширение файлов данных

const char\* ocfe = "IC\_"; // Префикс файлов корреспонденции

const char\* oa = "AO\_"; // Префикс файлов адресов

// --- Функции валидации символов ---

bool isValidFileNameChar(char c) { // Проверка символа для имени файла

unsigned char uc = (unsigned char)c;

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

}

bool isValidPathChar(char c) { // Проверка символа для пути

unsigned char uc = (unsigned char)c;

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

}

bool isValidOrgNameChar(char c) { // Проверка символа для названия организации

unsigned char uc = (unsigned char)c;

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

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

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

// Доп. символы: - \_ . , " ( ) №(211) + ! & : «(171) »(187) #

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 == '#');

}

bool isValidAddressChar(char c) { // Проверка символа для адреса

unsigned char uc = (unsigned char)c;

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

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

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

// Доп. символы: - , . / №(211)

return (is\_letter || is\_digit || is\_space || uc == '-' || uc == ',' || uc == '.' || uc == '/' || uc == 211);

}

bool isValidContactPersonChar(char c) { // Проверка символа для ФИО руководителя

unsigned char uc = (unsigned char)c;

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

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

// Доп. символы: -

return (is\_letter || is\_space || uc == '-');

}

bool isValidCorrTypeChar(char c) { // Проверка символа для вида корреспонденции

unsigned char uc = (unsigned char)c;

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

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

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

// Доп. символы: - \_ . , ( )

return (is\_letter || is\_digit || is\_space || uc == '-' || uc == '\_' || uc == '.' || uc == ',' || uc == '(' || uc == ')');

}

bool isValidCorrDateChar(char c) { // Проверка символа для даты корреспонденции (разрешаем буквы для "нет данных")

unsigned char uc = (unsigned char)c;

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

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

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

// Доп. символы: .

return (is\_letter || is\_digit || is\_space || uc == '.');

}

// --- Функции ввода/вывода и работы с путями ---

// Получение строки с консоли с фильтрацией символов и поддержкой буфера обмена (Ctrl+V)

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

printf("%s", instruction); // Вывод приглашения

int i = 0;

buffer[0] = '\0';

while (true) {

int key = \_getch();

if (key == 22) { // CTRL V

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) { // Проверка на спецсимволы

if (\_kbhit()) {

\_getch();

continue;

}

}

if (key == 27) { // Esc

buffer[0] = '\0';

printf("\n");

return;

}

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

buffer[i] = '\0';

printf("\n");

return;

}

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

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++;

}

}

}

}

}

// Нормализация пути: абсолютный, без двойных слешей, с '\' на конце

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

if (path == nullptr || path[0] == '\0') { // Если путь пуст, берем текущую директорию

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); // Добавляем '\' в конец, если нет

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) { // Получаем полный путь

SAFE\_STRCPY(normalized\_path, path, normalized\_path\_size); // Если не вышло, копируем как есть

}

int j = 0; // Удаление двойных слешей

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); // Добавляем '\' в конец, если нет

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 '\\'.\n");

}

}

}

// Выбор и проверка пути к папке

void selectFolderPath() {

SetConsoleCP(1251); SetConsoleOutputCP(1251); // Установка кодировок

printf("\n");

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

char normalized\_path\_temp[256];

// Запрос пути с валидацией

getLineWithRestrictedChars("Введите путь к папке (или оставьте пустым для текущей, Esc для отмены): ",

folder\_way\_new, sizeof(folder\_way\_new), isValidPathChar);

if (folder\_way\_new[0] == '\0' && strlen(folder\_way\_new) == 0) { // Проверка на отмену (Esc или Enter на пустом поле)

printf("Выбор пути отменен.\n");

if (folder\_way[0] == '\0') normalizePath("", folder\_way, sizeof(folder\_way)); // Установить текущий, если ничего не было выбрано

else printf("Используется последний корректный путь: %s\n", folder\_way); // Иначе оставляем старый

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

return;

}

normalizePath(folder\_way\_new, normalized\_path\_temp, sizeof(normalized\_path\_temp)); // Нормализация введенного пути

struct stat path\_stat; // Структура для информации о файле/папке

if (stat(normalized\_path\_temp, &path\_stat) == 0) { // Проверка существования пути

if (path\_stat.st\_mode & S\_IFDIR) { // Проверка, является ли путь директорией

char test\_file\_path[512]; // Попытка создать/удалить файл для проверки прав на запись

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) { // Права есть

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)); // Сохраняем путь

}

else { // Нет прав на запись

printf("Ошибка: Нет прав для записи в указанную папку.\n");

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

}

}

else { // Путь существует, но это не папка

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

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

}

}

else { // Путь не существует или недоступен

printf("Ошибка: Указанный путь не существует или недоступен.\n");

// Доп. проверка на случай, если stat не сработал из-за '\' на конце

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

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

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

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

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

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

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

}

else { // Путь все равно неверный

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

}

}

else { // Путь точно неверный

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

}

}

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

}

// --- Функции основной логики ---

// Чтение и вывод инструкций из файла

void readInstructionsFromFile(const string& filename) {

system("cls"); // Очистка экрана

FILE\* file = fopen(filename.c\_str(), "r"); // Открытие файла инструкций

if (file) { // Если файл открыт

char line[256];

while (fgets(line, sizeof(line), file)) { // Чтение и вывод строк

printf("%s", line);

}

fclose(file);

}

else { // Ошибка открытия файла

printf("Не удалось открыть файл с инструкцией: %s\n", filename.c\_str());

char current\_path[256]; // Попытка показать текущую папку программы

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

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

}

else {

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

}

}

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

\_getch();

system("cls"); // Очистка экрана

}

// Удаление пробелов в начале и конце строки

string trim(const string& str) {

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

if (string::npos == first) return str;

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

return str.substr(first, (last - first + 1));

}

// Обработка данных для одной организации: поиск в файлах и формирование вывода

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

ifstream corrFile, addrFile;

vector<Address> addresses;

vector<Correspondence> correspondences;

bool addressFound = false, correspondenceFound = false;

string addrFileError = "", corrFileError = "";

// Лямбда для вывода в буфер или файл

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

if (selectiveOutput) outputBuffer.push\_back(str);

else if (outfile) \*outfile << str << endl;

};

// --- 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; // Пропуск некорректных строк

currentOrg = trim(currentOrg); // Убираем пробелы у имени

if (!currentOrg.empty() && currentOrg == orgName) { // Сравниваем с искомым именем

address = trim(address);

contactPerson = trim(contactPerson);

// Добавляем, заменяя пустые поля на "нет данных"

addresses.push\_back({ currentOrg, address.empty() ? "нет данных" : address, contactPerson.empty() ? "нет данных" : contactPerson });

addressFound = true;

}

}

addrFile.close();

}

else {

addrFileError = "Ошибка: Не удалось открыть файл адресов: " + addrFilename; // Сохраняем ошибку

}

// --- 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;

currentOrg = trim(currentOrg);

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

type = trim(type);

date = trim(date);

correspondences.push\_back({ type.empty() ? "нет данных" : type, date.empty() ? "нет данных" : date, currentOrg });

correspondenceFound = true;

}

}

corrFile.close();

}

else {

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

}

// --- Вывод результата ---

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

if (!addrFileError.empty()) writeToTarget(addrFileError); // Вывод ошибок файлов

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

if (addressFound || correspondenceFound || !addrFileError.empty() || !corrFileError.empty()) { // Если есть что выводить

writeToTarget("Название организации: " + orgName);

if (addressFound) { // Вывод адресов

for (const auto& addr : addresses) {

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

writeToTarget(" Фамилия руководителя: " + addr.contactPerson);

}

}

else if (addrFileError.empty()) { // Адрес не найден, но ошибки не было

writeToTarget(" Адрес: нет данных");

writeToTarget(" Фамилия руководителя: нет данных");

}

writeToTarget(" Корреспонденция:"); // Вывод корреспонденции

if (correspondenceFound) {

for (const auto& corr : correspondences) {

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

}

}

else if (corrFileError.empty()) { // Корреспонденция не найдена, ошибки не было

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

}

writeToTarget("--------------------"); // Разделитель

}

}

else { // Режим отчета (в файл)

if (printedOrganizations.count(orgName)) return; // Пропуск дубликатов

printedOrganizations.insert(orgName); // Отмечаем организацию как выведенную

writeToTarget("Название организации: " + orgName);

if (addressFound) { // Вывод адресов

for (const auto& addr : addresses) {

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

writeToTarget(" Фамилия руководителя: " + addr.contactPerson);

}

}

else if (addrFileError.empty()) { // Адрес не найден, ошибки не было

writeToTarget(" Адрес: нет данных");

writeToTarget(" Фамилия руководителя: нет данных");

}

else {

writeToTarget(" " + addrFileError); // Вывод ошибки адреса

}

if (!corrFileError.empty()) writeToTarget(" " + corrFileError); // Вывод ошибки корреспонденции

writeToTarget(" Корреспонденция:"); // Вывод корреспонденции

if (correspondenceFound) {

for (const auto& corr : correspondences) {

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

}

}

else if (corrFileError.empty()) { // Корреспонденция не найдена, ошибки не было

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

}

writeToTarget("--------------------"); // Разделитель

}

}

// Запрос имен файлов у пользователя

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

char corrBase[256], addrBase[256];

char fullCorr[512], fullAddr[512];

cout << "\nДля отмены ввода файла оставьте поле пустым и нажмите Enter, или нажмите Esc.\n\n";

while (true) { // Запрос файла корреспонденции

getLineWithRestrictedChars("Введите имя файла исходящей корреспонденции (без IC\_ и .txt): ", corrBase, sizeof(corrBase), isValidFileNameChar);

if (strlen(corrBase) == 0) return { "", "" }; // Отмена

snprintf(fullCorr, sizeof(fullCorr), "%s%s%s%s", folderPath, ocfe, corrBase, file\_extension); // Формирование полного пути

if (!filesystem::exists(fullCorr)) { // Проверка существования

cerr << "Ошибка: Файл " << fullCorr << " не существует.\nПопробуйте еще раз.\n";

continue;

}

break; // Файл найден

}

while (true) { // Запрос файла адресов

getLineWithRestrictedChars("Введите имя файла адресов организаций (без AO\_ и .txt): ", addrBase, sizeof(addrBase), isValidFileNameChar);

if (strlen(addrBase) == 0) return { "", "" }; // Отмена

snprintf(fullAddr, sizeof(fullAddr), "%s%s%s%s", folderPath, oa, addrBase, file\_extension); // Формирование полного пути

if (!filesystem::exists(fullAddr)) { // Проверка существования

cerr << "Ошибка: Файл " << fullAddr << " не существует.\nПопробуйте еще раз.\n";

continue;

}

break; // Файл найден

}

return { string(fullCorr), string(fullAddr) }; // Возврат полных путей

}

// Основная функция программы: выбор режима, поиск/обработка данных, вывод

void runProgram(const char\* folderPath, const string& correspondenceFilename, const string& addressesFilename, string& outputFilename) {

while (true) { // Цикл подменю

cout << "\nВыберите режим работы:\n"

<< "1. Поиск по данным и вывод в консоль\n"

<< "2. Полный вывод в файл (по всем организациям из обоих файлов)\n"

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

char choice = \_getch();

cout << endl;

switch (choice) {

case '1': { // Режим 1: Избирательный поиск

cout << "\nВыберите поле для поиска:\n"

<< "1. Название организации\n"

<< "2. Адрес организации\n"

<< "3. Фамилия руководителя\n"

<< "4. Вид корреспонденции\n"

<< "5. Дата подготовки\n"

<< "Esc - отмена\n";

char searchFieldChoice;

int searchFieldIndex;

bool (\*validatorFunc)(char) = nullptr;

string searchFieldName = "", fileToScan = "";

cout << endl;

while (true) {

searchFieldChoice = \_getch();

searchFieldIndex = searchFieldChoice - '0';

if (searchFieldChoice == 27) break; // Отмена

switch (searchFieldChoice) {

case '1': validatorFunc = isValidOrgNameChar; searchFieldName = "Название организации"; break;

case '2': validatorFunc = isValidAddressChar; searchFieldName = "Адрес организации"; fileToScan = addressesFilename; break;

case '3': validatorFunc = isValidContactPersonChar; searchFieldName = "Фамилия руководителя"; fileToScan = addressesFilename; break;

case '4': validatorFunc = isValidCorrTypeChar; searchFieldName = "Вид корреспонденции"; fileToScan = correspondenceFilename; break;

case '5': validatorFunc = isValidCorrDateChar; searchFieldName = "Дата подготовки"; fileToScan = correspondenceFilename; break;

default: continue;

}

break;

}

char searchQueryBuffer[256];

getLineWithRestrictedChars(("Введите " + searchFieldName + " для поиска (Esc - отмена): ").c\_str(),

searchQueryBuffer, sizeof(searchQueryBuffer), validatorFunc);

if (searchQueryBuffer[0] == '\0') break;

string searchQuery = trim(string(searchQueryBuffer));

if (searchQuery.empty()) {

printf("Поисковый запрос не может быть пустым.\n");

break;

}

vector<string> matchingRecords;

set<string> foundOrganizations;

// --- Сканирование файла ---

if (searchFieldIndex == 1) {

foundOrganizations.insert(searchQuery);

bool recordFound = false; ifstream tempFile;

tempFile.open(addressesFilename);

if (tempFile.is\_open()) { string line; while (getline(tempFile, line)) { stringstream ss(line); string org, f1, f2; if (getline(ss, org, ';') && getline(ss, f1, ';') && getline(ss, f2, ';') && trim(org) == searchQuery) { matchingRecords.push\_back("Название организации: " + trim(org)); recordFound = true; break; } } tempFile.close(); }

if (!recordFound) {

tempFile.open(correspondenceFilename);

if (tempFile.is\_open()) { string line; while (getline(tempFile, line)) { stringstream ss(line); string f1, f2, org; if (getline(ss, f1, ';') && getline(ss, f2, ';') && getline(ss, org, ';') && trim(org) == searchQuery) { matchingRecords.push\_back("Название организации: " + trim(org)); recordFound = true; break; } } tempFile.close(); }

}

}

else {

ifstream inputFile(fileToScan);

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

cerr << "Ошибка: Не удалось открыть файл: " << fileToScan << endl;

}

else {

string line;

while (getline(inputFile, line)) {

stringstream ss(line); string orgName, f1, f2, fieldToCompare, matchInfo = "";

if (fileToScan == addressesFilename) {

if (getline(ss, orgName, ';') && getline(ss, f1, ';') && getline(ss, f2, ';')) {

orgName = trim(orgName);

if (searchFieldIndex == 2) { fieldToCompare = trim(f1); if (!fieldToCompare.empty() && fieldToCompare.find(searchQuery) != string::npos) { foundOrganizations.insert(orgName); matchInfo = "Адрес: " + f1 + ", Название организации: " + orgName; } }

else if (searchFieldIndex == 3) { fieldToCompare = trim(f2); if (!fieldToCompare.empty() && fieldToCompare == searchQuery) { foundOrganizations.insert(orgName); matchInfo = "Фамилия руководителя: " + f2 + ", Название организации: " + orgName; } }

}

}

else {

if (getline(ss, f1, ';') && getline(ss, f2, ';') && getline(ss, orgName, ';')) {

orgName = trim(orgName);

if (searchFieldIndex == 4) { fieldToCompare = trim(f1); if (!fieldToCompare.empty() && fieldToCompare == searchQuery) { foundOrganizations.insert(orgName); matchInfo = "Вид: " + f1 + ", Дата: " + trim(f2) + ", Название организации: " + orgName; } }

else if (searchFieldIndex == 5) { fieldToCompare = trim(f2); if (!fieldToCompare.empty() && fieldToCompare == searchQuery) { foundOrganizations.insert(orgName); matchInfo = "Дата: " + f2 + ", Вид: " + trim(f1) + ", Название организации: " + orgName; } }

}

}

if (!matchInfo.empty()) matchingRecords.push\_back(matchInfo);

}

inputFile.close();

}

}

// --- Вывод результатов поиска (Изменено!) ---

vector<string> outputBuffer; // Буфер для вывода полной информации

set<string> dummyPrintedSet; // Пустышка для processOrganization

// Собираем полную информацию обо всех найденных организациях

for (const string& orgNameToProcess : foundOrganizations) {

processOrganization(orgNameToProcess, correspondenceFilename, addressesFilename, true, nullptr, dummyPrintedSet, outputBuffer);

}

// Проверяем, найдены ли какие-либо данные

bool dataFound = !outputBuffer.empty() || !matchingRecords.empty();

if (!dataFound) {

cout << "\nПо введенному вами запросу \"" << searchQuery << "\" ничего не найдено." << endl;

}

else {

// Логика вывода в зависимости от поля поиска

if (searchFieldIndex == 4 || searchFieldIndex == 5) { // Вид или Дата

cout << "\nПо введенному вами запросу " << searchFieldName << " \"" << searchQuery << "\" найдены следующие данные:\n" << endl;

for (const string& record : matchingRecords) cout << record << endl;

cout << "\nСвязанные компании и их данные:\n" << endl;

for (const string& line : outputBuffer) cout << line << endl;

}

else if (searchFieldIndex == 2 || searchFieldIndex == 3) { // Адрес или Руководитель

cout << "\nСвязанные компании и их данные:\n" << endl;

for (const string& line : outputBuffer) cout << line << endl;

}

else if (searchFieldIndex == 1) { // Название организации

// Просто выводим buffer, заголовок "Связанные компании..." не нужен

cout << endl; // Отступ перед выводом

for (const string& line : outputBuffer) cout << line << endl;

}

}

// Убрана пауза

break; // Выход из case '1' и возврат к началу цикла while(true)

}

case '2': { // Режим 2: Полный вывод в файл

// Формирование имени файла отчета

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

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

size\_t pos = corrName.rfind(".txt"); if (pos != string::npos) corrName = corrName.substr(0, pos);

pos = addrName.rfind(".txt"); if (pos != string::npos) addrName = addrName.substr(0, pos);

string baseFilename = string(folderPath) + "Отчет\_" + corrName + "\_" + addrName;

string filename = baseFilename + ".txt"; int i = 1;

while (filesystem::exists(filename)) filename = baseFilename + "(" + to\_string(i++) + ").txt";

try {

ofstream outfile(filename);

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

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

break;

}

set<string> allUniqueOrgNames;

set<string> printedOrganizations;

ifstream fileReader; string line;

// Чтение имен из файла адресов

fileReader.open(addressesFilename);

if (fileReader.is\_open()) { while (getline(fileReader, line)) { stringstream ss(line); string org; if (getline(ss, org, ';')) { org = trim(org); if (!org.empty()) allUniqueOrgNames.insert(org); } } fileReader.close(); }

else { cerr << "Предупреждение: не удалось открыть файл адресов: " << addressesFilename << endl; outfile << "!!! Ошибка чтения файла адресов: " << addressesFilename << " !!!" << endl << endl; }

// Чтение имен из файла корреспонденции

fileReader.clear();

fileReader.open(correspondenceFilename);

if (fileReader.is\_open()) { while (getline(fileReader, line)) { stringstream ss(line); string f1, f2, org; if (getline(ss, f1, ';') && getline(ss, f2, ';') && getline(ss, org, ';')) { org = trim(org); if (!org.empty()) allUniqueOrgNames.insert(org); } } fileReader.close(); }

else { cerr << "Предупреждение: не удалось открыть файл корреспонденции: " << correspondenceFilename << endl; outfile << "!!! Ошибка чтения файла корреспонденции: " << correspondenceFilename << " !!!" << endl << endl; }

// Обработка и запись в файл

vector<string> dummyOutputBuffer;

for (const string& orgNameToProcess : allUniqueOrgNames) {

processOrganization(orgNameToProcess, correspondenceFilename, addressesFilename, false, &outfile, printedOrganizations, dummyOutputBuffer);

}

outfile.close();

cout << "Отчет успешно сформирован и записан в файл: " << filename << ".\n";

}

catch (const exception& e) {

cerr << "Ошибка файловой операции: " << e.what() << endl;

}

// Убрана пауза

break; // Выход из case '2' и возврат к началу цикла while(true)

}

case 27: return; // Esc - выход из подменю (возврат в главное меню)

default:

cout << "Неверный выбор. Пожалуйста, используйте 1, 2 или Esc." << endl;

break;

}

}

}

// --- Главное меню ---

void menu() {

SetConsoleCP(1251); SetConsoleOutputCP(1251); // Установка кодировок

if (folder\_way[0] == '\0') normalizePath("", folder\_way, sizeof(folder\_way)); // Установка текущего пути, если не выбран

while (true) { // Главный цикл меню

system("cls"); // Очистка экрана

// Отображение заголовка и меню

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

"\* Программа обработки данных корреспонденции и адресов \*\n"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

"\* Анализирует файлы:\n"

"\* - Исходящая корреспонденция (Формат: %s\*%s)\n"

"\* - Адреса организаций (Формат: %s\*%s)\n"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"

"Главное меню:\n"

"1. Начать работу\n"

"2. Изменить путь к папке с файлами\n"

"3. Показать инструкцию\n\n"

"Текущий путь к папке: %s\n\n"

"Esc - выход из программы\n\n",

ocfe, file\_extension, oa, file\_extension, (folder\_way[0] == '\0' ? "НЕ ВЫБРАН" : folder\_way));

char mainChoice = \_getch();

switch (mainChoice) {

case '1': { // Начать работу

printf("\n");

if (folder\_way[0] == '\0') { printf("Ошибка: Сначала необходимо выбрать путь к папке (пункт 2).\n"); printf("Нажмите любую клавишу..."); \_getch(); break; }

pair<string, string> filenames = getFilenamesFromUser(folder\_way); // Запрос имен файлов

if (filenames.first == "" || filenames.second == "")

break;

string dummyOutput;

runProgram(folder\_way, filenames.first, filenames.second, dummyOutput); // Запуск основной логики

break; // Возврат в меню после runProgram

}

case '2': { // Изменить путь

printf("\n");

selectFolderPath();

break;

}

case '3': { // Инструкция

printf("\n");

readInstructionsFromFile(Constants::INSTRUCTIONS\_FILE);

break;

}

case 27: { // Esc - выход

printf("\nЗавершение работы программы.\n");

return; // Выход из menu() и программы

}

default: { // Неверный выбор

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.

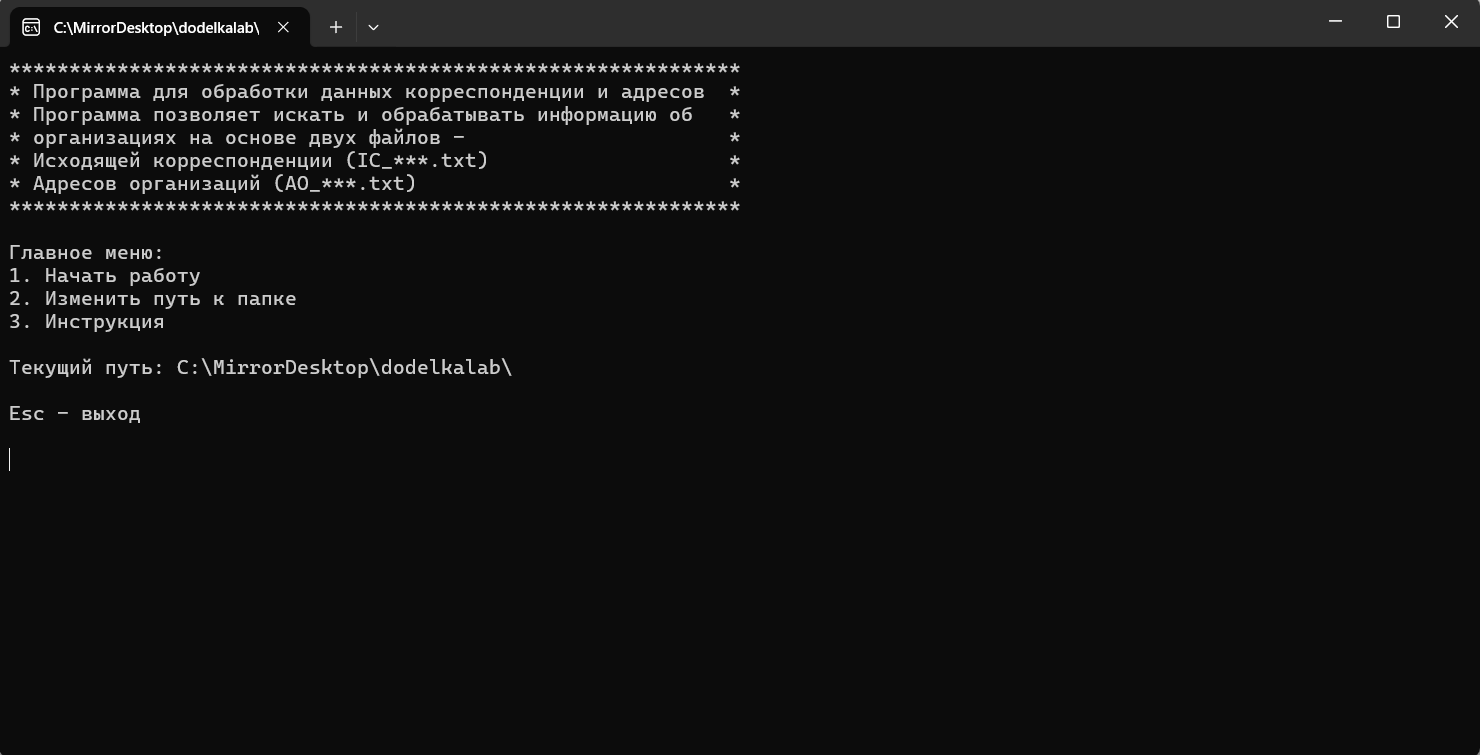


Рисунок 1 Главное меню.

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

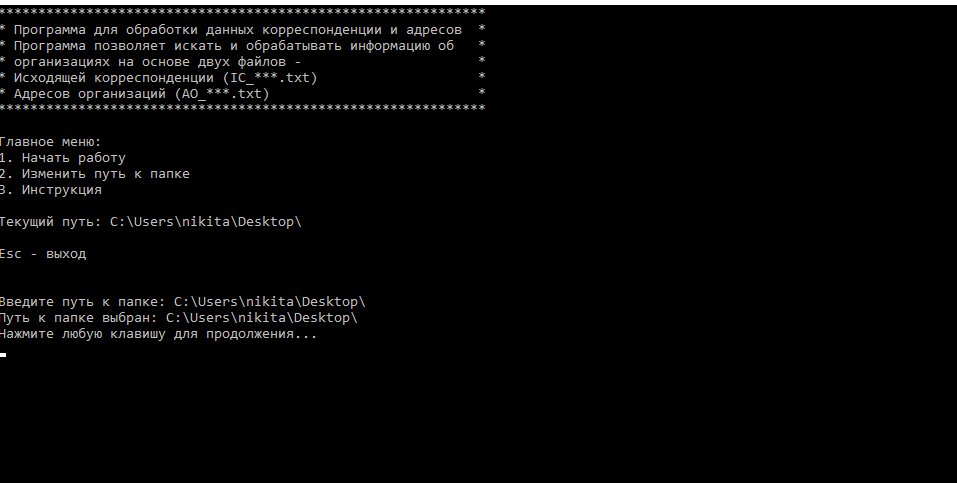
****

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).

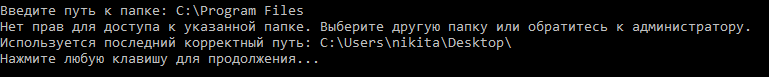
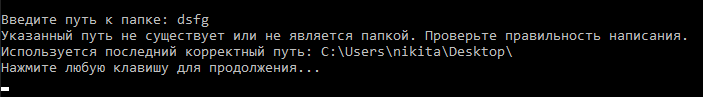


Рисунок 3 Некорректный ввод пути к папке хранения файлов.

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

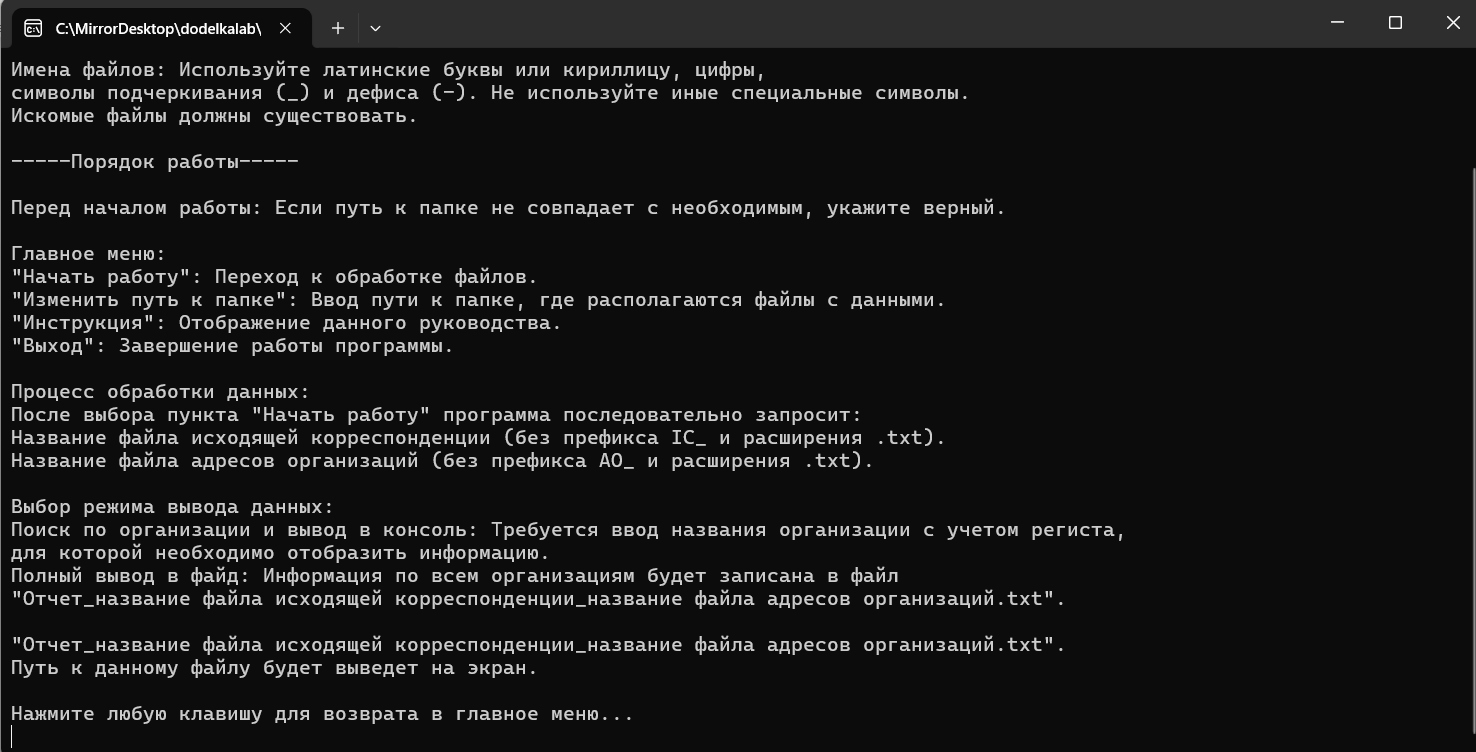


Рисунок 4 При выборе инструкции.

**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).

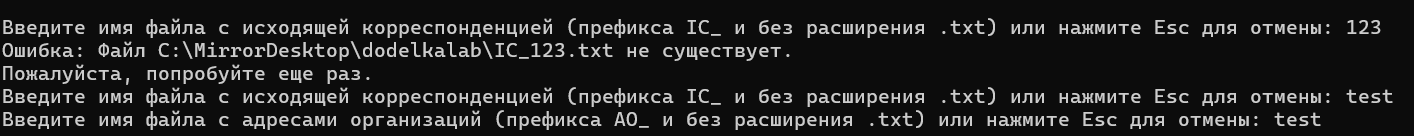


Рисунок 5 Пример заполнения имен файлов.

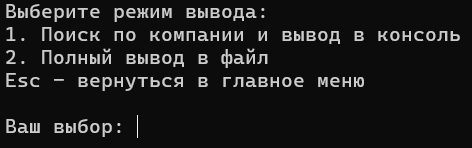


Рисунок 6 Выбор режима вывода.

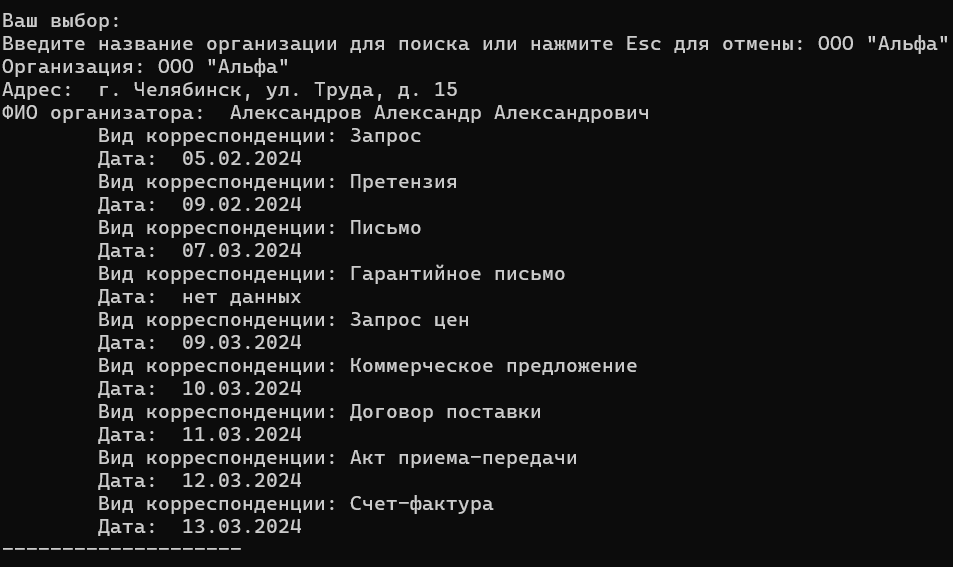


Рисунок 7 Пример вывода в консоль избирательного вывода.

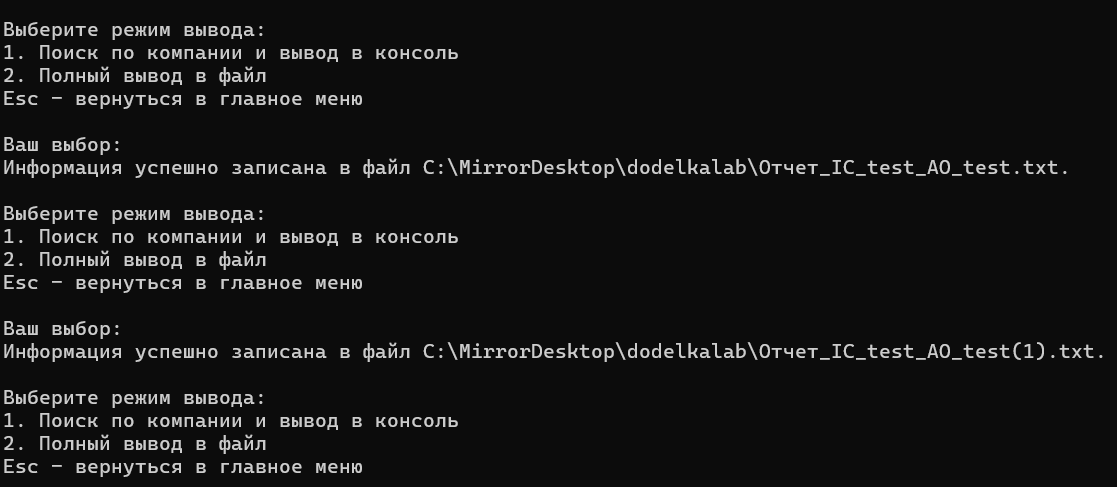


Рисунок 8 Пример использования полного вывода, с повтором, в файл.

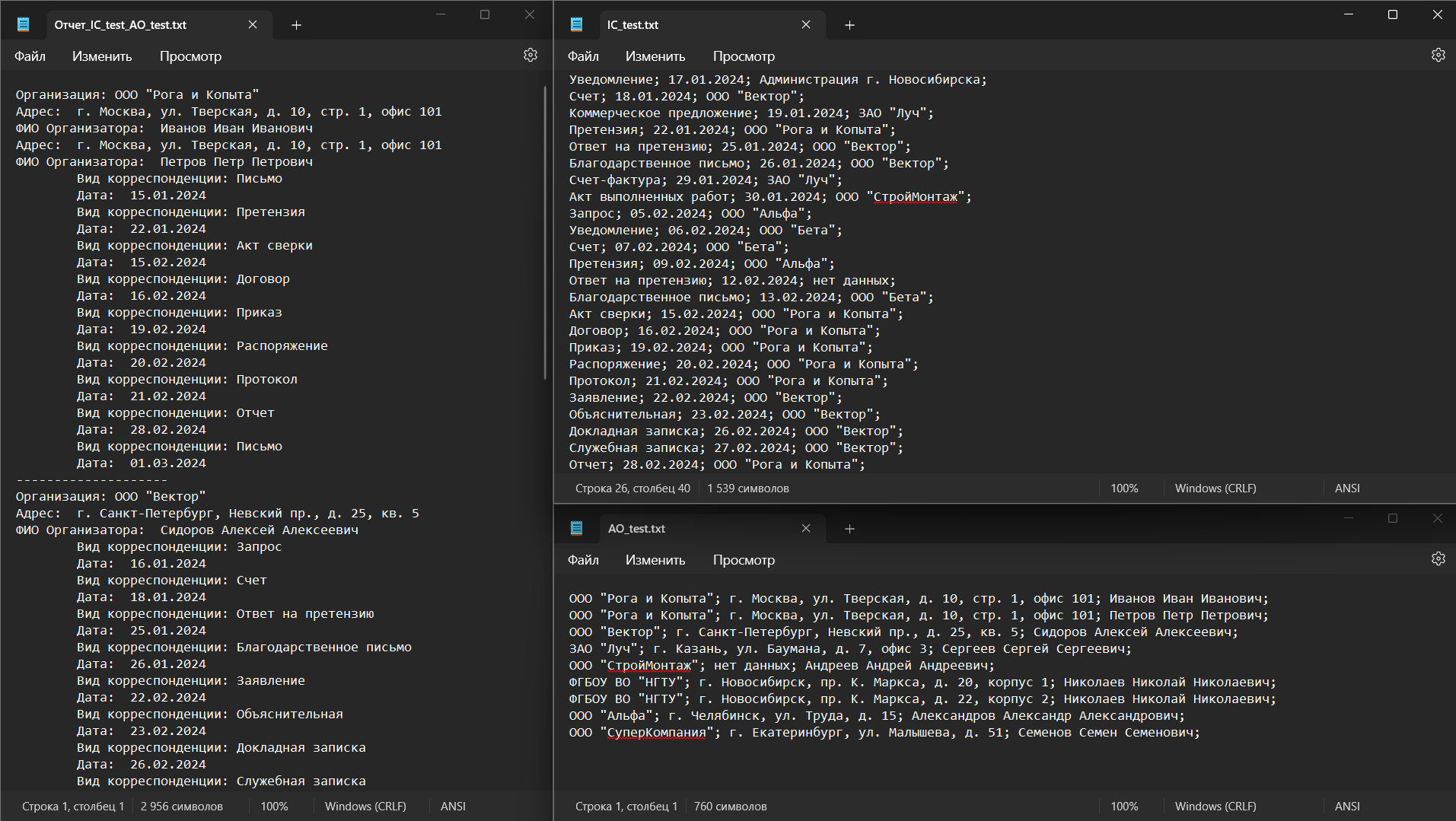


Рисунок 9 Пример преобразования входных файлов (слева) в выходной файл (справа).