**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

(национальный исследовательский университет)»

**ЛАБОРАТОРНАЯ РАБОТА**

**по курсу**

**«Программирование на языках высокого уровня»**

|  |
| --- |
| Вариант 18  Часть 1 |
|  |

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

Барчев Николай  
 Борисович

Москва, 2025 г.

Оглавление

[1. Задание 3](#_Toc194063734)

[2. Псевдокод 3](#_Toc194063735)

[3. Сведения о программной реализации 9](#_Toc194063736)

[3.1. Язык программирования и среда разработки 9](#_Toc194063737)

[3.2. Описание входных и выходных данных 9](#_Toc194063738)

[3.3. Описание программных единиц 10](#_Toc194063739)

[4. Инструкция пользователю 12](#_Toc194063740)

[5. Листинг программной разработки 15](#_Toc194063741)

[6. Результаты тестирования программной разработки 30](#_Toc194063742)

# 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 для получения ввода с ограниченными символами

ФУНКЦИЯ получитьСтрокуСОграниченнымиСимволами(инструкция, буфер, размер\_буфера, функцияПроверкиСимвола)

ВЫВЕСТИ инструкция

ИНИЦИАЛИЗИРОВАТЬ счетчик = 0

УСТАНОВИТЬ буфер[0] = НУЛЬ

ПОКА ИСТИНА

клавиша = ПОЛУЧИТЬ ВВОД СИМВОЛА

ЕСЛИ клавиша ЯВЛЯЕТСЯ СПЕЦИАЛЬНОЙ КЛАВИШЕЙ ТОГДА

ИГНОРИРОВАТЬ ЕЕ

ИНАЧЕ ЕСЛИ клавиша ЯВЛЯЕТСЯ ESC ТОГДА

ОЧИСТИТЬ буфер

ВЫВЕСТИ ЧИСТУЮ СТРОКУ

ВЕРНУТЬСЯ

ИНАЧЕ ЕСЛИ клавиша ЯВЛЯЕТСЯ ENTER ТОГДА

УСТАНОВИТЬ буфер[счетчик] = НУЛЬ

ВЫВЕСТИ буфер

ВЕРНУТЬСЯ

ИНАЧЕ ЕСЛИ клавиша ЯВЛЯЕТСЯ BACKSPACE ТОГДА

ЕСЛИ счетчик > 0 ТОГДА

УМЕНЬШИТЬ счетчик

ВЫВЕСТИ BACKSPACE И ПРОБЕЛ ДЛЯ УДАЛЕНИЯ ПОСЛЕДНЕГО СИМВОЛА

УСТАНОВИТЬ буфер[счетчик] = НУЛЬ

ИНАЧЕ ЕСЛИ клавиша ЯВЛЯЕТСЯ CTRL+V ТОГДА

ЕСЛИ БУФЕР ОБМЕНА МОЖЕТ БЫТЬ ОТКРЫТ ТОГДА

ПОЛУЧИТЬ ТЕКСТ ИЗ БУФЕРА ОБМЕНА

ДЛЯ КАЖДОГО СИМВОЛА В ТЕКСТЕ БУФЕРА ОБМЕНА

ЕСЛИ функцияПроверкиСимвола(СИМВОЛ) ТОГДА

ДОБАВИТЬ СИМВОЛ В буфер И ВЫВЕСТИ ЕГО

ИНАЧЕ

ЕСЛИ функцияПроверкиСимвола(клавиша) ТОГДА

ЕСЛИ счетчик < размер\_буфера - 1 ТОГДА

ДОБАВИТЬ клавиша В буфер И ВЫВЕСТИ ЕГО

УВЕЛИЧИТЬ счетчик

// Функция для отображения инструкций

ФУНКЦИЯ инструкция()

ОЧИСТИТЬ ЭКРАН

ОТКРЫТЬ "instruction.txt" ДЛЯ ЧТЕНИЯ

ЕСЛИ ФАЙЛ СУЩЕСТВУЕТ ТОГДА

ЧИТАТЬ И ВЫВОДИТЬ КАЖДУЮ СТРОКУ

ИНАЧЕ

ВЫВЕСТИ СООБЩЕНИЕ ОБ ОШИБКЕ

ВЫВЕСТИ ТЕКУЩИЙ ПУТЬ ЕСЛИ ДОСТУПЕН

ВЫВЕСТИ НАЖМИТЕ ЛЮБУЮ КЛАВИШУ ДЛЯ ПРОДОЛЖЕНИЯ

ПОЛУЧИТЬ ВВОД СИМВОЛА

ОЧИСТИТЬ ЭКРАН

// Функция для открытия файла для дозаписи

ФУНКЦИЯ открытьФайлДляДозаписи(файл, полное\_имя, заголовки)

ОТКРЫТЬ файл В РЕЖИМЕ ДОБАВЛЕНИЯ

ЕСЛИ ФАЙЛ НЕ ОТКРЫТ ТОГДА

ВЫВЕСТИ СООБЩЕНИЕ ОБ ОШИБКЕ

ВЕРНУТЬСЯ

ОЧИСТИТЬ ЭКРАН

ЕСЛИ ФАЙЛ ПУСТОЙ ТОГДА

ВЫВЕСТИ ФАЙЛ СОЗДАН

ИНАЧЕ

ВЫВЕСТИ ФАЙЛ ОТКРЫТ ДЛЯ ДОБАВЛЕНИЯ

ВЫВЕСТИ ИМЯ ФАЙЛА И ИНСТРУКЦИИ ДЛЯ ВВОДА

ПОКА ИСТИНА

ДЛЯ КАЖДОГО ИЗ 3 ПОЛЕЙ

ПОЛУЧИТЬ ВВОД ДЛЯ ПОЛЯ С СООТВЕТСТВУЮЩИМИ ОГРАНИЧЕНИЯМИ

ЕСЛИ ВВОД КОРРЕКТЕН ТОГДА

СОХРАНИТЬ ВВОД

ИНАЧЕ

ВЫВЕСТИ СООБЩЕНИЕ ОБ ОШИБКЕ И ПОВТОРИТЬ ВВОД

// Функция для исходящей корреспонденции

ФУНКЦИЯ исходящаяКорреспонденция()

ПОКА ИСТИНА

ОТОБРАЗИТЬ ВАРИАНТЫ МЕНЮ

ПОЛУЧИТЬ ВЫБОР ПОЛЬЗОВАТЕЛЯ

ЕСЛИ ВЫБОР ЯВЛЯЕТСЯ ESC ТОГДА

ВЕРНУТЬСЯ

ВЫБОР ВЫБОРА

СЛУЧАЙ 'Дозаписи':

ПОЛУЧИТЬ ИМЯ ФАЙЛА

ЕСЛИ КОРРЕКТЕН ТОГДА

ВЫЗВАТЬ открытьФайлДляДозаписи()

СЛУЧАЙ 'Перезаписи':

ПОЛУЧИТЬ ИМЯ ФАЙЛА

ЕСЛИ КОРРЕКТЕН ТОГДА

ПРОВЕРИТЬ СУЩЕСТВУЕТ ЛИ ФАЙЛ

ЕСЛИ СУЩЕСТВУЕТ ТОГДА СПРОСИТЬ ПЕРЕЗАПИСАТЬ

ВЫЗВАТЬ открытьФайлДляДозаписи()

// Функция для адресов организаций

ФУНКЦИЯ адресаОрганизаций()

ПОКА ИСТИНА

ОТОБРАЗИТЬ ВАРИАНТЫ МЕНЮ

ПОЛУЧИТЬ ВЫБОР ПОЛЬЗОВАТЕЛЯ

ЕСЛИ ВЫБОР ЯВЛЯЕТСЯ ESC ТОГДА

ВЕРНУТЬСЯ

ВЫБОР ВЫБОРА

СЛУЧАЙ '1':

ПОЛУЧИТЬ ИМЯ ФАЙЛА

ЕСЛИ КОРРЕКТЕН ТОГДА

ОТКРЫТЬ ФАЙЛ ДЛЯ ДОБАВЛЕНИЯ С ОПРЕДЕЛЕННЫМИ ЗАГОЛОВКАМИ

СЛУЧАЙ '2':

ПОЛУЧИТЬ ИМЯ ФАЙЛА

ЕСЛИ КОРРЕКТЕН ТОГДА

ПРОВЕРИТЬ СУЩЕСТВУЕТ ЛИ ФАЙЛ

ЕСЛИ СУЩЕСТВУЕТ ТОГДА СПРОСИТЬ ПЕРЕЗАПИСАТЬ

ОТКРЫТЬ ФАЙЛ ДЛЯ ЗАПИСИ ЗАТЕМ ДЛЯ ДОБАВЛЕНИЯ

// Функция главного меню

ФУНКЦИЯ выборыМеню()

ПОКА ИСТИНА

ОТОБРАЗИТЬ ГЛАВНОЕ МЕНЮ

ПОЛУЧИТЬ ВЫБОР ПОЛЬЗОВАТЕЛЯ

ЕСЛИ ВЫБОР ЯВЛЯЕТСЯ ESC ТОГДА

ВЕРНУТЬСЯ

ВЫБОР ВЫБОРА

СЛУЧАЙ '1':

ЕСЛИ путь\_папки УСТАНОВЛЕН ТОГДА

ВЫЗВАТЬ исходящаяКорреспонденция()

ИНАЧЕ

ВЫВЕСТИ СООБЩЕНИЕ ОБ ОШИБКЕ

СЛУЧАЙ '2':

ЕСЛИ путь\_папки УСТАНОВЛЕН ТОГДА

ВЫЗВАТЬ адресаОрганизаций()

ИНАЧЕ

ВЫВЕСТИ СООБЩЕНИЕ ОБ ОШИБКЕ

// Функция для нормализации пути

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

ЕСЛИ путь ПУСТОЙ ТОГДА

ПОЛУЧИТЬ ТЕКУЩИЙ РАБОЧИЙ КАТАЛОГ

ИНАЧЕ

ПОЛУЧИТЬ ПОЛНЫЙ ПУТЬ пути

УДАЛИТЬ ЛИШНИЕ СЛЕШИ

УБЕДИТЬСЯ, ЧТО ПУТЬ НЕ ЗАКАНЧИВАЕТСЯ СЛЕШОМ, ЕСЛИ ЭТО НЕ КОРНЕВОЙ

// Функция для установки пути программы

ФУНКЦИЯ путьПрограммы()

ПОЛУЧИТЬ НОВЫЙ ПУТЬ ОТ ПОЛЬЗОВАТЕЛЯ

НОРМАЛИЗОВАТЬ ПУТЬ

ПРОВЕРИТЬ СУЩЕСТВУЕТ ЛИ ПУТЬ И ЯВЛЯЕТСЯ ЛИ КАТАЛОГОМ

ЕСЛИ КОРРЕКТЕН ТОГДА

ПОПЫТАТЬСЯ СОЗДАТЬ ВРЕМЕННЫЙ ФАЙЛ ДЛЯ ПРОВЕРКИ ПРАВ НА ЗАПИСЬ

ЕСЛИ УСПЕШНО ТОГДА

УСТАНОВИТЬ путь\_папки В НОРМАЛИЗОВАННЫЙ ПУТЬ

ИНАЧЕ

ВЫВЕСТИ СООБЩЕНИЕ ОБ ОШИБКЕ

ИНАЧЕ

ВЫВЕСТИ СООБЩЕНИЕ ОБ ОШИБКЕ

// Основной цикл меню

ФУНКЦИЯ меню()

УСТАНОВИТЬ ПУТЬ ПО УМОЛЧАНИЮ ЕСЛИ НЕ УСТАНОВЛЕН

ПОКА ИСТИНА

ОТОБРАЗИТЬ ГЛАВНОЕ МЕНЮ

ПОЛУЧИТЬ ВЫБОР ПОЛЬЗОВАТЕЛЯ

ВЫБОР ВЫБОРА

СЛУЧАЙ '1':

ЕСЛИ путь\_папки УСТАНОВЛЕН ТОГДА

ВЫЗВАТЬ выборыМеню()

ИНАЧЕ

ВЫВЕСТИ СООБЩЕНИЕ ОБ ОШИБКЕ

СЛУЧАЙ '2':

ВЫЗВАТЬ путьПрограммы()

СЛУЧАЙ '3':

ВЫЗВАТЬ инструкция()

СЛУЧАЙ ESC:

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

В программе запрашиваются следующие входные данные:

· **Путь к папке для сохранения файлов:** Строка (любые символы, кроме запрещённых в именах файлов/папок, и не должна начинаться с кириллицы). Путь может быть абсолютным или относительным. Запрашивается в функции program\_way.

· **Имена файлов (без расширения):** В функциях outgoing\_correspondence и organization\_addresses, после выбора действия ("Создать/добавить" или "Перезаписать"), запрашивается строка - имя файла.

· **Поля данных (в зависимости от выбранного типа данных):**

**Для файла отправляемой корреспонденции:**

* Вид корреспонденции: Буквы, цифры, пробел, - \_ . , ( ).
* Дата подготовки: Цифры, ., буквы, пробел (для "нет данных").
* Название организации: Буквы, цифры, пробел, - \_ . , " ( ) № + ! & : « » #.

**Для файла адресов организаций:**

* Название организации: Буквы, цифры, пробел, - \_ . , " ( ) № + ! & : « » #.
* Адрес организации: Буквы, цифры, пробел, - , . / №.
* Фамилия руководителя: Буквы, пробел, - (тире).

Программа формирует два выходных файла в формате текстовых файлов, с разделителем полей в виде точки с запятой с пробелом “; ” .

**Файл отправляемой корреспонденции:**

Имя файла: IC\_<имя, введенное пользователем>.txt

Каждая строка файла представляет собой одну запись и содержит следующие поля, разделенные точкой с запятой:

* + Вид корреспонденции
  + Дата подготовки
  + Название организации

**Файл адресов организаций:**

Имя файла: AO\_<имя, введенное пользователем>.txt

Каждая строка файла представляет собой одну запись и содержит следующие поля, разделенные точкой с запятой:

* + Название организации
  + Адрес организации
  + Фамилия руководителя

# Описание программных единиц

**Константы:**

* const char\* file\_extension = ".txt"; // Расширение файла
* const char\* ocfe = "IC\_"; // Префикс для файлов исходящей корреспонденции
* const char\* oa = "AO\_"; // Префикс для файлов адресов организаций

**Глобальные переменные:**

* char folder\_way[256] = { 0 }; // Путь к папке, инициализируем пустой строкой
* const char\* file\_extension; // Расширение файлов.
* const char\* ocfe; // Префикс для файлов исходящей корреспонденции.
* const char\* oa; // Префикс для файлов адресов организаций

**Описание функций программы**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название функции** | **Назначение функции** | **Параметры функции** | **Возвращаемое значение функции** |
| is\_leap(int year) | Проверка на високосность года. | int year – год. | bool: true, если год високосный, false – в противном случае. |
| isValidDate(const char\* dateStr) | Проверка формата даты (ДД.ММ.ГГГГ или "нет данных"). | const char\* dateStr – строка с датой. | bool: true, если дата корректна, false – в противном случае. |
| isValidFileName(const char\* fileName) | Проверка допустимости символов в имени файла. | const char\* fileName – имя файла. | bool: true, если имя файла допустимо, false – в противном случае. |
| instruction() | Отображение инструкции из "instruction.txt". | (нет) | void: Ничего не возвращает. |
| getLineWithEsc(const char\* instruction, char\* buffer, int buffer\_size) | Ввод строки с обработкой Esc/Backspace/Enter. | const char\* instruction - приглашение, char\* buffer - буфер, int buffer\_size - размер буфера | void: Ничего не возвращает. |
| normalizePath(const char\* path, char\* normalized\_path, size\_t normalized\_path\_size) | Нормализация пути. | const char\* path - исходный путь, char\* normalized\_path - буфер для результата, size\_t normalized\_path\_size - размер буфера | void: Ничего не возвращает. |
| openFileForAppend(FILE\*& file, const char\* full\_name, const char\* headers) | Открытие файла для добавления и ввод/запись данных. | FILE\*& file - файл, const char\* full\_name - имя файла, const char\* headers - заголовки. | void: Ничего не возвращает. |
| outgoing\_correspondence() | Работа с исходящей корреспонденцией. | (нет) | void: Ничего не возвращает. |
| organization\_addresses() | Работа с адресами организаций. | (нет) | void: Ничего не возвращает. |
| menu\_choises() | Меню выбора типа данных. | (нет) | void: Ничего не возвращает. |
| program\_way() | Выбор пути к папке. | (нет) | void: Ничего не возвращает. |
| menu() | Главное меню программы. | (нет) | void: Ничего не возвращает. |

# Инструкция пользователю

1. **Запуск программы:**
   * Запустите исполняемый файл программы.
   * После запуска появится главное меню программы.
2. **Выбор пути к папке (необязательно):**
   * Программа автоматически определит путь к исполняемому файлу.
   * Если вы хотите изменить папку, в которой будут храниться файлы, выберите в главном меню пункт "2" (Изменить путь к папке).
   * Введите путь к папке и нажмите Enter. Убедитесь, что путь существует, и у программы есть права на запись в эту папку.
   * Если путь не будет указан, программа будет использовать последний доступный путь.
3. **Главное меню:**
   * Выберите один из пунктов меню, нажав соответствующую цифру:
     + "1" - Начать работу (переход к выбору типа данных).
     + "2" - Изменить путь к папке (см. пункт 2).
     + "3" - Инструкция (отображение инструкции из файла instruction.txt).
     + "Esc" - Выход из программы.
4. **Выбор типа данных (после выбора "1" в главном меню):**
   * Выберите тип данных, с которым вы хотите работать:
     + "1" - Исходящая корреспонденция (создание/дополнение/перезапись файла IC\_<имя>.txt).
     + "2" - Адреса организаций (создание/дополнение/перезапись файла AO\_<имя>.txt).
     + "Esc" - Возврат в главное меню.
5. **Работа с файлами (после выбора типа данных):**
   * Выберите действие:
     + "1" - Создать/открыть файл для дозаписи:
       - Введите имя файла (без расширения .txt). Программа добавит префикс IC\_ или AO\_ и расширение .txt автоматически.
       - Если файл с таким именем существует, он будет открыт для добавления новых записей в конец.
       - Если файл не существует, он будет создан.
     + "2" - Перезаписать файл:
       - Введите имя файла (без расширения .txt).
       - Если файл с таким именем существует, программа запросит подтверждение на перезапись. Если вы подтвердите перезапись, существующее содержимое файла будет удалено.
       - Если файл не существует, он будет создан.
     + "Esc" - Возврат в меню выбора типа данных.
6. **Ввод данных (после выбора действия с файлом):**
   * Программа будет запрашивать ввод данных для каждого поля записи. Поля зависят от выбранного типа данных (см. пункт 4).
   * **Ввод значения:** Введите значение поля и нажмите Enter.
   * **Переход к следующему полю:** Нажмите Tab.
     + Нажатие Tab без ввода значения приведёт к записи пустого значения для текущего поля.
   * **Отмена ввода/Возврат:** Нажмите Esc в любой момент, чтобы отменить ввод данных для текущей записи и вернуться в предыдущее меню.
   * **Специальные клавиши:**
     + Backspace: Удаление последнего введенного символа.
     + Enter: Подтверждение ввода значения текущего поля.
     + Esc: Отмена ввода данных/возврат в предыдущее меню
     + Tab: Переход к следующему полю
7. **Завершение работы:**
   * 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”:

ИНСТРУКЦИЯ ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММЫ

Программа для хранения данных об исходящей корреспонденции и адресах организаций (файлы .txt).

Требования: (Разрешенные символы)

Путь к папке: Путь должен существовать.

Имена файлов: Латиница, цифры, \_, -. Расширение .txt добавлять не нужно.

Дата подготовки: Формат ДД.ММ.ГГГГ или "нет данных".

Фамилия руководителя: Буквы (рус/лат), пробел, дефис или "нет данных".

Другие поля: Любые символы или "нет данных".

Работа с программой:

Запуск: При запуске определяется путь определяется как текущая папка.

Главное меню:

Начать работу: Выбор типа данных (корреспонденция/адреса).

Изменить путь: Указать другую папку.

Инструкция: Это руководство.

Выход (Esc): Закрыть программу.

Выбор типа данных:

Исходящая корреспонденция (IC\_\*\*\*.txt).

Адреса организаций (AO\_\*\*\*.txt).

Выход (Esc): Назад.

Управление файлами:

Создать/открыть (дозапись): Создать и открыть файл или записывать в конец существующего.

Перезаписать: Удалить содержимое файла (с защитой от случайного нажатия) и начать запись с начала.

Выход (Esc): Назад.

Ввод данных:

Программа указывает, что вводить.

Tab и Enter: Следующее поле.

Backspace: Удалить символ.

Esc: Если вы ввели текст, то нажатие на esc сотрет его полностью. Если поле пустое то вернет вас назад.

Если поле пустое, введите "нет данных".

# Листинг программной разработки

Главный файл

// Lab\_prog\_1.cpp

#include "Lab\_prog\_header\_1.h"

int main()

{

// Устанавливаем кодировку консоли для корректного отображения кириллицы

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Вызываем главную функцию меню

menu();

return 0;

}

Файл заголовка  
//Lab\_prog\_header\_1.h

#pragma once

#include <stdio.h> // Ввод-вывод: printf, fprintf, fopen, fclose, fgets, sscanf, snprintf

#include <stdlib.h> // Стандартная библиотека: system, malloc, free, atoi (если потребуется)

#include <string.h> // Операции со строками: strcpy, strncpy, strcmp, strlen, strncat, strtok, strrchr

#include <conio.h> // Консольный ввод-вывод: \_getch, \_kbhit

#include <ctype.h> // Функции для работы с символами: isalpha, tolower, isdigit

#include <sys/stat.h> // Для stat: определение типа файла/папки

#include <direct.h> // Для \_mkdir (Windows) и \_getcwd: создание папок и получение текущего пути

#include <Windows.h> // Для SetConsoleCP, SetConsoleOutputCP, буфера обмена: установка кодировки консоли

#include <io.h> // Для \_access: проверка существования файла

// Глобальные переменные

// Размер 256 выбран как достаточное, но не чрезмерное значение для большинства путей к файлам и именам файлов.

// Это компромисс между возможностью обрабатывать длинные пути и экономией памяти.

// Также, размер буфера ограничен константой MAX\_PATH (260 символов), определенной в Windows API.

extern char folder\_way[256];

extern const char\* file\_extension;

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

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

// Объявления функций

bool is\_leap(int year);

bool isValidDate(const char\* dateStr);

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

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

void instruction();

void outgoing\_correspondence();

void organization\_addresses();

void menu\_choises();

void program\_way();

void menu();

void getLineWithRestrictedChars(const char\* instruction, char\* buffer, int buffer\_size, bool (\*isValidCharFunc)(char)); // Функция ввода с проверкой символов

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

void openFileForAppend(FILE\*& file, const char\* full\_name, const char\* headers); // Убраны mode и mode\_size, они не используются

// Макрос для безопасного копирования строк. Он аналогичен strncpy, но гарантирует,

// что строка-результат будет завершена нуль-терминатором ('\0'), даже если исходная

// строка слишком длинная и обрезается. Это предотвращает переполнение буфера и

// связанные с ним ошибки. `do { ... } while(0)` используется для того, чтобы

// макрос можно было использовать в любом месте, где допустим оператор, например,

// в теле `if` без фигурных скобок.

#define SAFE\_STRCPY(dest, src, size) do { strncpy(dest, src, size - 1); dest[size - 1] = '\0'; } while(0)

Файл реализации библиотеки

//Lab\_prog\_header\_1.cpp

#include "Lab\_prog\_header\_1.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

// Глобальные переменные (определения)

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

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; // Для корректной работы с кириллицей

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 для получения строки с консоли с фильтрацией символов

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

printf("%s", instruction);

int i = 0;

buffer[0] = '\0'; // Инициализация буфера

while (true) {

int key = \_getch();

if (key == 0 || key == 224) { // Расширенные клавиши

if (\_kbhit()) { // Проверяем, есть ли второй байт

\_getch(); // Игнорируем второй байт

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 if (key == 22) { // Ctrl+V

if (OpenClipboard(NULL)) {

HANDLE hData = GetClipboardData(CF\_TEXT);

if (hData != NULL) {

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

if (pszText != NULL) {

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

// Проверяем вставляемый символ с помощью переданной функции

if (isValidCharFunc(pszText[j])) {

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

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

}

}

GlobalUnlock(hData);

}

}

CloseClipboard();

}

}

else { // Обычный символ

char c = (char)key;

// Проверяем символ с помощью переданной функции

if (isValidCharFunc(c)) {

if (i < buffer\_size - 1) {

buffer[i] = c;

printf("%c", c);

i++;

}

}

// Если символ невалидный, ничего не делаем (не печатаем, не добавляем)

}

}

}

// Функция проверки года на високосность

bool is\_leap(int year) {

return (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0));

}

// Функция проверки корректности даты

bool isValidDate(const char\* dateStr) {

if (strcmp(dateStr, "нет данных") == 0) {

return true; // "нет данных" всегда валидно

}

// Проверка формата ДД.ММ.ГГГГ (длина и разделители)

if (strlen(dateStr) != 10 || dateStr[2] != '.' || dateStr[5] != '.') {

return false; // Неправильный формат

}

int day, month, year;

// Проверка, что все символы - цифры или точки

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

if (!isdigit(dateStr[i]) && dateStr[i] != '.') {

return false; // Неправильный формат

}

}

// Пытаемся разобрать дату

int count = sscanf(dateStr, "%d.%d.%d", &day, &month, &year);

if (count != 3) {

return false; // Неправильный формат

}

// Проверка корректности значений дня, месяца, года

if (year < 0 || month < 1 || month > 12 || day < 1) {

return false; // Неправильная дата

}

int daysInMonth[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

if (is\_leap(year)) {

daysInMonth[2] = 29;

}

if (day > daysInMonth[month]) {

return false; // Неправильная дата (день)

}

return true; // Дата корректна

}

// Функция вывода инструкции

void instruction() {

system("cls");

FILE\* file = fopen("instruction.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 'instruction.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");

}

// Функция открытия файла для дозаписи (или создания, если не существует)

// Убраны параметры mode, т.к. режим всегда "a" (append)

void openFileForAppend(FILE\*& file, const char\* full\_name, const char\* headers) {

file = fopen(full\_name, "a");

if (!file) {

printf("Не удалось открыть файл для записи. Возможно, файл занят другой программой, или у вас нет прав доступа.\n");

printf("Для продолжения нажмите Enter.");

system("PAUSE>nul");

return;

}

system("cls");

if (ftell(file) == 0) { // Проверяем, пустой ли файл (был только что создан)

printf("Файл '%s' создан.\n\n", full\_name);

fflush(file);

}

else {

printf("Файл '%s' открыт для дозаписи.\n\n", full\_name);

}

printf("Путь к файлу: %s\n\n", full\_name);

printf("Нажмите TAB или Enter для перехода к следующему параметру.\n");

printf("Нажмите ESC для выхода из режима ввода.\n");

printf("Вы можете ввести 'нет данных' в любое поле (кроме даты - там только 'нет данных').\n");

// Размер 256 выбран как компромисс между возможностью обрабатывать длинные строки

// и экономией памяти, согласуется с размером буфера MAX\_PATH в Windows.

char fields[3][256];

int current\_col = 0;

while (true) {

for (current\_col = 0; current\_col < 3; current\_col++) {

int i = 0;

memset(fields[current\_col], 0, sizeof(fields[current\_col]));

// Вывод названия текущего поля

if (current\_col == 0) {

if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) printf("\nВид корреспонденции: "); else printf("\nНазвание организации: ");

}

else if (current\_col == 1) {

if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) printf("\nДата подготовки (ДД.ММ.ГГГГ или нет данных): "); else printf("\nАдрес: ");

}

else if (current\_col == 2) {

if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) printf("\nНазвание организации: "); else printf("\nФамилия руководителя: ");

}

// Цикл ввода для текущего поля с проверкой символов

while (true) {

int key = \_getch();

char c = (char)key;

bool allowed = false;

if (key == 0 || key == 224) { // Расширенные клавиши

if (\_kbhit()) { // Проверяем, есть ли второй байт

\_getch(); // Игнорируем второй байт

continue;

}

}

// Определяем, разрешен ли символ для текущего поля

unsigned char uc = (unsigned char)c; // Для корректной работы с кириллицей

bool is\_letter\_ru = (uc >= 'А' && uc <= 'Я') || (uc >= 'а' && uc <= 'я') || (uc >= 192); // Кириллица

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

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

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

if (current\_col == 0) { // Вид корреспонденции / Название организации (первое поле)

if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) { // Вид корреспонденции

allowed = is\_letter\_ru || is\_letter\_en || is\_digit || is\_space || uc == '-' || uc == '\_' || uc == '.' || uc == ',' || uc == '(' || uc == ')' || uc == 211;

}

else { // Название организации (из адресов)

allowed = is\_letter\_ru || is\_letter\_en || 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)

}

}

else if (current\_col == 1) { // Дата подготовки / Адрес

if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) { // Дата подготовки

allowed = is\_digit || uc == '.' || is\_letter\_ru || is\_space; // Разрешаем буквы и пробел для "нет данных"

}

else { // Адрес

allowed = is\_letter\_ru || is\_letter\_en || is\_digit || is\_space || uc == '-' || uc == ',' || uc == '.' || uc == '/' || uc == 211; // №(211)

}

}

else if (current\_col == 2) { // Название организации / Фамилия руководителя

if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) { // Название организации (из корреспонденции)

allowed = is\_letter\_ru || is\_letter\_en || 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)

}

else { // Фамилия руководителя

allowed = is\_letter\_ru || is\_letter\_en || uc == '-' || is\_space;

}

}

// Обработка клавиш управления

if (key == 27) { // ESC

if (file) fclose(file);

printf("\nВвод прерван.\n");

return;

}

else if (key == '\t' || key == '\r') { // Tab or Enter

fields[current\_col][i] = '\0';

// Проверка даты после ввода

if (current\_col == 1 && strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) {

if (!isValidDate(fields[current\_col])) {

// Определяем тип ошибки даты

bool format\_error = true;

if (strlen(fields[current\_col]) == 10 && fields[current\_col][2] == '.' && fields[current\_col][5] == '.') {

int d, m, y;

if (sscanf(fields[current\_col], "%d.%d.%d", &d, &m, &y) == 3) {

format\_error = false; // Формат верный, значит дата некорректна

}

}

if (format\_error) {

printf("\nНеправильный формат даты. Введите дату в формате 'ДД.ММ.ГГГГ' (например, 01.01.2023) или введите 'нет данных'.\n");

}

else {

printf("\nНеправильная дата. Проверьте, что введенные день, месяц и год существуют. Например, 31.02.2023 - некорректная дата.\n");

}

i = 0;

memset(fields[current\_col], 0, sizeof(fields[current\_col]));

// Заново выводим приглашение для ввода даты

printf("\nДата подготовки (ДД.ММ.ГГГГ или нет данных): ");

continue; // Продолжаем ввод для этого же поля

}

}

// Проверка на пустой ввод

if (i == 0 && strcmp(fields[current\_col], "нет данных") != 0 && !(current\_col == 1 && strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0 && fields[current\_col][0] == '\0'))

{

printf("\nВы не ввели никаких данных. Пожалуйста, введите информацию или укажите 'нет данных'.\n");

// Заново выводим приглашение для ввода

if (current\_col == 0) { if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) printf("\nВид корреспонденции: "); else printf("\nНазвание организации: "); }

else if (current\_col == 1) { if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) printf("\nДата подготовки (ДД.ММ.ГГГГ или нет данных): "); else printf("\nАдрес: "); }

else if (current\_col == 2) { if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) printf("\nНазвание организации: "); else printf("\nФамилия руководителя: "); }

continue; // Продолжаем ввод для этого же поля

}

break; // Переход к следующему полю

}

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

if (i > 0) {

i--;

printf("\b \b");

fields[current\_col][i] = '\0';

}

}

else if (key == 22) { // Ctrl+V

if (OpenClipboard(NULL)) {

HANDLE hData = GetClipboardData(CF\_TEXT);

if (hData != NULL) {

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

if (pszText != NULL) {

for (int j = 0; pszText[j] != '\0' && i < sizeof(fields[current\_col]) - 1; ++j) {

// Проверяем вставляемый символ на допустимость для ТЕКУЩЕГО ПОЛЯ

unsigned char uct = (unsigned char)pszText[j];

bool allowed\_paste = false;

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

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

bool is\_space\_p = (uct == ' ');

if (current\_col == 0) { if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) { allowed\_paste = is\_letter\_p || is\_digit\_p || is\_space\_p || uct == '-' || uct == '\_' || uct == '.' || uct == ',' || uct == '(' || uct == ')'; } else { allowed\_paste = is\_letter\_p || is\_digit\_p || is\_space\_p || uct == '-' || uct == '\_' || uct == '.' || uct == ',' || uct == '(' || uct == ')' || uct == 211 || uct == '+' || uct == '!' || uct == '&' || uct == ':' || uct == 171 || uct == 187 || uct == '#'; } }

else if (current\_col == 1) { if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) { allowed\_paste = is\_digit\_p || uct == '.' || is\_letter\_p || is\_space\_p; } else { allowed\_paste = is\_letter\_p || is\_digit\_p || is\_space\_p || uct == '-' || uct == ',' || uct == '.' || uct == '/' || uct == 211; } }

else if (current\_col == 2) { if (strcmp(headers, "Вид корреспонденции; Дата подготовки; Название организации;\n") == 0) { allowed\_paste = is\_letter\_p || is\_digit\_p || is\_space\_p || uct == '-' || uct == '\_' || uct == '.' || uct == ',' || uct == '(' || uct == ')' || uct == 211 || uct == '+' || uct == '!' || uct == '&' || uct == ':' || uct == 171 || uct == 187 || uct == '#'; } else { allowed\_paste = is\_letter\_p || uct == '-' || is\_space\_p; } }

if (allowed\_paste) {

fields[current\_col][i++] = pszText[j];

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

}

}

GlobalUnlock(hData);

}

}

CloseClipboard();

}

}

else { // Обычный символ

if (allowed && i < sizeof(fields[current\_col]) - 1) {

fields[current\_col][i] = c;

printf("%c", c);

i++;

}

// Если символ не разрешен (allowed == false), ничего не делаем

}

} // конец while(true) для ввода одного поля

} // конец for по полям (current\_col)

if (strcmp(fields[0], "нет данных") == 0 &&

strcmp(fields[1], "нет данных") == 0 &&

strcmp(fields[2], "нет данных") == 0)

{

// Если все поля "нет данных", выводим сообщение и не записываем

printf("\nВсе поля содержат 'нет данных'. Запись не будет добавлена.\n");

}

else

{

// Если хотя бы одно поле не "нет данных", записываем строку

fprintf(file, "%s; %s; %s;\n", fields[0], fields[1], fields[2]);

fflush(file); // Сбрасываем буфер файла на диск (для надежности)

printf("\nДанные сохранены. Введите следующую запись или нажмите ESC для выхода.\n");

}

}

}

// Функция для работы с исходящей корреспонденцией

void outgoing\_correspondence() {

char txt\_name[256];

char full\_name[512];

do {

system("cls");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("Исходящая корреспонденция:\n");

printf("1. Создать/открыть файл для дозаписи\n");

printf("2. Перезаписать файл\n");

printf("Esc. Выход в выбор типа файла\n\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

printf("Выбранный путь к папке: %s\n\n", folder\_way);

char menu\_out\_case = \_getch();

if (menu\_out\_case == 27) {

return;

}

FILE\* file;

switch (menu\_out\_case) {

case '1': {

printf("\n");

// Используем getLineWithRestrictedChars с проверкой для имени файла

getLineWithRestrictedChars("Введите имя файла: ", txt\_name, sizeof(txt\_name), isValidFileNameChar);

if (txt\_name[0] == '\0') { // Если пользователь нажал Esc

break;

}

// Дополнительная проверка на пустой ввод после getLine...

if (strlen(txt\_name) == 0) {

printf("Имя файла не может быть пустым.\n");

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

break;

}

snprintf(full\_name, sizeof(full\_name), "%s%s%s%s", folder\_way, ocfe, txt\_name, file\_extension);

openFileForAppend(file, full\_name, "Вид корреспонденции; Дата подготовки; Название организации;\n");

break;

}

case '2': {

printf("\n");

getLineWithRestrictedChars("Введите имя файла для перезаписи: ", txt\_name, sizeof(txt\_name), isValidFileNameChar);

if (txt\_name[0] == '\0') { // Esc

break;

}

if (strlen(txt\_name) == 0) {

printf("Имя файла не может быть пустым.\n");

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

break;

}

snprintf(full\_name, sizeof(full\_name), "%s%s%s%s", folder\_way, ocfe, txt\_name, file\_extension);

bool exists = (\_access(full\_name, 0) == 0);

if (exists) {

printf("Файл с таким именем найден. Перезаписать? (y/n): ");

char overwriteConfirm = \_getch();

printf("\n"); // Перевод строки после \_getch

if (tolower(overwriteConfirm) != 'y') {

printf("Файл не был перезаписан.\n");

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

break;

}

}

file = fopen(full\_name, "w"); // Открываем для перезаписи ("w")

if (file) {

fclose(file); // Закрываем, чтобы openFileForAppend мог открыть в "a"

printf("Файл '%s' успешно %s.\n", full\_name, exists ? "перезаписан" : "создан");

openFileForAppend(file, full\_name, "Вид корреспонденции; Дата подготовки; Название организации;\n");

}

else {

printf("Не удалось открыть файл для записи. Возможно, файл занят другой программой, или у вас нет прав доступа.\n");

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

}

break;

}

default:

// Некорректный ввод в меню обрабатывается циклом do-while

break;

}

} while (true);

}

// Функция для работы с адресами организаций

void organization\_addresses() {

char txt\_name[256];

char full\_name[512];

do {

system("cls");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("Адреса организаций:\n");

printf("1. Создать/открыть файл для дозаписи\n");

printf("2. Перезаписать файл\n");

printf("Esc. Выход в выбор типа файла\n\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

printf("Выбранный путь к папке: %s\n\n", folder\_way);

char menu\_out\_case = \_getch();

if (menu\_out\_case == 27) {

return;

}

FILE\* file;

switch (menu\_out\_case) {

case '1': {

printf("\n");

getLineWithRestrictedChars("Введите имя файла: ", txt\_name, sizeof(txt\_name), isValidFileNameChar);

if (txt\_name[0] == '\0') { // Esc

break;

}

if (strlen(txt\_name) == 0) {

printf("Имя файла не может быть пустым.\n");

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

break;

}

snprintf(full\_name, sizeof(full\_name), "%s%s%s%s", folder\_way, oa, txt\_name, file\_extension);

openFileForAppend(file, full\_name, "Название организации; Адрес; Фамилия руководителя;\n");

break;

}

case '2': {

printf("\n");

getLineWithRestrictedChars("Введите имя файла для перезаписи: ", txt\_name, sizeof(txt\_name), isValidFileNameChar);

if (txt\_name[0] == '\0') { // Esc

break;

}

if (strlen(txt\_name) == 0) {

printf("Имя файла не может быть пустым.\n");

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

break;

}

snprintf(full\_name, sizeof(full\_name), "%s%s%s%s", folder\_way, oa, txt\_name, file\_extension);

bool exists = (\_access(full\_name, 0) == 0);

if (exists) {

printf("Файл с таким именем найден. Перезаписать? (y/n): ");

char overwriteConfirm = \_getch();

printf("\n");

if (tolower(overwriteConfirm) != 'y') {

printf("Файл не был перезаписан.\n");

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

break;

}

}

file = fopen(full\_name, "w");

if (file) {

fclose(file);

printf("Файл '%s' успешно %s.\n", full\_name, exists ? "перезаписан" : "создан");

openFileForAppend(file, full\_name, "Название организации; Адрес; Фамилия руководителя;\n");

}

else {

printf("Не удалось открыть файл для записи. Возможно, файл занят другой программой, или у вас нет прав доступа.\n");

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

}

break;

}

default:

break;

}

} while (true);

}

// Функция меню выбора типа данных

void menu\_choises() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

do {

system("cls");

printf("Меню выбора типа данных:\n");

printf("1. Исходящая корреспонденция\n");

printf("2. Адреса организаций\n");

printf("Esc. Выход в меню\n\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

printf("Выбранный путь к папке: %s\n", folder\_way);

char menu\_out = \_getch();

if (menu\_out == 27) {

system("cls");

return;

}

switch (menu\_out) {

case '1':

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

outgoing\_correspondence();

}

else {

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

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

}

break;

case '2':

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

organization\_addresses();

}

else {

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

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

}

break;

default:

break;

}

} while (true);

}

// Функция нормализации пути (упрощенная для Windows)

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';

}

return;

}

// Используем \_fullpath для нормализации под Windows

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

// Ошибка нормализации (например, путь слишком длинный или недопустимый)

// В normalized\_path может быть мусор, очистим его

SAFE\_STRCPY(normalized\_path, path, normalized\_path\_size); // Копируем как есть для дальнейшей проверки

// Можно добавить вывод ошибки, но stat потом все равно проверит

}

// Дополнительно убираем повторяющиеся слеши, т.к. \_fullpath их может оставить

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';

// Убираем завершающий слеш, если он не является частью корневого пути (C:\)

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

if (len > 3 && normalized\_path[len - 1] == '\\') {

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

}

}

// Функция выбора пути к папке

void program\_way() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("\n");

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

char normalized\_path\_temp[256]; // Временный буфер для нормализации

// Ввод пути с проверкой символов

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

if (folder\_way\_new[0] == '\0') { // Пользователь нажал Esc

return;

}

// Нормализация пути

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

// Проверка существования пути и прав доступа

struct stat path\_stat;

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

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

// Проверка прав на запись (пытаемся создать и удалить тестовый файл)

char test\_file\_path[512];

// Добавляем слеш к нормализованному пути для snprintf, если его нет

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

char path\_with\_slash[257]; // +1 для слеша и +1 для нуля

SAFE\_STRCPY(path\_with\_slash, normalized\_path\_temp, sizeof(path\_with\_slash));

if (len\_norm > 0 && path\_with\_slash[len\_norm - 1] != '\\') {

if (len\_norm < sizeof(path\_with\_slash) - 1) {

path\_with\_slash[len\_norm] = '\\';

path\_with\_slash[len\_norm + 1] = '\0';

}

else {

// Путь слишком длинный для добавления слеша, вероятно ошибка

printf("Указанный путь не существует. Проверьте правильность написания и убедитесь, что папка существует.\n");

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

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

return;

}

}

snprintf(test\_file\_path, sizeof(test\_file\_path), "%stest\_access.tmp", path\_with\_slash); // Используем временное расширение

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

if (test\_file) { // Права на запись есть

fclose(test\_file);

remove(test\_file\_path); // Удаляем тестовый файл

printf("Путь к папке выбран: %s\n", path\_with\_slash);

SAFE\_STRCPY(folder\_way, path\_with\_slash, sizeof(folder\_way)); // Сохраняем путь со слешем

}

else { // Нет прав на запись

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 { // Путь существует, но это не директория

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

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

}

}

else { // Путь не существует

printf("Указанный путь не существует. Проверьте правильность написания и убедитесь, что папка существует.\n");

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

}

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

}

// Главное меню программы

void menu() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Автоматическое определение пути при запуске

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

char current\_path[256];

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

{

// Добавляем слеш в конце

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

if (len > 0 && current\_path[len - 1] != '\\' && len < sizeof(folder\_way) - 1) {

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

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

}

SAFE\_STRCPY(folder\_way, current\_path, sizeof(folder\_way));

}

else

{

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

}

}

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

while (true) {

system("cls");

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

printf("\* Программа для ведения базы данных о корреспонденции \*\n");

printf("\* и адресах организаций. \*\n");

printf("\* Позволяет добавлять, изменять и удалять данные. \*\n");

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

printf("Главное меню:\n");

printf("1. Начать работу\n");

printf("2. Изменить путь к папке\n");

printf("3. Инструкция\n");

printf("Esc. Выход\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

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

char c = \_getch();

switch (c) {

case '1':

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

menu\_choises();

}

else {

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

printf("Для продолжения нажмите Enter."); system("PAUSE>nul");

}

break;

case '2':

program\_way();

break;

case '3':

instruction();

break;

case 27: // ESC

return;

default:

break;

}

}

}

# Результаты тестирования программной разработки

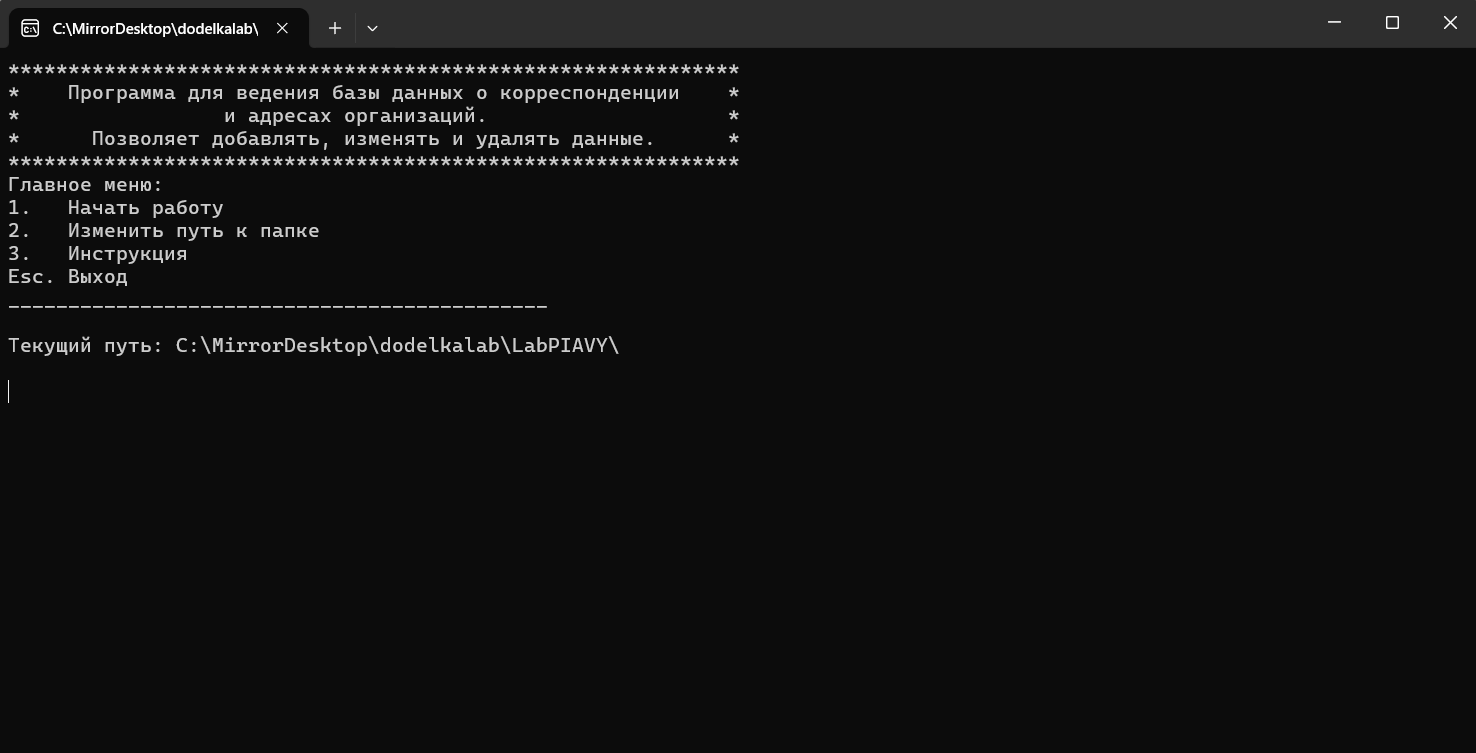
При запуске программы выводится главное меню (рис. 1). Для нажатия доступны только цифры 1, 2, 3, esc. Остальные символы при нажатии игнорируются.  


Рисунок 1 Главное меню

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

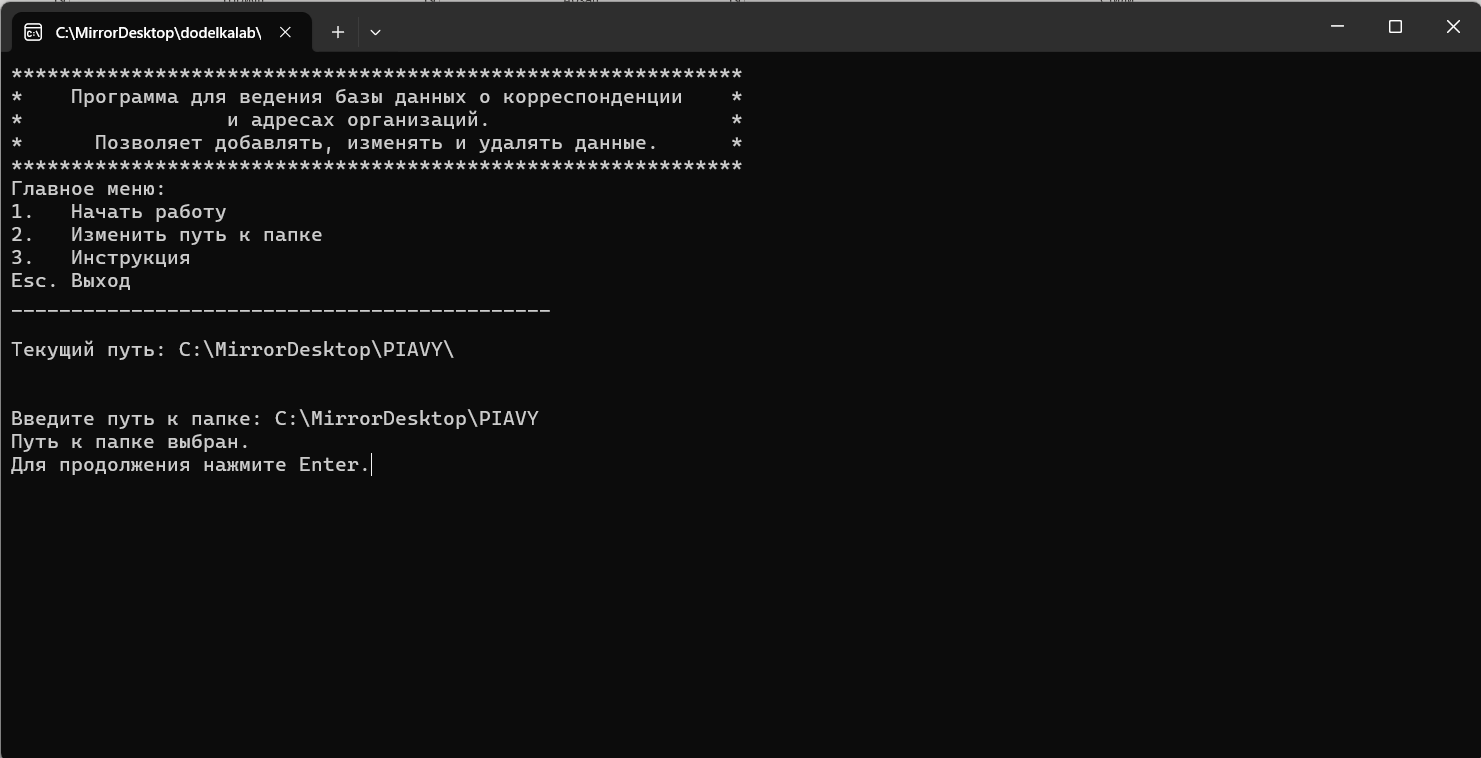
****

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).

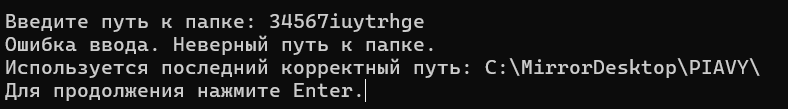


Рисунок 3 Некорректный ввод пути к папке хранения файлов.

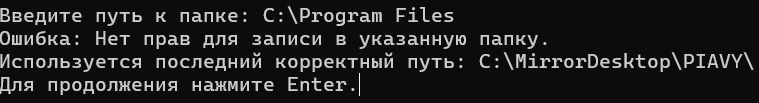


Рисунок 4 Программа уведомляет если у нее нет прав записывать в папку.

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

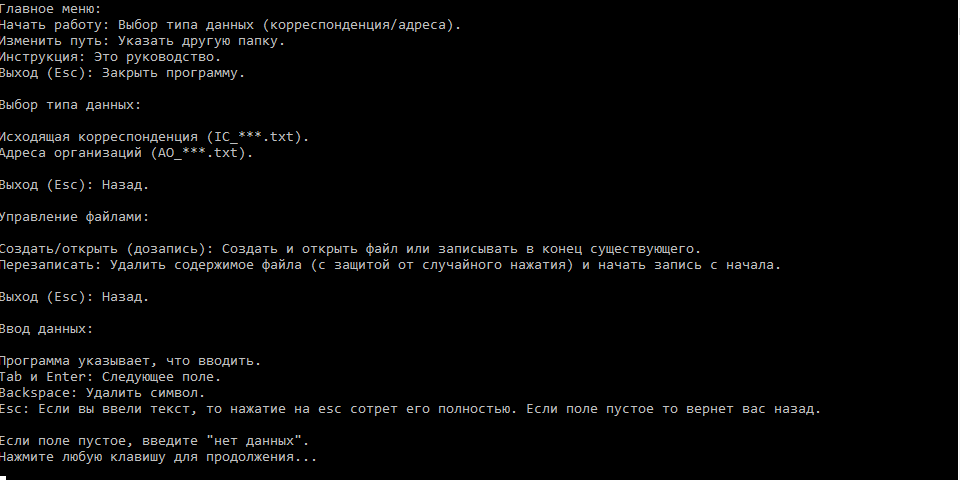
****

Рисунок 5 При выборе инструкции.

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

В меню выбора типа данных доступен выбор ввода исходящей корреспонденции или адресов организации. Последующие интерфейсы при работе с данными идентичны, поэтому будет рассмотрена работа с исходящей корреспонденцией (рис. 7).

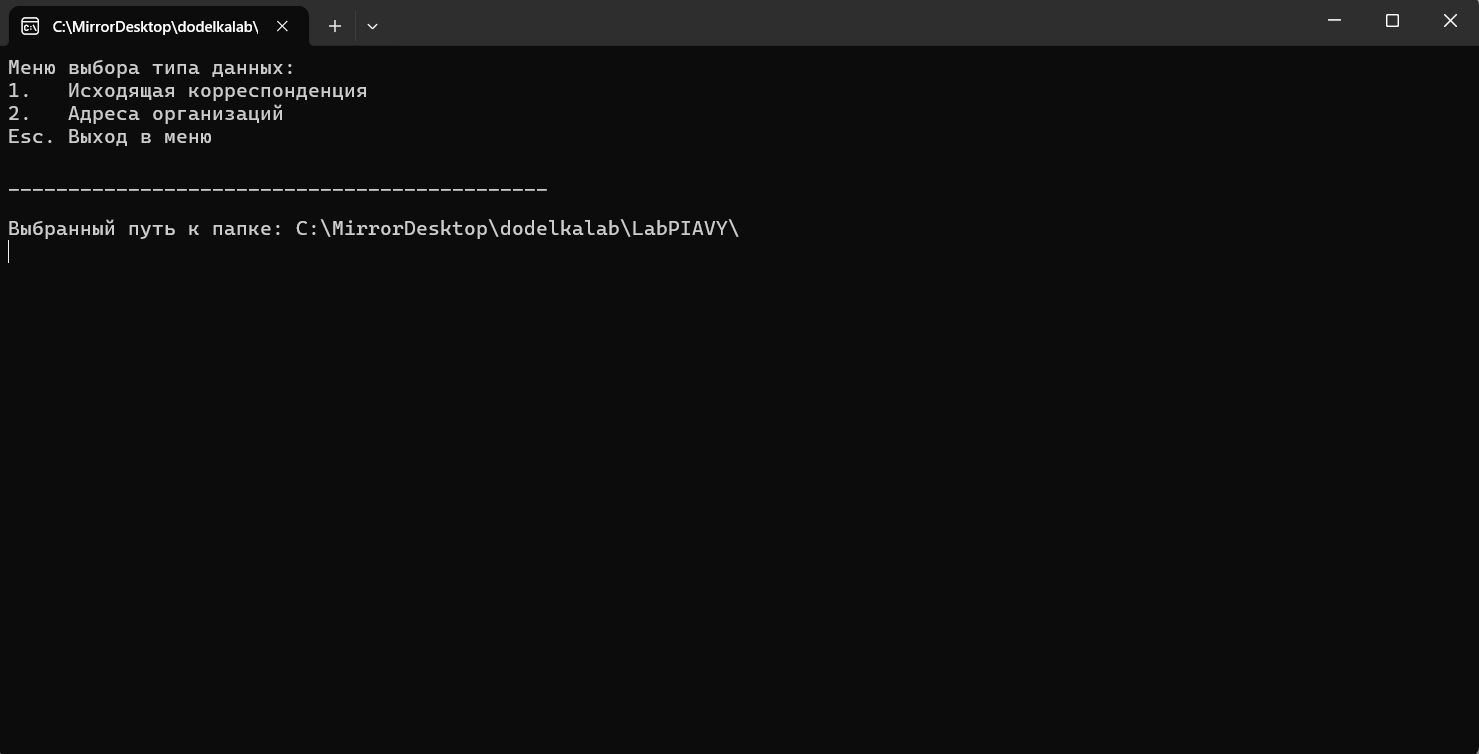


Рисунок 6 Меню выбора типа данных.

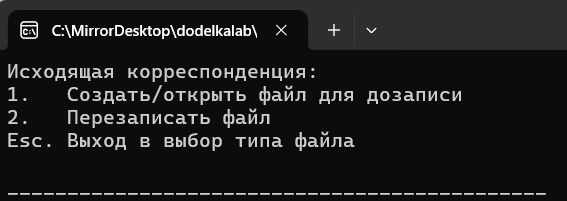


Рисунок 7 Работа с данными исходящей корреспонденции

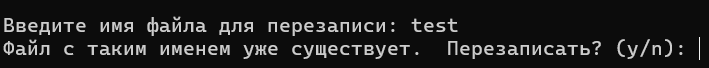


Рисунок 8 Перезапись файла с подтверждением.

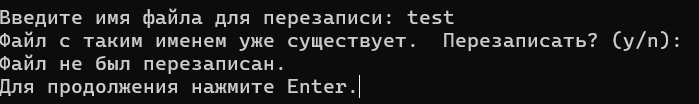


Рисунок 9 Отказ от перезаписи файла.

Создание нового файла создает новый файл в указанной папке (рис. 8). Если пользователь откроет несуществующий файл, то программа автоматически создаст данный файл. Режим перезаписи очищает файл перед записью в него. Режим дозаписи в файл добавляет записи в конец редактируемого файла, не затрагивая изначальное содержимое. Интерфейсы режимов одинаковы, поэтому будет рассмотрен режим дозаписи в файл.

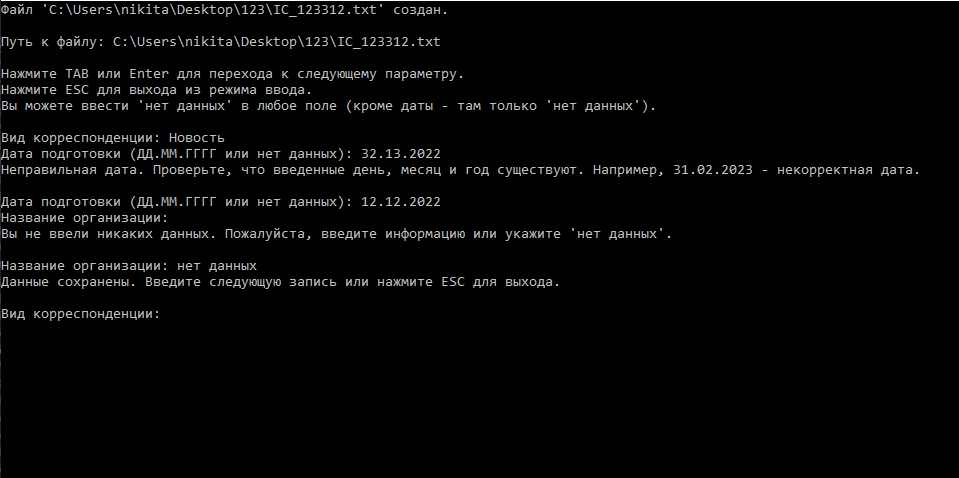


Рисунок 10 Ввод данных.

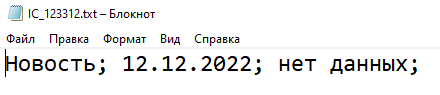
****

Рисунок 11 Результат работы программы.

При неправильном вводе даты или ФИО программа запросит повторный ввод (рис. 10). Если у пользователя отсутствует информация по какому-либо полю, он может написать “нет данных”. После перехода на новую запись (новые 3 поля ввода), программа автоматически сохранит предыдущу введенную информацию в файл (рис. 11).