МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий

Кафедра/департамент «Информационные системы»

КУРСОВАЯ РАБОТА / КУРСОВОЙ ПРОЕКТ

по дисциплине

Алгоритмизация и программирование

на тему «Программа учета выданных книг в библиотеке»

Выполнил: обучающийся
группы: ИС/б-21-3-о
Пышногуб В.С.
« __» _____20 22 г.
Научный руководитель:
Сметанина Т.И.
«__» _____20 22 г.

Оценка_____
« __» _____ 20 22 г.

Севастополь 2022

АННОТАЦИЯ

Пояснительная записка содержит описание программы для работы с табличными данными, а именно, записями о выданных книгах в библиотеке, разработанной в рамках курсового проектирования, целью которого закрепление и углубление знаний в области основ структурного и модульногопрограммирования. Также целью является приобретение практических навыков разработки приложений с использованием сторонних открытых библиотек и примитивной терминальной графики.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках настоящего курсового проектирования ведется разработка программы по теме «Программа учета выданных книг в библиотеке» на основании документа – техническое задание – и в рамках организации – Севастопольский государственный университет. Дата выдачи задания: 08.09.2022.

С появлением и широким распространением табличных процессоров, одним из самых известных представителей которых является Microsoft Exel, и иных программ, позволяющих обрабатывать большие объёмы табличных данных без необходимости знания пользователем языков программирования, написание узкоспециализированных программ, примером которых является разрабатываемая программа, утратило свою актуальность, что, тем не менее, никак не сказалось на возможности использования разработки программы, преимуществом которой является простота и крайне малое потребление ресурсов компьютера.

Целью курсового проектирования является систематизация, закрепление и углубление знаний в области основ процедурного программирования и совершенствование практических навыков разработки программ на языке Си на примере разработки программы «Программа учета выданных книг в библиотеке», представляющей собой упрощённое подобие базы данных и позволяющей выполнять различные операции над записями.

Для достижения цели на разных этапах курсового проектирования должны быть решены следующие задачи:

– выбор варианта задания и детализация поставки задачи;

– определение требований к функциям, выполняемых разрабатываемой программой;

– выбор типов и проектирование структур данных, определяющих способы представления, хранения и преобразования входных, выходных и промежуточных данных;

– разработка модульной структуры программы, определение функций модулей и способов их взаимодействия;

– написание текста программных модулей на алгоритмическом языке;

– разработка тестовых примеров;

– тестирование и отладка программы;

Также должны быть разработаны программные документы в соответствии с действующими стандартами.

1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ

Программа предназначена для организации, хранения, поиска и модификации данных о выданных в библиотеке книгах через пользователь-ориентированный терминальный интерфейс.

Области применения программы: библиотеки, находящиеся в городе Севастополь и в его округе. Одной из возможностей программы является импорт данных, сохранённых в формате «txt», что позволяет работать с данными как из других программ, так и с данными, изначально созданными с помощью текстовых редакторов.

2 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ

2.1 Постановка задачи на разработку программы

Был выдан 14 вариант задания.

Входной файл имеет следующую структуру:

– ид;

– ФИО абонента;

– автор книги;

– название книги;

– издательство;

– дата выдачи;

– стоимость книги;

Программа должна предоставлять меню-ориентированный интерфейс, позволяющий выполнять следующий минимально необходимый набор действий:

– создание и добавление элементов в таблицу;

– просмотр существующих записей;

– удаление записи по ключевому полю;

– редактирование записи;

– сортировка записей в таблице по выбранному полю;

– сохранение данных в бинарный/текстовый файл с заданным именем файла (по выбору пользователя);

– загрузка данных из бинарного/текстового файла с заданным именем файла (по выбору пользователя);

– реализовать выполнение требуемого по варианту задания;

– корректное завершение работы программы.

В случае обработки записей по варианту программа должна выбрать ФИО и количество книг задолжников, которые не сдали книги обратно в течении 90 дней с момента выдачи.

2.2 Описание и обоснование выбора метода организации входных, выходных и промежуточных данных

Для хранения данных в оперативной памяти было принято решение использовать бинарное дерево, потому что бинарное дерево позволяет ускорить поиск требуемых элементов и постраничный вывод данных на экран.

Были определены дополнительные структуры данных.

Структура хранения ФИО:

```
typedef struct { //Структура для хранения фио
    char name[80]; //Имя
    char surname[80];//Фамилия
    char secondname[80];//отчество
}fio_t;
```

Структура хранения пользовательских данных:

```
typedef struct { // основная структура информационного поля
    unsigned int id; //идентификатор поля
    fio_t fio; //поле структуры фио
    struct { // структура данных для автора книги
        char surname[60];
        char inicial[20];
    }autor;
    char book_name[160]; //поле для названия книги
    char izd[70]; // поле для издания
    struct { // структура, хранит дату выдачи книги
        int d;
        int m;
        int y;
    }date_out;
    float cost; // цена книги
} abonent_t;
```

Элемент дерева:

```
typedef struct abonent_l {
    abonent_t info; // информационное поле
    struct abonent_l* right; //правая нода
    struct abonent_l* left; // левая нода
} abonent;
```

Структура для хранения должников:

```
typedef struct { // структура для хранения должников
    unsigned int id;  //идентификатор
    fio_t fio; // поле фио
    int count_dolg_books; // количество книг в задолжности
```

```
}dolgi_pers_t;
```

Структура для хранения массива должников и его размера:

```
typedef struct { // структура-обертка для хранения данных должников
    dolgi_pers_t* info_mass; // массив должников
    int count; // количество должников
}dolgi_pers_t_obr;
```

Структура для хранения меню.

```
typedef struct { // структура одного элемента меню
    int _menu_size; //размер меню
    char _name[80]; //наименование пункта меню
    char** _sub_menu; //указатель на массив сабменю
    int* _sub_menu_lenght; //количество элементов сабменю
    int _max_sub_lenght; //длина самой большого элемента в сабменю
    int _menu_name_lenght; //длина имени меню
    } _menu_item;
```

Структура для хранения информации о строке таблицы.

```
typedef struct { // структура хранящая данные столбца таблицы
    char* name; //имя столбца
    int size; //его размер
    int resizebl; //флаг, можно ли изменять ее размер ?
}_table_col;
```

Структура для хранения общей информации о таблице.

```
typedef struct { // структура для хранения метаданных таблицы
    _table_col* _cols; //массив столбцов
    int _col_count; //количество столбцов
}_tabel_metadata;
```

2.3 Обоснование выбора языка и среды программирования

Исходный код программы был написан на языке программирования C, стандарт C99. Причина выбора:

– скорость выполнения функций;

– эффективное потребление памяти;

Для разработки программы была создана следующая среда разработки:

– компилятор: Microsoft C ++. Основные причины выбора: совместимость с многими версия ос Windows, быстрая скорость работы, отслеживание множества ошибок;

– текстовый редактор: Microsoft Visual Studio. Основные причины выбора: наличие огромного функционала по работе с кодом и реваторингу кода; возможность установки дополнений из предоставленной библиотеки; наличие встроенного гитхаба;

– система сборки проекта: Windwos 11. Причина использования: самая популярная операционная среда среди пользователей.

– ситема контроля версий кода: git.

В качестве основной операционной системы была выбрана Windows .

2.4 Разработка модульной структуры программы

В основу организации программы был положен принцип событийного управления. До начала разработки было принято решение разделить программу на 5 частей:

– главный файл, содержащий инициализатор меню, инициализатор таблицы и реализующий вызовы других модулей;

– ядро интерфейса, содержащее основные функции для работы с пользовательским интерфейсом;

– ядро программы, содержащий функции работы с данными.

– вспомогательные утилиты ввода, содержит в себе утилиты форматированного ввода данных

– вспомогательные утилиты для работы с идентификатором, содержащая функции создания хеш-кода для ид

2.4.1 Дополнительные константы, массивы и перечисления

Перечисление кодов для обработки вызовов меню.

```
enum MenuItemCodes //Коды для обработки вызовов меню
{
    ADD_NEW_RECORD = 1,
    LOAD_FROM_FILE_TYPE = 2,
    LOAD_FROM_FILE = 3,
    SAVE_TO_FILE_TYPE = 4,
    SAVE_TO_FILE = 5,
    TREE_SIZE = 6,
    PRINT_TREE_STRUCT = 7,
    CLEAN_TREE = 8,
    DOLGNIKI_WINDWO = 9,
    PRINT_HORRIBLE_ANIMATION = 10,
    PROGRAM_EXIT = 11
};
```

Перечисление кодов кнопок для обработки вызовов нажатий.

```
enum KeyboardCodes
{
    KEY_ARROW_UP = 72,
    KEY_ARROW_DOWN = 80,
    KEY_ARROW_LEFT = 75,
    KEY_ARROW_RIGHT = 77,
    KEY_TAB = 9,
    KEY_HOME = 71,
    KEY_END = 79,
    KEY_ENTER = 13,
    KEY_ESC = 27,
    KEY_DEL = 83,
    KEY_BACKSPACE = 8,
    KEY_PGUP = 73,
    KEY_PGDOWN = 81
};
```

Перечисление кодов для обработки сортировки.

```
enum sort_fild {
    DEF,
    FIO,
    AUTHOR,
    BOOK_NAME,
    IZD,
    DATE_OUT,
    COST,
    ZADANIE
};
```

Перечисление кодов для обработки ввода.

```
enum WorkingMode
{
    NORMAL = 0,
    PERSONAL = 1,
    INICIAL = 2,
    DATA = 3,
};
```

2.5 Описание алгоритмов функционирования программы

Рекурсивная функция добавления объекта в дерево. В качестве параметров требует указатель на указатель корня и указатель на структуру данных для записи.

```
void tree_add(abonent** root, const abonent_t* info);
```

Рекурсивная функция получения количества объектов в дерево. Возвращает количество элементов, либо 0 если их нет. В качестве параметров требует указатель на корень дерева, и текущее количество просчетов

```
int tree_getNodeCount(const abonent* root, const int accum);
```

Рекурсивная функция получения объекта в дереве по его ид. Возвращает указатель на объект дерева, либо NULL если объект не найден. В качестве параметров требует указатель на корень дерева, и ключь для поиска.

```
abonent* tree_getLeafById(abonent* root, const int id);
```

Рекурсивная функция удаления объекта в дереве по его ид. В качестве параметров требует указатель на корень дерева, и ключь для поиска.

```
void tree_deleteNodeById(abonent** root, const int id);
```

Рекурсивная функция отрисовки структуры дерева. В качестве параметров требует указатель на корень дерева, и параметр оступа.

```
void tree_deleteNodeById(abonent** root, const int id);
```

Рекурсивная функция удаления дерева. В качестве параметров требует указатель на корень дерева. Вернет пустой указатель.

```
abonent* tree_delete(abonent* root);
```

Рекурсивная функция записи дерева в бинарный файл. В качестве параметров требует указатель на файл и корень дерева.

```
void printToFile(FILE* f, abonent* root);
```

Рекурсивная функция записи дерева в тектовый файл. В качестве параметров требует указатель на файл и корень дерева.

```
void printToFile_Text(FILE* f, abonent* root);
```

Рекурсивная функция получения данных из дерева для дальнейшей работы с ними. В качестве параметров требует указатель на корень дерева, указатель на область хранения данных для вывода, указатель на индекс. Вернет указатель на массив данных.

```
abonent_t* _get_output_info(abonent* root, abonent_t* _output_memory, int* index);
```

Рекурсивная функция получения данных должников из дерева для дальнейшей работы с ними. В качестве параметров требует указатель на корень дерева, указатель на область хранения данных для вывода, день, месяц и год для сравнения. Вернет указатель на струткту данных.

```
dolgi_pers_t_obr* _get_dolgi_info(abonent* root, dolgi_pers_t_obr* _output_memory, int d, int m, int y)
```

Функция циклического добавления элементов в дерево. В качестве параметров передается указатель на указатель на корень дерева и указатель на данные таблицы. Возвращает EXIST_SUCCESS.

```
int addNewElement(abonent** st, _tabel_metadata *table);
```

Функция-обертка для вызова печати дерева в файл. В качестве параметров передается указатель на файл, указатель на корень дерева. Возвращает 0 в случае успеха, 666 если возникла ошибка.

```
int create_file_type(FILE* f, abonent* St);
```

Функция считывания данных из типизированного файла. В качестве параметров передается указатель на файл. Возвращает указатель на корень дерева.

```
abonent* loadFromFile_new_type(FILE* f);
```

Функция считывания данных из текстового файла. В качестве параметров передается указатель на файл. Возвращает указатель на корень дерева.

```
abonent* loadFromFile_new_text(FILE* f);
```

Функция вызова функций программы согласно выбраного индекса. В качестве параметров передается номер выбранного пункта, указатель на файл, указатель на структуру с данными таблицы.

```
void MenuSelect(int selector, FILE* f, _tabel_metadata* table);
```

Функция инициализации меню программы. В качестве параметров передается указатель на пустую структуру меню.

```
_menu_item* _init_menu(_menu_item* menu);
```

Функция инициализации таблицы программы. В качестве параметров передается указатель на пустую структуру таблицы.

```
_tabel_metadata* _init_table(_tabel_metadata* table);
```

Функция получения хеш кода из ФИО. В качестве параметра, передается указатель на информационное поле. Возвращает беззнаковое значение.

```
unsigned int util_hashCodeFromAbonentStruct(const abonent_t* a);
```

Функция получения хеш кода из ФИО. В качестве параметра, передается указатель на поле структуры fio. Возвращает беззнаковое значение.

```
unsigned int util_hashCodeFromFio(const fio_t* fio);
```

Функция получения хеш кода из строки. В качестве параметра, передается указатель на поле структуры fio. Возвращает беззнаковое значение.

```
unsigned int util_hashCodeFromString(const char* string);
```

Функция получения информации о консоли. В качестве параметра, передается указатель на пустой указатель на буфер консоли. Возвращает 1 или 0 в зависимости от успешности.

```
BOOL _get_con_info_local(CONSOLE_SCREEN_BUFFER_INFO* x);
```

Функция установки курсора в консоли на определенные координаты. В качестве параметра, структура COORD. Возвращает 1 или 0 в зависимости от успешности.

```
BOOL _set_cur_to_pos_local(COORD cor);
```

Функция перевода строки из utf-8 в 1251. В качестве параметра передается один элемент типа char. Возвращает элемент типа char.

```
char convert_u8_to_1251(int c);
```

Функция форматированного ввода данных с плавающей точкой. В качестве параметра передается указатель на поле, в которое необходимо записать значение. Возвращает номер нажатого спецсимвола.

```
int input_float(float * info);
```

Функция форматированного ввода данных. В качестве параметра передается указатель на буфер, в которое необходимо записать значение, размер буфера, а так же режим работы: 0 - разрешение на ввод всего, кроме спец символов, 1- для реализации ввода фамилии\имени\отчества\издательства (не допускаются цифры и т.д), 2 - ввод инициалов. Возвращает номер нажатого спецсимвола.

```
int input_string(char* input_buff, int buff_size, int mode);
```

Функция форматированного ввода даты. В качестве параметров передается указатели на значения, в которое необходимо записать данные. Возвращает номер нажатого спецсимвола.

```
int in_date(int* d, int*m,int *y);
```

Функция очистки внутреннего поля окна.

```
void clear();
```

Функция очистки поля табличного окна для отрисовки таблицы.

```
void clear_table();
```

Функция очистки поля окна программы без очистки меню.

```
void clear_for_info();
```

Функция получения новой позиции курсора. В качестве параметров получает символ, который ввел пользователь, указатель на двумерный массив текущей

позиции, максимальное число строк, максимальное число столбцов и флаг разрешения на изменение x. Вернет указатель на двумерный массив.

```
int* _get_curent_selection(char  c // Символ клавиатуры
    ,int * position // Массив в котором хранятся x и y
    ,int MaxY // Максимальный количество строк
    , int Colums  // Количество столбцов, по умолчанию - 1
    ,int _flag_x_readonly //можно ли менять x?
)
```

Функция отрисовки меню программы и работы с окнами. В качестве параметров получает указатель на массив объектов меню, указатель на двумерный массив текущей позиции, количество элементов в меню, количество кнопок в меню, указатель на массив информации для вывода, количесвто данных для вывода, указатель на структуру, хранящую данные о полях таблицы, двойной указатель на корень дерева и структуру, содержащую текущие параметры сортировки. Вернет индекс выбраного элемента.

```
int _print_menu(_menu_item* _menu //Массив объектов  меню
    , int* position //Массив текущей позиции x и y
    , int _menu_size  //Колличество элементов в массиве объектов меню
    , int _menu_buttons,//Количество кнопок меню
    abonent_t * _output_mas, //массив для вывода информации
    int _output_colcount, // количество данных для вывода
    _tabel_metadata * table, //структура хранящая информацию о полях таблицы
    abonent** root, //указатель на указатель на корень дерева
    sort_struct* sort // структура, хранящая текущую сортировку
)
```

Функция печати заднего фона и рабочей области окна. В качестве параметров требует ширину и высоту окна.

```
void _print_bakground(int _window_w,int _window_h);
```

Функция печати рабочей области программы. В качестве параметров принимает ширину и высоту окна.

```
void _print_border(int _window_w, int _window_h);
```

Функция получения текущего размера окна. В качестве параметра передается указатель на двумерный массив. Возвращает указатель на двумерный массив.

```
int* _get_window_size(int* size);
```

Функция отрисовки маленького окна. В качестве параметров требует текущюю ширину и высоту окна, а так же указатель на строку названия файла.

```
void _window(int _window_w, int _window_h, char* title);
```

Функция отрисовки окна выбора. В качестве параметра требует указатель на отображаемое сообщение. Возвращает 1 в случае подтверждения , иначе 0.

```
int _confirm_window(char * message);
```

Функция отрисовки окна ввода значения.

```
void _in_window();
```

Функция отрисовки окна с сообщением. В качестве параметра требует указатель на сообщение.

```
void _message_window(char* message);
```

Функция отрисовки окна таблицы с выводом данных и возможностью управления и работы с записями. В качестве параметров принимает указатель на данные о таблице, указатель на массив данных для вывода, указатель на количество данных для отрисовки, указатель на номер текущей страницы, указатель на флаг активности фокуса на таблице, двойной указатель на дерево и указатель на структуру, хранящую сортировку. В качестве результата возвращает EXIT_SUCCESS.

```
int _table_window(_tabel_metadata * table, abonent_t * _output_mass, int * _info_count, int*  page, int * _table_focus_flag, abonent** root, sort_struct* sort);
```

Функция отрисовки большого окна. В качестве параметра получает указатель на массив, хранящий наименование окна.

```
void _big_window(char* title);
```

Функция отрисовки окна ввода/редактирования данных о записи. В качестве параметров получет указатель на данные таблицы, указатель на структуру для хранения данных или NULL если нужно создать новую запись, флаг цикличности.

```
abonent_t* _in_info_window(_tabel_metadata* table, abonent_t *_output_info,int _cycle_in_flag);
```

Функция печати помощи в главном окне программы. В качестве параметра получает указатель на сообщение, которое нужно вывести в помощь. Вернет 0 в случае успеха.

```
int print_help(char * help_message);
```

Функция сортировки данных по различным полям. В качестве параметров получает указатель на массив данных, указатель на количество данных, указатель на структуру с типами сортировки. Возвращает указатель на массив данных для вывода.

```
abonent_t* _sort_output(abonent_t* _output_mass, int* filds_count, sort_struct* sorts);
```

Функция отрисовки окна с выводом таблицы с должниками и вохможностью просмотра нескольких страниц. В качестве параметра передается двойной указатель на корень дерева.

```
void dolgiWindow(abonent ** root);
```

Рисунок А.1 – структурная схема функции…...

Поблочное описание:

– блок 1 –;

– блок 2 –;

2.6 Обоснование состава технических и программных средств

Для работы программы требуется использование компьютера с дисплеем для вывода данных и клавиатурой для приёма команд от пользователя. Должна быть предустановлена операционная система семейства Windows версии не ниже Windows 7. Требования относительно жёсткого диска: 160 килобайта для хранения программы, дополнительное пространство для хранения файлов базы данных. Для запуска программы требуется не менее 1024 килобайт оперативной памяти для 64 битных систем, а также дополнительная память для обработки базы данных.

3 ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1 Условия выполнения программы

Таблица 3.1 демонстрирует минимально необходимые системные требования для запуска приложения.

Таблица 3.1 – Системные требования

Операционная система	Windows 7 и выше
Процессор	Одноядерный процессор, x86_64, не менее 1ГГц
ОЗУ	Не менее 1024 КБ
Место на внешнем носителе	Не менее 160 КБ
Требования к терминалу	поддержка 256bit цветов; размеры окна – не менее 107x24 символов; поддержка кодировки UTF-8

3.2 Загрузка и запуск программы

Запустить программу можно двумя способами: …..

3.3 Проверка работоспособности программы

Для проверки работоспособности программы был создан файл в формате «txt». Содержимое файла представлено ниже: ……

Проверка работоспособности программы была проведена успешно: программа корректно считала данные из одного файла, изменила их и записала изменения в указанный пользователем другой файл.

ЗАКЛЮЧЕНИЕ

В соответствии с вариантом задания разработана программа, в основу алгоритма которой положена структура данных в виде бинарного дерева, позволяющего выполнять просмотр данных. Особенностями программы являются: сортировка элементов конкретной таблицы по всем полям как по возрастанию (в алфавитном порядке), так и по убыванию (не в алфавитном порядке); удобный, ориентированный на пользователя интерфейс программы, схожий с стандартным интерфейсом некоторых текстовых редакторов.

Разработка велась на базе Windows 11 в редакторе Visual Studio с использованием Microsoft C++ для компиляции и git для контроля версий кода.

В процессе разработки возникало несколько проблем.

Во-первых, для хранения строк было принято решение использовать массив символов char. С одной стороны, это дало возможность хранить строки в кодировке 1251, а также корректное отображение нелатинские символов. С другой стороны, был усложнён процесс разработки ввиду того, что для отрисовки меню приложения используется кодировка UTF-8. Соответственно пришлось использовать отдельные функции конвертации данных из UTF-8 в 1251, а так же использовать стороний модуль utf_8, который позволял работать с кодировкой в UTF-8 в обычном массиве типа char.

Во-вторых, ввиду громосткости и наполненности графическими элементами отлаживать программу было тяжелее.

Таким образом были достигнуты цели курсового проектирования: углублены знания языка Си; получен навык разработки программ с использованием методологии структурного программирования, а также получены практические навыки разработки приложений с использованием стронних открытых библиотек. Полученные навыки помогут разрабатывать пользователь-ориентированные приложения в терминальной среде.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Г. Шилдт C++ для начинающих. Серия "Шаг за шагом" / Г. Шилдт; пер. с англ. - М.: ЭКОМ Паблишерз, 2013. - 643 с.

2. Методические указания к лабораторным работам по дисциплине «Информатика» для студентов дневной и заочной форм обучения направления 09.03.02 – «Информационные системы и технологии», часть 1 / Сост. В.Н.Бондарев, Т.И. Сметанина. – Севастополь: Изд-во СевГУ, 2014. – 44 с.

3. Методические указания к лабораторным работам по дисциплине «Информатика» для студентов дневной и заочной форм обучения направления 09.03.02 – «Информационные системы и технологии», часть 2 / Сост. В.Н.Бондарев, Т.И. Сметанина – Севастополь: Изд-во СевГУ, 2014. – 64с.

4. Структурное программирование на языке С/С++: методические указания клабораторным работам по дисциплине «Основы программирования и алгоритмические языки» для студентов дневной и заочной форм обучения направления 09.03.02 – «Информационные системы и технологии», часть 1 / Сост. В.Н. Бондарев, Т.И. Сметанина.– Севастополь: Изд-во СевГУ, 2015. – 60 с.

5. Павловская Т. А. Паскаль. Программирование на языке высокого уровня: практикум / Т. А. Павловская. – СПб.: Питер, 2006. – 317 с.

6. Павловская Т. А. Паскаль. Программирование на языке высокого уровня: учеб. для вузов / Т. А. Павловская. – СПб.: Питер, 2008. – 393 с.

7. Керниган Б., Ритчи Д. Язык программирования СИ: пер. с англ./Под ред. и спредисл. В.С. Штаркмана. –2-е изд., перераб. и доп. – М.; СПб. ; К. : Вильямс, 2006. –272с.

8. Вирт Н. Алгоритмы и структуры данных / Н. Вирт; пер. с англ. – М.: Мир,1989. – 360 с.

9. Белецкий Я. Энциклопедия языка Си / Я. Белецкий; пер. с польск. – М.: Мир, 1992. – 687 с.

10. Разработка САПР: в 10 кн. Кн. 3. Проектирование программногообеспечения САПР: практ. пособие / Б. С. Федоров, Н. Б. Гуляев; под ред. А.В. Петрова. – М.: Высш. шк., 1990. – 159с.

ПРИЛОЖЕНИЕ А

Структурные схемы подпрограмм

Рисунок А.1 – Подпрограмма ….

ПРИЛОЖЕНИЕ Б

Листинг программы

Листинг Б.1 – Исходный код программы

Исходный код header-файла «data_utils.h»

```
#include <stdlib.h>
#include "../ThreeStruct.h"

#ifndef _DATAUTILS_H
#define _DATAUTILS_H

extern unsigned int util_hashCodeFromAbonentStruct(const abonent_t*);
extern unsigned int util_hashCodeFromFio(const fio_t*);
extern unsigned int util_hashCodeFromString(const char*);

#endif //! _DATAUTILS_H
```

Исходный код header-файла «input_utils.h»

```
#ifndef INPUT_UTILS
#define INPUT_UTILS


/// <summary>
/// Ввод любых строковых данных
/// </summary>
/// <param name="input_buff">Буфер для ввода</param>
/// <param name="buff_size">Размер буфера</param>
/// <param name="mode">Режим работы, от которого зависят допустимые символы.  0 -
разрешение на ввод всего, кроме спец символов, 1- для реализации ввода
Фамилии\имени\Отчества\Издательства(не допускаются цифры и т.д),2 - Ввод
инициалов</param>
/// <returns></returns>
int input_string(char* input_buff, int buff_size, int mode);
int in_date(int* d, int* m, int* y);
int input_float(float* info);

typedef struct {
    int d;
    int m;
    int y;
}date;

enum WorkingMode
{
    NORMAL = 0,
    PERSONAL = 1,
    INICIAL = 2,
    DATA = 3,

};

#endif // !input_utils

#ifndef KEYCODE
#define KEYCODE
```

```
enum KeyboardCodes
{
    KEY_ARROW_UP = 72,
    KEY_ARROW_DOWN = 80,
    KEY_ARROW_LEFT = 75,
    KEY_ARROW_RIGHT = 77,
    KEY_TAB = 9,
    KEY_HOME = 71,
    KEY_END = 79,
    KEY_ENTER = 13,
    KEY_ESC = 27,
    KEY_DEL = 83,
    KEY_BACKSPACE = 8
};
#endif // !1
```

Исходный код header-файла «MenuStruct.h»

```
#ifndef MenuStruct
#define MenuStruct

typedef struct { // структура одного элемента меню
    int _menu_size; //размер меню
    char _name[80]; //наименование пункта меню
    char** _sub_menu; //указатель на массив сабменю
    int* _sub_menu_lenght; //количество элементов сабменю
    int _max_sub_lenght; //длина самой большого элемента в сабменю
    int _menu_name_lenght; //длина имени меню
    } _menu_item;

typedef struct { // структура хранящая данные столбца таблицы
    char* name; //имя столбца
    int size; //его размер
    int resizebl; //флаг, можно ли изменять ее размер ?
}_table_col;

typedef struct { // структура для хранения метаданных таблицы
    _table_col* _cols; //массив столбцов
    int _col_count; //количество столбцов
}_tabel_metadata;

enum MenuItemCodes //Коды для обработки вызовов меню
{
    ADD_NEW_RECORD = 1,
    LOAD_FROM_FILE_TYPE = 2,
    LOAD_FROM_FILE = 3,
    SAVE_TO_FILE_TYPE = 4,
    SAVE_TO_FILE = 5,
    TREE_SIZE = 6,
    PRINT_TREE_STRUCT = 7,
    CLEAN_TREE = 8,
    DOLGNIKI_WINDWO = 9,
    PRINT_HORRIBLE_ANIMATION = 10,
    PROGRAM_EXIT = 11
};

#endif //! MenuStruct
```

Исходный код header-файла «ThreeStruct.h»

```
#pragma once
typedef struct { //Структура для хранения фио
    char name[80]; //Имя
    char surname[80];//Фамилия
```

```
    char secondname[80];//отчество
}fio_t;

typedef struct { // основная структура информационного поля
    unsigned int id; //идентификатор поля
    fio_t fio; //поле структуры фио
    struct { // структура данных для автора книги
        char surname[60];
        char inicial[20];
    }autor;
    char book_name[160]; //поле для названия книги
    char izd[70]; // поле для издания
    struct { // структура, хранит дату выдачи книги
        int d;
        int m;
        int y;
    }date_out;
    float cost; // цена книги
} abonent_t;

typedef struct { // структура для хранения должников
    unsigned int id;  //идентификатор
    fio_t fio; // поле фио
    int count_dolg_books; // количество книг в задолжности
}dolgi_pers_t;

typedef struct { // структура-обертка для хранения данных должников
    dolgi_pers_t* info_mass; // массив должников
    int count; // количество должников
}dolgi_pers_t_obr;


/// <summary>
/// Структура ноды дерева
/// </summary>
typedef struct abonent_l {
    abonent_t info; // информационное поле
    struct abonent_l* right; //правая нода
    struct abonent_l* left; // левая нода
} abonent;
const int size_abonent_t; //константа, хранит размер информационного поля
```

Исходный код header-файла «tree_operation.h»

```
#include "ThreeStruct.h"
#include <stdio.h>

#ifndef TREE_OPERATION
#define TREE_OPERATION

/// <summary>
/// Вспомогательная рекурсивная функция поиска необходимого листка/ноды
/// </summary>
/// <param name="root">Корень дерева или текущаяя нода</param>
/// <param name="indexToSerch">Ид который необходимо найти</param>
/// <returns>Возвращает указатель на элемент с соответствующим ид, иначе
NULL</returns>
abonent* tree_getLeafById(abonent* root, const int id);

/// <summary>
/// Удалить дерево или часть дерева. Также удалить все поддеревья, относящиеся к
данному узлу.
/// </summary>
/// <param name="st">Корень дерева/текущаяя нода</param>
```

```
/// <returns>Пустой указатель на структуру</returns>
abonent* tree_delete(abonent* root);

/// <summary>
/// Удалить узел дерева по id.
/// </summary>
/// <param name="root">Узел дерева, с которого начинать поиск.</param>
/// <param name="id">ID искомого узла.</param>
/// <returns>Корень дерева после изменения.</returns>
//abonent** tree_deleteNodeById(abonent** root, const int id);
 void tree_deleteNodeById(abonent** root, const int id);
/// <summary>
/// Подсчитать количество узлов в дереве.
/// </summary>
/// <param name="root">Узел дерева, с которого начинать подсчёт.</param>
/// <param name="count">Стартовое значение счётчика.</param>
/// <returns>Количество элементво в дереве.</returns>
int tree_getNodeCount(const abonent* root, const int accum);

/// <summary>
/// Вставить элемент в дерево.
/// </summary>
/// <param name="head">Указатель на указатель дерева.</param>
/// <param name="info">Вставляемый элемент.</param>
void tree_add(abonent** head, const abonent_t* info);


void printToFile(FILE* f, abonent* St);

void printToFile_Text(FILE* f, abonent* St);

void View(abonent* top, int offset);

abonent_t* _get_output_info(abonent* root, abonent_t* _output_memory, int* index);

dolgi_pers_t_obr* _get_dolgi_info(abonent* root, dolgi_pers_t_obr* _output_memory,
int d, int m, int y);



#endif // !TREE_OPERATION
```

Исходный код header-файла «VicMenuDLL.h»

```
#include "MenuStruct.h"
#include "ThreeStruct.h"

#ifndef VicMenuDLL
#define VicMenuDLL

typedef struct {
    int sort_f;
    int sort_t;
}sort_struct;

int* _get_curent_selection(char  c // Символ клавиатуры
    , int* position // Массив в котором хранятся x и y
    , int MaxY // Максимальный количество строк
    , int Colums  // Количество столбцов, по умолчанию - 1
    ,int _flag_x_readonly
);

/// <summary>
```

```
/// Функция печати основного окна меню. Предназначена для создания внутренней области
для работы с данными и реализует возможность
/// работы с самим меню используя стрелки.
/// </summary>
/// <param name="_menu"> структура данных содержащая в себе меню </param>
/// <param name="position">массив текущей позиции x,y</param>
/// <param name="_menu_size">Количество основных пунктов меню</param>
/// <param name="Colums">Количество кнопок меню</param>
/// <param name="_output_mas">Массив, необходимый для вывода информации</param>
/// <param name="_output_colcount">Количество данных в массиве вывода </param>
/// <param name="table">Структура хранящая информациию о таблице</param>
/// <param name="root">Корень дерева, предназначен для вызова функций из
внутренностей меню</param>
/// <param name="sort">Структура, хранящая настройки сортировки</param>
/// <returns>Индекс выбранного элемента</returns>
int _print_menu(_menu_item* _menu, int* position, int _menu_size, int Colums,
abonent_t* _output_mas,
    int _output_colcount, _tabel_metadata* table, abonent** root, sort_struct* sort);
/// <summary>
/// Анимация(над сделать)
/// </summary>
void animatedNeko();
/// <summary>
/// Рисует задний фон согласно параметрам, делая область для работы
/// </summary>
/// <param name="_window_w">ширина окна</param>
/// <param name="_window_h">высота окна</param>
void _print_bakground(int _window_w, int _window_h);
/// <summary>
/// чистит экран
/// </summary>
void clear();
/// <summary>
/// Печать границ окна с очисткой в нутри
/// </summary>
/// <param name="_window_w">ширина окна</param>
/// <param name="_window_h">высота окна</param>
void _print_border(int _window_w, int _window_h);
/// <summary>
/// Возвращает текущий размер окна
/// </summary>
/// <returns>Двумерный массив, ширина и высота</returns>
int* _get_window_size(int* size);
/// <summary>
/// Функция отричовки окна подверждения с кастомным сообщением
/// </summary>
/// <param name="msg">Указатель на строку сообщения, если NULL то выведет стандартное
окно</param>
/// <returns>0 либо 1 в зависимости от того что выбрал пользователь</returns>
int _confirm_window(char *);
/// <summary>
/// Функция печати окна для работы в дальнейшем
/// </summary>
/// <param name="_window_w">Текущая ширина</param>
/// <param name="_window_h">Текущая высота </param>
/// <param name="title">Название окна </param>
void _window(int _window_w, int _window_h,char * title);
/// <summary>
/// Очистка только внутренней части окна
/// </summary>
void clear_for_info();
/// <summary>
/// Окно ввода данных. Считывать данные можно чем угодно вызвав после окна функцию
считывания
```

```
/// </summary>
void _in_window();
/// <summary>
/// Окно сообщения.
/// </summary>
/// <param name="message">Сообщение</param>
void _message_window( char* message);
/// <summary>
/// Табличная форма вывода данных с поддержкой управления в нутри таблицы
/// </summary>
/// <param name="table">структура данных, содержащая информацию о полях
таблицы</param>
/// <param name="_output_mass">Массив данных для вывода</param>
/// <param name="_info_count">Количество данных в массиве для вывода</param>
/// <param name="page">Указатель, хранящий текущюю страницу</param>
/// <param name="_table_focus_flag">Флаг работы с таблицей</param>
/// <param name="root">Корень дерева</param>
/// <param name="sort">Структура сортировки </param>
/// <returns></returns>
int _table_window(_tabel_metadata* table, abonent_t* _output_mass, int* _info_count,
int* page, int* _table_focus_flag,abonent ** root, sort_struct* sort);
/// <summary>
/// Большое окно для работы с данными
/// </summary>
/// <param name="title">Название окна </param>
void _big_window(char* title);
/// <summary>
/// Окно ввода/редактирования информации
/// </summary>
/// <param name="table">Поля из таблицы, используются для отрисовки пунктов</param>
/// <param name="_output_mass">указатель на элемент данных, null для ввода</param>
/// <param name="flag">Флаг цикличного ввода. Для формирования дерева в цикле</param>
/// <returns>Возвращает элемент</returns>
abonent_t* _in_info_window(_tabel_metadata* table, abonent_t* _output_mass,int);
/// <summary>
/// Печать информационной помощи
/// </summary>
/// <param name="help_message">Сообщение помощи. Поддерживает изменение цвета</param>
/// <returns></returns>
int print_help(char* help_message);
/// <summary>
///  Сортирует данные согласно указанной структуре
/// </summary>
/// <param name="">Указатель на данные (массив)</param>
/// <param name="">Количество данных в массиве</param>
/// <param name="">Структура хранящаяя текущюю сортировку</param>
/// <returns>Указатель на новые отсортированные данные</returns>
abonent_t* _sort_output(abonent_t*, int *,sort_struct *);
/// <summary>
/// Окно просмотра должников
/// </summary>
/// <param name="root">Корень дерева для получения информации</param>
void dolgiWindow(abonent** root);
#endif

#ifndef KEYCODE
#define KEYCODE
/// <summary>
/// Енум хранящий коды кнопок
/// </summary>
enum KeyboardCodes
{
    KEY_ARROW_UP = 72,
    KEY_ARROW_DOWN = 80,
```

```
    KEY_ARROW_LEFT = 75,
    KEY_ARROW_RIGHT = 77,
    KEY_TAB = 9,
    KEY_HOME = 71,
    KEY_END = 79,
    KEY_ENTER = 13,
    KEY_ESC = 27,
    KEY_DEL = 83,
    KEY_BACKSPACE = 8,
    KEY_PGUP = 73,
    KEY_PGDOWN = 81
};
#endif
/// <summary>
/// Енум , хранящий поля сортировок
/// </summary>
enum sort_fild {
    DEF,
    FIO,
    AUTHOR,
    BOOK_NAME,
    IZD,
    DATE_OUT,
    COST,
    ZADANIE
};
enum sort_type {
    UP,
    DOWN
};
```

Исходный код файла «Cursach_main.cpp»

```
#pragma warning(disable : 4996);

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <conio.h>
#include <Windows.h>

#include "utf8.h"

#include "include/data_utils.h"
#include "ThreeStruct.h"
#include "VicMenuDLL.h"
#include "MenuStruct.h"
#include "tree_operation.h"


#define clearf() system("cls");

const int size_abonent_t = sizeof(abonent_t);
const int menu_size = 5;
HANDLE hConsole;

int addNewElement(abonent** st, _tabel_metadata *table);
int create_file_type(FILE* f, abonent* St);
abonent* loadFromFile_new_type(FILE* f);
abonent* loadFromFile_new_text(FILE* f);
int correctInfo(abonent* st);
void MenuSelect(int selector, FILE* f, _tabel_metadata* table);
```

```
int correctInfo(abonent* st);
_menu_item* _init_menu(_menu_item* menu);
_tabel_metadata* _init_table(_tabel_metadata* table);
abonent* abonents = NULL;
int position[] = { 1,1 };
int* size;

int main(void) {
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO structCursorInfo;
    GetConsoleCursorInfo(hConsole, &structCursorInfo);
    structCursorInfo.bVisible = FALSE;
    SetConsoleCursorInfo(hConsole, &structCursorInfo);
    system("chcp 1251");
    SetConsoleCP(1251); // Задаем таблицу символов для консоли.
    SetConsoleOutputCP(65001);
    clearf();
    _menu_item* menu = NULL;
    _tabel_metadata* table = NULL;
    menu = _init_menu(menu);
    table = _init_table(table);
    abonent_t* _output_info;
    // system("color F0");
    FILE* f =  fopen("data.dat", "rb+");//Открытие существующего файла для чтения и
записи в конец
    if (!f) {
        f = fopen("data.dat", "wb+"); //Создание нового файла для обновления
        if (!f) {
            puts("Не могу открыть (создать) файл\n");
            return 1;
        }
    }
    while (1) {//вывод меню и запуск соответствующих функций
        clear();
        int leafCount = tree_getNodeCount(abonents, 0);
        _output_info = (abonent_t*)calloc(leafCount, sizeof(abonent_t));
        int  temp = 0;
        _output_info =  _get_output_info(abonents, _output_info,&temp);
        if (leafCount == 0)
            _output_info = NULL;
        static sort_struct sort;
        sort.sort_f = DEF;
        sort.sort_t = UP;
        MenuSelect(_print_menu(menu, position, menu_size,
5,_output_info,leafCount,table, &abonents, &sort), f,table);
    }
}
int Posid = 1;

void MenuSelect(int selector, FILE* f,_tabel_metadata *table )
{
    char* a = (char*)calloc(200, sizeof(char));
    switch (selector) {

    case ADD_NEW_RECORD:
        addNewElement(&abonents,table);
        break;

    case LOAD_FROM_FILE_TYPE:
        if (_confirm_window(NULL)) {
            f = fopen("data.dat", "rb+");
            abonents = loadFromFile_new_type(f);
        }
        break;
```

```
    case SAVE_TO_FILE_TYPE:
        create_file_type(f, abonents);
        break;
    case CLEAN_TREE:
        size = _get_window_size(size);
        if (_confirm_window(NULL)) {
            abonents = tree_delete(abonents);
            sprintf(a, "Дерево очищено");
            _message_window( a);
            getch();
        }
        break;
    case PRINT_TREE_STRUCT:
        clear();
        printf("---------------------------------------------- Структура дерева -----
-----------------------------------\n");
        View(abonents, 1);
        printf("---------------------------------------------- Конец струк. дерева ----
-----------------------------------\n");
        getch();
        break;
    case TREE_SIZE:
        if (_confirm_window(NULL)) {
            sprintf(a, "Дерево содержит %d записей.",tree_getNodeCount(abonents, 0));
            _message_window(a);
            getch();
        }
        break;
    case PRINT_HORRIBLE_ANIMATION:
        animatedNeko();
        break;
    case LOAD_FROM_FILE:
        if (_confirm_window(NULL)) {
            _in_window();
            char a[90]; scanf("%s", a);
            sprintf(a, "%s.txt", a);
            FILE* text = fopen(a, "rt");
            if (!text) break;
            abonents = loadFromFile_new_text(text);
            fclose(text);
        }
        break;
    case SAVE_TO_FILE:
        if (_confirm_window(NULL)) {
            _in_window();
            char a[90]; scanf("%s", a);
            sprintf(a, "%s.txt", a);
            FILE* text = fopen(a, "wt");
            if (!text) break;
            printToFile_Text(text, abonents);
            fclose(text);

        }
            break;
    case DOLGNIKI_WINDWO:
        if (_confirm_window(NULL)) {
            dolgiWindow(&abonents);


        }
        break;
    case PROGRAM_EXIT:
        if (_confirm_window(NULL)) {
```

```
            if (!abonents) {
                tree_delete(abonents);
            }
            exit(666);
        }
        break;

    }
}

int create_file_type(FILE* f, abonent* root)
{
    if (!root)
        return 666;

    fclose(f);
    f = fopen("data.dat", "wb+");

    if (f == NULL)
    {
        _message_window("Не удалось открть файл для записи");
        Sleep(2000);
        return 666;
    }

    fseek(f, 0, SEEK_SET);

    printToFile(f, root);
    fclose(f);
    _message_window("Данные успешно сохранены...");
    Sleep(2000);

    return 0;
}

int addNewElement(abonent** st,_tabel_metadata* table)
{
    abonent_t* d;
        while (1) {
         d =   _in_info_window(table, NULL, 1);
         if (!d)   return EXIT_SUCCESS;
         d->id = util_hashCodeFromFio(&d->fio);
         tree_add(st, d);
              }
    return EXIT_SUCCESS;
}

abonent* loadFromFile_new_text(FILE* f)
{
    abonent_t tmp;
    abonent* head = NULL;
    int count = 1;
    while (1) {
        fscanf(f, "%s %s %s", tmp.fio.surname, tmp.fio.name, tmp.fio.secondname);
        fgets(tmp.book_name, 160, f);
        fscanf(f, "%s %s", tmp.autor.surname, tmp.autor.inicial);
        fgets(tmp.izd, 70, f);
        fscanf(f, "%d %d %d", &(tmp.date_out.d), &(tmp.date_out.m),
&(tmp.date_out.y));
        fscanf(f, "%f", &(tmp.cost));
        if (feof(f)) break;
        for (int i = 0; i < strlen(tmp.book_name); i++) {
            if (tmp.book_name[i] == '\n') tmp.book_name[i] = '\0';
        }
```

```
        for (int i = 0; i < strlen(tmp.izd); i++) {
            if (tmp.izd[i] == '\n') tmp.izd[i] = '\0';
        }

        tmp.id = util_hashCodeFromFio(&tmp.fio);
        tree_add(&head, &tmp);
        count++;
    }
    _message_window("Данные считаны");
    Sleep(2000);
    return head;
}

abonent* loadFromFile_new_type(FILE* f)
{
    abonent_t tmp;
    abonent* head = NULL;
    int count = 1;
    fseek(f, 0, SEEK_SET);
    while (fread(&tmp, sizeof(abonent_t), 1, f))
    {
        tree_add(&head, &tmp);
        count++;
    }
    _message_window("Данные считаны");
    Sleep(2000);
    return head;
}

int correctInfo(abonent* st)
{
   return EXIT_SUCCESS;
}


_menu_item* _init_menu(_menu_item* menu) {
    menu = (_menu_item *) calloc(1, sizeof(_menu_item) * 5);
//-------------------------------------------------------------------------------------
----------------------
    strcpy(menu[0]._name, "Внести данные");
    menu[0]._menu_name_lenght = 14;
    menu[0]._menu_size = 3;
    menu[0]._sub_menu = (char**)calloc(menu[0]._menu_size, sizeof(char*) );
    for (int i = 0; i < menu[0]._menu_size; i++) {
        menu[0]._sub_menu[i] = (char*)calloc(90, sizeof(char));
                }
    menu[0]._sub_menu_lenght = (int*)calloc(3, sizeof(int));
        strcpy(menu[0]._sub_menu[0], "Добавить новый элемент с клавиатуры");
menu[0]._sub_menu_lenght[0] = 36;
        strcpy(menu[0]._sub_menu[1], "Загрузить из типизированного файла");
menu[0]._sub_menu_lenght[1] = 35;
        strcpy(menu[0]._sub_menu[2], "Загрузить из текстового файла");
menu[0]._sub_menu_lenght[2] = 30;
//-------------------------------------------------------------------------------------
----------------------
        strcpy(menu[1]._name, "Сохранить");
        menu[1]._menu_name_lenght = 10;
        menu[1]._menu_size = 2;
        menu[1]._sub_menu = (char**)calloc(menu[1]._menu_size, sizeof(char*) );
        for (int i = 0; i < menu[1]._menu_size; i++) {
            menu[1]._sub_menu[i] = (char*)calloc(90, sizeof(char));
        }
        menu[1]._sub_menu_lenght = (int*)calloc(4, sizeof(int));
```

```
        strcpy(menu[1]._sub_menu[0], "Сохранить в типизированный файл");
menu[1]._sub_menu_lenght[0] = 32;
        strcpy(menu[1]._sub_menu[1], "Сохранить в текстовый файл");
menu[1]._sub_menu_lenght[1] = 27;
//--------------------------------------------------------------------------------
----------------------
        strcpy(menu[2]._name, "Работа с деревом");
        menu[2]._menu_name_lenght = 17;
        menu[2]._menu_size = 4;
        menu[2]._sub_menu = (char**)calloc(menu[2]._menu_size, sizeof(char*));
        for (int i = 0; i < menu[2]._menu_size; i++) {
            menu[2]._sub_menu[i] = (char*)calloc(90, sizeof(char));

        }
        menu[2]._sub_menu_lenght = (int*)calloc(5, sizeof(int));
        strcpy(menu[2]._sub_menu[0], "Отобразить структуру дерева");
menu[2]._sub_menu_lenght[0] = 28;
        strcpy(menu[2]._sub_menu[1], "Колличество элементов в дереве");
menu[2]._sub_menu_lenght[1] = 31;
        strcpy(menu[2]._sub_menu[2], "Очистить дерево");  menu[2]._sub_menu_lenght[2]
= 16;
        strcpy(menu[2]._sub_menu[3], "Список должников");
menu[2]._sub_menu_lenght[3] = 17;
//--------------------------------------------------------------------------------
----------------------
        strcpy(menu[3]._name, "Прочее");
        menu[3]._menu_name_lenght = 7;
        menu[3]._menu_size = 2;
        //"Анимация",
        //"Картинка",
        menu[3]._sub_menu = (char**)calloc(menu[3]._menu_size, sizeof(char*) );
        for (int i = 0; i < menu[3]._menu_size; i++) {
            menu[3]._sub_menu[i] = (char*)calloc(90, sizeof(char));

        }
        menu[3]._sub_menu_lenght = (int*)calloc(2, sizeof(int));
        strcpy(menu[3]._sub_menu[0], "Анимация"); menu[3]._sub_menu_lenght[0] = 9;
        strcpy(menu[3]._sub_menu[1], "Картинка"); menu[3]._sub_menu_lenght[1] = 9;
//--------------------------------------------------------------------------------
----------------------
        strcpy(menu[4]._name, "Выход");
        menu[4]._menu_name_lenght = 6;
        menu[4]._menu_size = 0;
        //"Выход"
        menu[4]._sub_menu = NULL;
    return menu;
}

_tabel_metadata* _init_table(_tabel_metadata* table) {
    table = (_tabel_metadata*)calloc(sizeof(_tabel_metadata), 1);
    table->_col_count = 7;
    table->_cols = (_table_col*)calloc(sizeof(_table_col) , table->_col_count);
    //--------------------------------------------------------------------------
------------//
    table->_cols[0].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[0].name, "№");
    table->_cols[0].resizebl = 0;
    table->_cols[0].size = u8_strlen(table->_cols[0].name);
    //--------------------------------------------------------------------------
------------//
    table->_cols[1].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[1].name, "ФИО абонента");
    table->_cols[1].resizebl = 1;
    table->_cols[1].size = u8_strlen(table->_cols[1].name);
```

```
    //-------------------------------------------------------------------------------
------------//
    table->_cols[2].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[2].name, "Автор книги");
    table->_cols[2].resizebl = 0;
    table->_cols[2].size = u8_strlen(table->_cols[2].name); table->_cols[2].size =18;
    //-------------------------------------------------------------------------------
------------//
    table->_cols[3].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[3].name, "Название книги");
    table->_cols[3].resizebl = 1;
    table->_cols[3].size = u8_strlen(table->_cols[3].name);
    //-------------------------------------------------------------------------------
------------//
    table->_cols[4].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[4].name, "Издательство");
    table->_cols[4].resizebl = 0;
    table->_cols[4].size = u8_strlen(table->_cols[4].name);
    //-------------------------------------------------------------------------------
------------//
    table->_cols[5].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[5].name, "Дата выдачи");
    table->_cols[5].resizebl = 0;
    table->_cols[5].size = u8_strlen(table->_cols[5].name);
    //-------------------------------------------------------------------------------
------------//
    table->_cols[6].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[6].name, "Стоимость,р.");
    table->_cols[6].resizebl = 0;
    table->_cols[6].size = u8_strlen(table->_cols[6].name);
    //-------------------------------------------------------------------------------
------------//
    return table;
}
```

Исходный код файла «data_utils.c»

```
#include "include/data_utils.h"

unsigned int util_hashCodeFromAbonentStruct(const abonent_t* a)
{
      return util_hashCodeFromFio(&a->fio);
}

unsigned int util_hashCodeFromFio(const fio_t* fio)
{
      return util_hashCodeFromString(fio->name) + util_hashCodeFromString(fio-
>secondname) + util_hashCodeFromString(fio->surname);
}

unsigned int util_hashCodeFromString(const char* string)
{
      unsigned int ret = 0;

      for (int position = 0; string[position] != '\0'; ++position)
      {
            ret += string[position];
      }

      return ret;
}
```

Исходный код файла «input_utils.cpp»

```
#pragma warning(disable : 4996);
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <Windows.h>
#include <time.h>


#include "utf8.h"


#include "input_utils.h"

BOOL _get_con_info_local(CONSOLE_SCREEN_BUFFER_INFO* x)
{
    return GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), x);
}

BOOL _set_cur_to_pos_local(COORD cor) {
    return SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), cor);
}



char convert_u8_to_1251(int c) {
    wchar_t s[2];
    s[0] = (wchar_t)c; s[1] = '\0';
    char utf8[20];
    wchar_t wstr[20];
    char s1251[20];
    WideCharToMultiByte(CP_UTF8, 0, s, -1, utf8, 100, NULL, NULL);
    utf8[WideCharToMultiByte(CP_UTF8, 0, s, -1, utf8, 0, NULL, NULL)] = '\0';
    // Подготовили строку UTF8 дальше идет ее преобразование в 1251
    MultiByteToWideChar(CP_UTF8, 0, utf8, -1, wstr, 100);
    wstr[MultiByteToWideChar(CP_UTF8, 0, utf8, -1, wstr, 0)];
    WideCharToMultiByte(1251, 0, wstr, -1, s1251, 100, NULL, NULL);
    s1251[WideCharToMultiByte(1251, 0, wstr, -1, s1251, 0, NULL, NULL)] = '\0';
    return s1251[0];
}



int input_float(float * info) {
    int flag_point = 0;
    char correct_sym[] = { '0','1','2','3' ,'4','5' ,'6','7' ,'8','9','.' };
    SetConsoleCP(65001);
    SetConsoleOutputCP(1251);
    CONSOLE_SCREEN_BUFFER_INFO con_inf; _get_con_info_local(&con_inf);
    int current_pos = 0;
    int in_sym_count = 0;
    char input_buff[400] = {""};
    float old_info = *info;
    COORD positionCur = { 0,0 };
    COORD positionCur_start = { 0,0 };
    positionCur = con_inf.dwCursorPosition;
    positionCur_start = con_inf.dwCursorPosition;
    if (u8_strlen(input_buff) > 0) {
        int k = u8_strlen(input_buff);
        current_pos = k;
        in_sym_count = k;
        // positionCur.X = positionCur.X - k;
        _set_cur_to_pos_local(positionCur);
        printf("%s", input_buff);
```

```
    positionCur = con_inf.dwCursorPosition;
}
int c;
while (1) {
    c = _getwch();
    switch (c)
    {
    case KEY_ENTER:
        if (in_sym_count > 0) {
            printf(" ");
            *info = atof(input_buff);
            return KEY_ENTER;
        }
        break;
    case KEY_ESC:
        //Вернуть старую инфу
        return KEY_ESC;
        break;
    case KEY_BACKSPACE:
    {
        if (current_pos > 0 && in_sym_count > 0) {
            if (current_pos > 0)
            {
                if (input_buff[current_pos-1] == '.') flag_point = 0;
                input_buff[current_pos-1] = '\0';
                current_pos--;  in_sym_count--;
                _get_con_info_local(&con_inf);
                positionCur.X = con_inf.dwCursorPosition.X;
                positionCur.X--; _set_cur_to_pos_local(positionCur);
                printf(" ");
                _set_cur_to_pos_local(positionCur);
            }

        }
    }
    break;
    default:
        c = convert_u8_to_1251(c);
         {
            int flag = 1;
            for (int i = 0; i < 11; i++)
            {
                if (c == correct_sym[i]) {
                    flag = 0;
                    break;
                }
            }
            if (c == '.' && flag_point) flag = 1;
            if (c == '.') {
                flag_point = 1;
            }
            if (c == ' ' )
            {
                if (in_sym_count > 0) {
                    printf(" ");
                    return KEY_ENTER;
                }
            }
            if (!flag) {

                if (in_sym_count < 400) {
                    input_buff[in_sym_count] = c;
                    printf("%c", c);
                    in_sym_count++;
```

```
                        current_pos++;
                    }
                }
            }
            break;
        }
    }
}



/// <summary>
/// Ввод любых строковых данных
/// </summary>
/// <param name="input_buff">Буфер для ввода</param>
/// <param name="buff_size">Размер буфера</param>
/// <param name="mode">Режим работы, от которого зависят допустимые символы.  0 -
разрешение на ввод всего, кроме спец символов, 1- для реализации ввода
Фамилии\имени\Отчества\Издательства(не допускаются цифры и т.д),2 - Ввод
инициалов</param>
/// <returns></returns>
int input_string(char* input_buff, int buff_size, int mode)
{
    int n_count;
    char* invalid_sym;
    int temp = 0;
    switch (mode)
    {
    case NORMAL:
    {
        n_count = 5;
        invalid_sym = (char*)calloc(n_count, sizeof(char));
        for (int i = 35; i < 40; i++) {
            invalid_sym[temp] = i;
            temp++;
        }
        break;
    }
    case PERSONAL: {
        n_count = 42;
        invalid_sym = (char*)calloc(n_count, sizeof(char));
        for (int i = 32; i < 65; i++) {
            invalid_sym[temp] = i;
            temp++;
        }
        for (int i = 91; i < 97; i++) {
            invalid_sym[temp] = i;
            temp++;
        }
        for (int i = 123; i < 127; i++) {
            invalid_sym[temp] = i;
            temp++;
        }
        break; }
    case INICIAL: {
        n_count = 42;
        invalid_sym = (char*)calloc(n_count, sizeof(char));
        for (int i = 32; i < 65; i++) {
            invalid_sym[temp] = i;
            temp++;
        }
        for (int i = 91; i < 97; i++) {
            invalid_sym[temp] = i;
```

```
        temp++;
    }
    for (int i = 123; i < 127; i++) {
        invalid_sym[temp] = i;
        temp++;
    }
    break; }

    default:
    {
        n_count = 5;
        invalid_sym = (char*)calloc(n_count, sizeof(char));
        for (int i = 35; i < 40; i++) {
            invalid_sym[temp] = i;
            temp++;
        }
        break;

    }
}
SetConsoleCP(65001);
SetConsoleOutputCP(1251);
CONSOLE_SCREEN_BUFFER_INFO con_inf; _get_con_info_local(&con_inf);
int current_pos = 0;
int in_sym_count = 0;
char* old_info = (char*)calloc(buff_size, sizeof(char));
strcpy(old_info, input_buff);
//old_info = *input_buff;
COORD positionCur = { 0,0 };
COORD positionCur_start = { 0,0 };
positionCur = con_inf.dwCursorPosition;
positionCur_start = con_inf.dwCursorPosition;
if (u8_strlen(input_buff) > 0) {
    int k = u8_strlen(input_buff);
    current_pos = k;
    in_sym_count = k;
   // positionCur.X = positionCur.X - k;
    _set_cur_to_pos_local(positionCur);
    printf("%s", input_buff);
    positionCur = con_inf.dwCursorPosition;
}
int c;
while (1) {
    c = _getwch();
    switch (c)
    {
    case KEY_ENTER:
        if (in_sym_count > 0) {
            printf(" ");
            return KEY_ENTER;
        }
        break;
    case KEY_ESC:
        strcpy(input_buff, old_info);
        return KEY_ESC;
        break;
    case KEY_ARROW_UP:
        if (in_sym_count > 0) {
            strcpy(input_buff, old_info);
            _get_con_info_local(&con_inf);
            positionCur.X = con_inf.dwCursorPosition.X;
            positionCur.X--; _set_cur_to_pos_local(positionCur);
            printf(" ");
            _set_cur_to_pos_local(positionCur);
```

```
            return KEY_ARROW_UP;
        }
        break;
    case KEY_ARROW_DOWN:
        if (in_sym_count > 0) {
            strcpy(input_buff, old_info);
            _get_con_info_local(&con_inf);
            positionCur.X = con_inf.dwCursorPosition.X;
            positionCur.X--; _set_cur_to_pos_local(positionCur);
            printf(" ");
            _set_cur_to_pos_local(positionCur);
            return KEY_ARROW_DOWN;
        }
        break;
    case KEY_ARROW_LEFT:
    case KEY_ARROW_RIGHT:
        _get_con_info_local(&con_inf);
        positionCur.X = con_inf.dwCursorPosition.X;
        positionCur.X--; _set_cur_to_pos_local(positionCur);
        printf(" ");
        _set_cur_to_pos_local(positionCur);
        break;
    case KEY_BACKSPACE:
        if (mode != INICIAL) {
            if (current_pos > 0 && in_sym_count > 0)
            {
                input_buff[current_pos] = '\0';
                current_pos--;  in_sym_count--;
                _get_con_info_local(&con_inf);
                positionCur.X = con_inf.dwCursorPosition.X;
                positionCur.X--; _set_cur_to_pos_local(positionCur);
                printf(" ");
                _set_cur_to_pos_local(positionCur);
            }
        }
        else
            {
                if (current_pos > 0 && in_sym_count > 0) {
                    if (current_pos>2)
                    {
                        input_buff[current_pos] = '\0';
                        current_pos--;  in_sym_count--;
                        _get_con_info_local(&con_inf);
                        positionCur.X = con_inf.dwCursorPosition.X;
                        positionCur.X--; _set_cur_to_pos_local(positionCur);
                        printf(" ");
                        _set_cur_to_pos_local(positionCur);
                    }
                    else {
                        input_buff[current_pos] = '\0';
                        current_pos--;
                        input_buff[current_pos] = '\0';
                        current_pos--;
                        in_sym_count--; in_sym_count--;
                        _get_con_info_local(&con_inf);
                        positionCur.X = con_inf.dwCursorPosition.X;
                        positionCur.X--;  positionCur.X--;
_set_cur_to_pos_local(positionCur);
                        printf(" ");  printf(" ");
                        _set_cur_to_pos_local(positionCur);
                    }
                }
            }
        break;
```

```
        default:
            c = convert_u8_to_1251(c);
            if (mode == INICIAL) {
                int flag = 0;
                for (int i = 0; i < n_count; i++)
                {
                    if (c == invalid_sym[i]) {
                        flag = 1;
                        break;
                    }
                }
                if (!flag) {
                    if (in_sym_count < buff_size) {
                        input_buff[in_sym_count] = c;

                        if (in_sym_count == 0) {
                            in_sym_count++;
                            current_pos++;
                            input_buff[in_sym_count] = '.';
                            printf("%c.", c);
                        }
                        else  printf("%c", c);
                        in_sym_count++;
                        current_pos++;
                    }
                }
            }
            else {
                int flag = 0;
                for (int i = 0; i < n_count; i++)
                {
                    if (c == invalid_sym[i]) {
                        flag = 1;
                        break;
                    }
                }
                if (c == ' ' && mode == PERSONAL)
                {
                    if (in_sym_count > 0) {
                        printf(" ");
                        return KEY_ENTER;
                    }
                }
                if (!flag) {
                    if (in_sym_count < buff_size) {
                        input_buff[in_sym_count] = c;
                        printf("%c", c);
                        in_sym_count++;
                        current_pos++;
                    }
                }
            }
            break;
        }
    }
}

int in_date(int* d, int*m,int *y) {
    int n_count;
    char* correct_sym;
    int temp = 0;
    n_count = 10;
    char buff[10] = "";
    correct_sym = (char*)calloc(n_count, sizeof(char));
```

```
for (int i = 48; i < 59; i++) {
    correct_sym[temp] = i;
    temp++;
}
SetConsoleCP(65001);
SetConsoleOutputCP(1251);
CONSOLE_SCREEN_BUFFER_INFO con_inf; _get_con_info_local(&con_inf);
int current_pos = 0;
int in_sym_count = 0;
int cur_selector = 1;
int old_d, old_m, old_y;
if (d) old_d = *d;
if(m)old_m = *m;
 if (y) old_y = *y;
COORD positionCur = { 0,0 };
COORD positionCur_start = { 0,0 };
positionCur = con_inf.dwCursorPosition;
printf("__________");
_set_cur_to_pos_local(positionCur);
positionCur_start = con_inf.dwCursorPosition;
int c;
while (1)
{
    c = _getwch();
    switch (c)
    {
    case KEY_ENTER:
        if (cur_selector > 3) {
            printf(" ");
            return KEY_ENTER;
        }
        break;
    case KEY_ESC:
        return KEY_ESC;
        break;
    case KEY_ARROW_UP:
        if (in_sym_count > 0) {
            _get_con_info_local(&con_inf);
            positionCur.X = con_inf.dwCursorPosition.X;
            positionCur.X--; _set_cur_to_pos_local(positionCur);
            printf(" ");
            _set_cur_to_pos_local(positionCur);
            return KEY_ARROW_UP;
        }
        break;
    case KEY_ARROW_DOWN:
        if (in_sym_count > 0) {
            _get_con_info_local(&con_inf);
            positionCur.X = con_inf.dwCursorPosition.X;
            positionCur.X--; _set_cur_to_pos_local(positionCur);
            printf(" ");
            _set_cur_to_pos_local(positionCur);
            return KEY_ARROW_DOWN;
        }
        break;
    case KEY_ARROW_LEFT:
    case KEY_ARROW_RIGHT:
        _get_con_info_local(&con_inf);
        positionCur.X = con_inf.dwCursorPosition.X;
        positionCur.X--; _set_cur_to_pos_local(positionCur);
        printf(" ");
        _set_cur_to_pos_local(positionCur);
        break;
    case KEY_BACKSPACE:
```

```
        {
            if (current_pos > 0 && in_sym_count > 0) {
                {
                    buff[current_pos] = '\0';
                    current_pos--;
                    in_sym_count--; int flag = 0;
                    if (current_pos <= 4 && current_pos > 2) {
                        if (cur_selector > 2) {
                            cur_selector--;
                            flag = 1;
                        }
                    }
                    else if (current_pos < 2) {
                        if (cur_selector > 1) {
                            cur_selector--;
                            flag = 1;
                        }
                    }
                    _get_con_info_local(&con_inf);
                    positionCur.X = con_inf.dwCursorPosition.X;
                    if (flag) {
                        positionCur.X--;  positionCur.X--;
_set_cur_to_pos_local(positionCur);
                        printf(" ");  printf(" ");
                        _set_cur_to_pos_local(positionCur);
                    }
                    else {
                        positionCur.X--;  _set_cur_to_pos_local(positionCur);
                        printf(" ");
                        _set_cur_to_pos_local(positionCur);
                    }

                }
            }
        }
        break;
        default: {
            c = convert_u8_to_1251(c);
            int flag = 0;
            for (int i = 0; i < n_count; i++)
            {
                if (c == correct_sym[i]) {
                    flag = 1;
                    break;
                }
            }
            if (flag) {
                if (cur_selector < 4) {
                    if (in_sym_count >= 2 && in_sym_count < 4) {
                        buff[in_sym_count-2] = c;
                    }
                    else if (in_sym_count >= 4){
                        buff[in_sym_count-4] = c;
                    }
                    else
                    buff[in_sym_count] = c;
                    printf("%c", c);
                    in_sym_count++;
                    if (in_sym_count == 2) {
                        *d = atoi(buff);
                        if (*d > 31) {
                        *d = 31;
                            _get_con_info_local(&con_inf);
                            positionCur.X = con_inf.dwCursorPosition.X;
```

```
                            positionCur.X--;  positionCur.X--;
_set_cur_to_pos_local(positionCur);
                            printf(" ");  printf(" ");
                            _set_cur_to_pos_local(positionCur);
                            printf("%d", *d);
                        }
                        printf(".");
                        memset(buff, 0, strlen(buff));
                        cur_selector++;
                    }
                    else if (in_sym_count == 4) {
                        *m = atoi(buff);
                        if (*m > 12) { *m = 12;
                        _get_con_info_local(&con_inf);
                        positionCur.X = con_inf.dwCursorPosition.X;
                        positionCur.X--;  positionCur.X--;
_set_cur_to_pos_local(positionCur);
                        printf(" ");  printf(" ");
                        _set_cur_to_pos_local(positionCur);
                        printf("%d", *m);
                        }
                        printf(".");
                        memset(buff, 0, strlen(buff));
                        cur_selector++;
                    }
                    else if (in_sym_count == 8) {
                        // printf(".");
                        *y = atoi(buff);
                        time_t  now = time(0);
                        struct tm* ltm = localtime(&now);
                        if (*y > ltm->tm_year + 1900) { *y = ltm->tm_year + 1900;
                        _get_con_info_local(&con_inf);
                        positionCur.X = con_inf.dwCursorPosition.X;
                        positionCur.X-=4; _set_cur_to_pos_local(positionCur);
                        printf("    ");
                        _set_cur_to_pos_local(positionCur);
                        printf("%d", *y);
                        }
                        return KEY_ENTER;
                    }
                    current_pos++;
                }
            }
        }
            break;
        }
      }
    }
```

Исходный код файла «tree_operation.cpp»

```
#include "tree_operation.h"
#include "VicMenuDLL.h"
#include <conio.h>
#include <time.h>
#include <Windows.h>
#include <stdlib.h>
#include <stdio.h>

abonent* tree_getLeafById(abonent* root, const int id)
{
    if (root == NULL)
        return NULL;
```

```
    else if (root->info.id == id)
        return root;
    else if (root->info.id < id)
        return tree_getLeafById(root->right, id);
    else
        return tree_getLeafById(root->left, id);
}

void tree_deleteNodeById(abonent** root, const int id)
{
    if (root == NULL || *root == NULL)
        return; // выход если пустой узел

    if ((*root)->info.id == id)
    {
        if ((*root)->right == NULL)
        {
            abonent* t = *root;
            *root = (*root)->left;
            free(t);
        }
        else
        {
            if ((*root)->right->left == NULL)
            {
                abonent* tmp = (*root)->right;
                (*root)->right->left = (*root)->left;
                free(*root);
                *root = tmp;
            }
            else
            {
                abonent** candidate_prev = &((*root)->right);
                for (; (*candidate_prev)->left->left != NULL; candidate_prev =
&((*candidate_prev)->left));

                abonent* candidate = (*candidate_prev)->left;
                (*candidate_prev)->left = candidate->right;
                candidate->left = (*root)->left;
                candidate->right = (*root)->right;
                free(*root);
                *root = candidate;
            }
        }
    }
    else
    {
        //бинарный поиск в левом или правом поддереве
        tree_deleteNodeById
        (
            (*root)->info.id > id ?
            &((*root)->left) :
            &((*root)->right),
            id
        );
    }
}



int tree_getNodeCount(const abonent* root, const int accum) {
   if (root != NULL)
   return tree_getNodeCount(root->left, accum) + 1 + tree_getNodeCount(root->right,
accum);
```

```
    else
        return 0;
}


void View(abonent* top, int offset) {
    if (top)
    {
        offset += 3; //отступ от края экрана
        View(top->right, offset); //обход правого поддерева
        for (int i = 0; i < offset; i++) printf(" ");
        printf("|%d\n", top->info.id);
        View(top->left, offset); //обход левого поддерева
    }
}

void tree_add(abonent** root, const abonent_t* info)
{
    if (root == NULL || info == NULL)
    {
        return;
    }

    if (*root == NULL)
    {
        *root = (abonent*)calloc(sizeof(abonent), 1);

        if (*root != NULL)
        {
            (*root)->info = *info;
        }
    }
    else
    {
        /*if (((*root)->info.id == info->id)) {
            _message_window("Запись с таким id уже существует\0");
            Sleep(2000);
            return;
        }
        else */
        if (((*root)->info.id > info->id))
            tree_add(&((*root)->left), info);
        else
            tree_add(&((*root)->right), info);
    }
}

abonent* tree_delete(abonent* root) {
    if (root) {
        if (root->left) {
            tree_delete(root->left);
        }
        if (root->right) {
            tree_delete(root->right);
        }
        free(root);
    }
    return NULL;
}

void printToFile(FILE* f, abonent* root)
{
    if (root) {
        abonent_t te = root->info;
```

```
        fwrite(&te, size_abonent_t, 1, f);
        if (root->left) {
            printToFile(f, root->left);
        }
        if (root->right) {
            printToFile(f, root->right);
        }

    }
}

void printToFile_Text(FILE* f, abonent* root)
{
    if (root) {
        abonent_t te = root->info;
        fprintf(f, "%s %s %s %s\n%s %s %s\n%d %d %d %f\n", te.fio.surname,
te.fio.name, te.fio.secondname, te.book_name, te.autor.surname, te.autor.inicial,
te.izd, te.date_out.d, te.date_out.m, te.date_out.y, te.cost);
        if (root->left) {
            printToFile_Text(f, root->left);
        }
        if (root->right) {
            printToFile_Text(f, root->right);
        }

    }
}


abonent_t* _get_output_info(abonent* root, abonent_t* _output_memory, int* index) {
    {
        if (root) {
            if (root->left) {
                _get_output_info(root->left, _output_memory, index);
            }
            _output_memory[*index] = root->info;
            (*index)++;
            if (root->right) {
                _get_output_info(root->right, _output_memory, index);
            }
            return _output_memory;
        }
    }
}



dolgi_pers_t_obr* _get_dolgi_info(abonent* root, dolgi_pers_t_obr* _output_memory,
int d, int m, int y) {
    {
        if (root) {
            if (root->left) {
                _get_dolgi_info(root->left, _output_memory, d, m, y);
            }
            if ((((y * 365) + (m * 31) + d) - ((root->info.date_out.y) * 365) -
((root->info.date_out.m) * 31) - (root->info.date_out.d)) > 90 ){
                if (!_output_memory) {

                    _output_memory->info_mass =
(dolgi_pers_t*)calloc(sizeof(dolgi_pers_t), 1);
                    _output_memory->info_mass[0].id = root->info.id;
                    _output_memory->info_mass[0].fio = root->info.fio;
                    _output_memory->info_mass[0].count_dolg_books++;
```

```
                    _output_memory->count++;
                }
                else {
                    int flag = 0;
                    for (int i = 0; i < _output_memory->count; i++ ) {
                        if (_output_memory->info_mass[i].id == root->info.id) {
                            _output_memory->info_mass[i].count_dolg_books++;
                            flag = 1;
                        }
                    }
                    if (!flag) {
                        _output_memory->info_mass =
(dolgi_pers_t*)realloc(_output_memory->info_mass, sizeof(dolgi_pers_t) *
((_output_memory->count) + 1));
                        _output_memory->count++;
                        _output_memory->info_mass[(_output_memory->count) -
1].count_dolg_books = 1;
                        _output_memory->info_mass[(_output_memory->count) - 1].fio =
root->info.fio;
                        _output_memory->info_mass[(_output_memory->count) - 1].id =
root->info.id;
                    }
                }
            }
            if (root->right) {
                _get_dolgi_info(root->right, _output_memory, d,m,y);
            }
            //return _output_memory;
        }
        return _output_memory;
    }
}
```

Исходный код файла «VicMenuDLL.cpp»

```
#pragma warning(disable : 4996)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#include <conio.h>
#include <Windows.h>

#include "utf8.h"

#include "VicMenuDLL.h"
#include "MenuStruct.h"
#include "tree_operation.h"
#include "input_utils.h"

const int _otstup = 3; //Хранит отступ слева и справа в окне
const int _interval = 3; //хранит интервал сверху и снизу окна
int* _window_size; //указатель на массив размера
int _first_start = 1; //флаг первого запуска.
COORD positionCur = {4,4}; //Хранит текущую позицию курсора

#define clearf() system("cls"); // Полная очистка экрана
#define KEY_ENTER 13 // дефайны для кнопок
#define KEY_ESC 27

int page = 1; // хранит текущую страницу.
```

```
BOOL _get_con_info(CONSOLE_SCREEN_BUFFER_INFO* x) //Внутренняя функция, получает
информацию из буфера экрана.
{
    return GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), x);
}

BOOL _set_cur_to_pos(HANDLE h, COORD cor) { //Устанавливает курсор на необходимую
позицию
    return SetConsoleCursorPosition(h, cor);
}


void clear() { //функция кастомной очистки , чистит только окно
    positionCur.X = _otstup + 1;  positionCur.Y = _interval + 1; //установить
стартовые позиции
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); //получить хендлер консоли
    int* temp_window_size = NULL; //получить размеры окна
    temp_window_size = _get_window_size(temp_window_size);
    _set_cur_to_pos(hConsole, positionCur); // установить курсор в позицию
    _print_border(temp_window_size[0], temp_window_size[1]); // напечатать внутренний
бордер
    positionCur.X = _otstup + 1;  positionCur.Y = _interval + 1; //данные курсора
обнулить в исходное состояние
        _set_cur_to_pos(hConsole, positionCur); // установить курсор в исходное
состояние
}



void clear_table() { //Внутренняя функция очистки таблицы
    positionCur.X = _otstup + 2;  positionCur.Y = _interval + 3; //ставим курсор
согласно параметрам положения таблицы
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); //получаем хендлер консоли
    int* temp_window_size = NULL; // получаем размер кона
    temp_window_size = _get_window_size(temp_window_size);
    _set_cur_to_pos(hConsole, positionCur); int _window_h = temp_window_size[1];
    int _window_w = temp_window_size[0]; //запомнинаем рахмеры , ставим курсор в
нужную позицию
    for (int y = _interval+3; y <= _window_h - _interval-3; y++) { // цикл очистки
таблицы
        _set_cur_to_pos(hConsole, positionCur);
        for (int x = _otstup + 2; x < _window_w - _otstup-1; x++) {
             printf(" "); //_otstup+2,_interval+3
        }
        positionCur.Y++;
    }
}



void clear_for_info() { //установка курсора в позицию для вывода информации
    positionCur.X = _otstup + 4;  positionCur.Y = _interval + 4;
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    _set_cur_to_pos(hConsole, positionCur);
}


int* _get_curent_selection(char  c // Символ клавиатуры
    ,int * position // Массив в котором хранятся x и y
    ,int MaxY // Максимальный количество строк
    , int Colums  // Количество столбцов, по умолчанию - 1
    ,int _flag_x_readonly //можно ли менять x?
)
```

```
{
    int x = position[0]; int y = position[1];
    switch (c)
    {
    case KEY_ARROW_UP://вверх
        if (y > 1) y--;
        break;
    case KEY_ARROW_DOWN://вниз
        if (y < MaxY) y++;
        break;
    case KEY_ARROW_LEFT://лево
        if (x > 1) x--;
        if(_flag_x_readonly) if (y > 1) y--;
        break;
    case KEY_ARROW_RIGHT://право
        if (x < Colums) x++;
        if (_flag_x_readonly) if (y < MaxY) y++;
        break;
    }
    if (!_flag_x_readonly)
     position[0] = x; //запоминаем позицию по столбцу
    position[1] = y;//позиция по строке
    return position;
}

int _print_menu(_menu_item* _menu //Массив объектов  меню
    , int* position //Массив текущей позиции x и y
    , int _menu_size  //Колличество элементов в массиве объектов меню
    , int _menu_buttons,//Количество кнопок меню
    abonent_t * _output_mas, //массив для вывода информации
    int _output_colcount, // количество данных для вывода
    _tabel_metadata * table, //структура хранящая информацию о полях таблицы
    abonent** root, //указатель на указатель на корень дерева
    sort_struct* sort // структура, хранящая текущую сортировку
)
{
    int  table_focus_flag = 0; // флаг работы с таблицей
    int _padding; //оступ
    int _new_padding; //новый отступ
    int* _size_now = NULL; //текущий размер окна
    if (_first_start) { // Если у нас первый запуск
        _window_size = _get_window_size(_window_size); //порлучаем размер окна
        _print_bakground(_window_size[0], _window_size[1]); // вызываем отрисовку
заднего фона
        _first_start = 0; // первый запуск закончен
    }
    COORD positionCur = { _otstup + 1, _interval + 1 }; //позиция x и y у курсора.
Стартовый отступ
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); //получаем хендлер консоли
    _set_cur_to_pos(hConsole, positionCur);// установим курсор в позици.
    CONSOLE_SCREEN_BUFFER_INFO info_x; //необходимо для получения размера консоли
    int _max_subm_lenght = 0;  //поиск максимальной длины записи в сабменю
    for (int i = 0; i < _menu_size; i++) { //цикл прохода по всем записям в меню
        if (_menu[i]._menu_size > 0) {
            _max_subm_lenght = _menu[i]._sub_menu_lenght[0];
            for (int j = 1; j < _menu[i]._menu_size; j++) {
                if (_menu[i]._sub_menu_lenght[j] > _max_subm_lenght) {
                    _max_subm_lenght = _menu[i]._sub_menu_lenght[j];
                }
                _menu[i]._max_sub_lenght = _max_subm_lenght; //получаем максимальный
размер в сабменю, который будет использован для его вывода в дальнейшем
            }
        }
    }
```

```
    while (1) //цикл отрисовки меню
    {
        _size_now = _get_window_size(_size_now); //получение текущего размера окна
        if ((_size_now[0] != _window_size[0]) || (_size_now[1] != _window_size[1]))
//ели текузий размер не совпал с стартовым
        {
            _window_size[0] = _size_now[0];  _window_size[1] = _size_now[1]; //
запоминаем текущий размер
            clearf(); // очистка экрана
            _print_bakground(_window_size[0], _window_size[1]); //по новой
отрисовываем задний фон и рабочую область
            positionCur.X = _otstup + 1; positionCur.Y = _interval + 1; // задаем
базовые стартовые параметры курсора
            _set_cur_to_pos(hConsole, positionCur); //устанавливаем курсор в
начальное положение
            page = 1;
        }
        positionCur.X = _otstup + 1; positionCur.Y = _interval + 1;
        _set_cur_to_pos(hConsole, positionCur);//устанавливаем курсор в начальное
положение
        for (int i = 0; i < _menu_size; i++) { //Пока у нас есть элементы в меню
            _get_con_info(&info_x); // получаем текущее положение
            _padding = info_x.dwCursorPosition.X - 1; //задаем текущий падинг
            if (position[0] == i + 1) { //если у нас позиция курсора-указателя
совпадает с текущей позицией
                printf("\x1b[43m %s \x1b[0m", _menu[i]._name); // выделяем цветом
            }
            else printf(" %s ", _menu[i]._name); // иначе просто выводим на экран
            printf("│");  //разделитель
            _get_con_info(&info_x);
            positionCur.X = info_x.dwCursorPosition.X - 1; // получаем текущую
позицию курсора, сместив его на один символ назад
            _new_padding = info_x.dwCursorPosition.X - 1;  //запоминаем новый
падинг(оступ)
            positionCur.Y -= 1; // подымаем курсор на 1
            _set_cur_to_pos(hConsole, positionCur);  //Установили курсор
            printf("┬"); // выводим разделитель между двумя
            _get_con_info(&info_x); //получаем снова информацию о курсоре
            positionCur.Y += 2; positionCur.X = _padding; //смещаем курсор вниз на
две ячейки, установив его на оступ
            _set_cur_to_pos(hConsole, positionCur);
            if (_padding == _otstup) // печатаем разделитель
            {
                printf("├"); positionCur.X++;
            }
            else {
                positionCur.X++; //иначе увеличели X
                _set_cur_to_pos(hConsole, positionCur);
            }
            for (int j = positionCur.X; j < _new_padding; j++) { //отрисовка нижнего
отделителя
                printf("─");
                            }
            if (i == _menu_size - 1) {
                printf("┘"); // если элемент последний , отрисовать уголок
            }
            else
                printf("┴"); //иначе отрисовка отделителя
            positionCur.Y -= 1; positionCur.X = _new_padding + 1;
            _set_cur_to_pos(hConsole, positionCur);
        }
        if (!table_focus_flag)  //если у нас флаг не активен
```

```
        print_help("\x1b[45mESC\x1b[0m:Выход \x1b[45mENTER\x1b[0m:Ввод
\x1b[45mСТРЕЛКИ\x1b[0m: Переключение селектора меню \x1b[45mTAB\x1b[0m:Переключить
фокус на таблицу "); //печатаем один хелп
        else  print_help("\x1b[45mESC\x1b[0m:Выход \x1b[45mENTER\x1b[0m:Редактировать
\x1b[45mСТРЕЛКИ\x1b[0m:Навигация  \x1b[45mTAB\x1b[0m:Фокус на меню
\x1b[45mDEL\x1b[0m: Удалить запись
\x1b[45mHOME|END|PgUp|PgDown\x1b[0m:Сортировка");// иначе другой

_table_window(table,_output_mas,&_output_colcount,&page,&table_focus_flag,root,sort);
// вызываем печать окна таблицы
        char c = getch(); // получаем селектор меню
        if (c == KEY_TAB) {   // если таб то переход на таблицу с проверкой
            if (_output_mas && _output_colcount != 0)
                table_focus_flag = 1; else {
                print_help("\x1b[41mДанных для вывода нет. Невозможно переключиться
на таблицу.\x1b[0m");
                Sleep(1000);
            }
        } else
        if (c == KEY_ENTER) { // если ввели enter
            if (_menu[position[0] - 1]._menu_size > 0) { //если в текущем пункте меню
есть сабменю , то
                position[1] = 1;
                while (1) {//бесконечный цикл работы с сабменю
                    positionCur.X = _menu[position[0] - 1]._menu_name_lenght +
2;//устанавливаем текущие позиции для курсора
                    positionCur.Y = _interval + 2;
                    _set_cur_to_pos(hConsole, positionCur);
                    for (int i = 0; i <= _menu[position[0] - 1]._menu_size; i++) {
//цикл отрисовки каждого окошка меню
                        for (int j = 0; j <= _menu[position[0] - 1]._max_sub_lenght +
2; j++) { //цикл отрисовки бордера
                            if (j == 0) {
                                if (i == 0) {//если позиция 0 , то у нас верхние
ограничитель
                                    printf("┌");
                                }
                                else
                                    if (i == _menu[position[0] - 1]._menu_size) {
                                        printf("└"); //если конечная позиция , то
нижний
                                    }
                                    else printf("├"); // иначе соеденитель
                            }
                            else
                                if (j == _menu[position[0] - 1]._max_sub_lenght + 2)
{
                                    if (i == 0) {
                                        printf("┐"); //печать верхнего ограничителя
                                    }
                                    else
                                        if (i == _menu[position[0] - 1]._menu_size) {
                                            printf("┘"); //печать нижнего
ограничителя , если достигли конца
                                             _get_con_info(&info_x);
                                            positionCur.X =
info_x.dwCursorPosition.X;
                                            positionCur.Y =
info_x.dwCursorPosition.Y;
                                            positionCur.Y--; positionCur.X--;
                                            _set_cur_to_pos(hConsole, positionCur);
// ставим курсор на один назад и на один вверх
                                            printf("│"); //печать разделителя
                                            positionCur.Y++;
```

```
                                        _set_cur_to_pos(hConsole,
positionCur);//возврат на текущую позициию
                                    }
                                    else {
                                        printf("┤"); //печать соеденителя
                                         _get_con_info(&info_x);
                                        positionCur.X =
info_x.dwCursorPosition.X;
                                        positionCur.Y =
info_x.dwCursorPosition.Y;
                                        positionCur.Y--; positionCur.X--;
                                        _set_cur_to_pos(hConsole, positionCur);//
ставим курсор на один назад и на один вверх
                                        printf("│");// печать разделителя
                                        positionCur.Y++;
                                        _set_cur_to_pos(hConsole,
positionCur);//возврат на текущую позициию
                                    }
                                }
                                else (printf("─")); //печать промежуточного
разделителя
                            }
                            positionCur.Y++; positionCur.X = _menu[position[0] -
1]._menu_name_lenght + 2; //стартовые параметры для печати данных
                            _set_cur_to_pos(hConsole, positionCur); // установим курсор
на данную позицию
                            if (i == _menu[position[0] - 1]._menu_size) break; //если
текущий номер совпал с размером , то ничего не рисуем дальше
                            int _margin = _menu[position[0] - 1]._max_sub_lenght + 2 -
_menu[position[0] - 1]._sub_menu_lenght[i]; //высчитываем отступ
                            _margin = _margin / 2;
                            printf("│");//печатаем разделитель
                            for (int l = 0; l < _margin; l++) { printf(" "); } //делаем
отступ слева
                            if (position[1] - 1 == i) {
                                printf("\x1b[43m%s\x1b[0m", _menu[position[0] -
1]._sub_menu[i]); // если текущий выбор сабменю совпал , то выделяем его
                            }
                            else { printf("%s ", _menu[position[0] - 1]._sub_menu[i]); }
// иначе печатаем обычно
                            for (int l = 0; l < _margin; l++) { printf(" "); } // печать
отступа справа
                            positionCur.Y++;
                            _set_cur_to_pos(hConsole, positionCur); // увеличиваем y , и
ставим курсор на след позицию
                        }
                        c = getch(); //получаем нажатую кнопку

                        if (c == KEY_ENTER) { // если у нас нажат enter
                            int result = 0; //возвращаемое значение
                            for (int l = 0; l < position[0] - 1; l++) {
                                result += _menu[l]._menu_size; //до текущей позиции
высчитываем результат выбора
                            }
                            result += position[1]; // добавляем текущее смещение
                                        return result; // вернем результат
                                    }
                        if (c == KEY_ESC) { clear(); break; } // если escape , то очистим
экран меню , выйдем с цикла
                        position = _get_curent_selection(c, position, _menu[position[0] -
1]._menu_size, _menu_buttons, 1); //получение новой позиции курсора согласно с
                    }
                }
```

```
            else return PROGRAM_EXIT;   //Вернте выход из программы если в текущем
меню 0 сабменю
            }
        else { //внутренний цикл работы будет получать новую позицию курсора
            position = _get_curent_selection(c, position, 1, _menu_buttons, 0);
        }
    }

    return EXIT_SUCCESS; //вернет успешный выход
}
/// <summary>
/// Анимированная картинка в консоли
/// </summary>
void animatedNeko() {
    FILE* f; int count = 1;
    while (1) {
        for (int i = 0; i < 11; i++) {
            char url[256] = { 0 };
            sprintf(url, "bakemonogatari-monogatari/banner (%d).txt", i);
            f = fopen(url, "r"); char a[210];
            while (fgets(a,210,f) != NULL)
            {
                printf("         %s", a);
            }
            Sleep(30);
            COORD positionCur = { 0,0 }; //позиция x и y
            HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
            _set_cur_to_pos(hConsole, positionCur);
        }
        count++;
        if (count >= 20) return;
    }
}

void _print_bakground(int _window_w,int _window_h)
{
    clearf();
    char c =  rand() % (47 - 33 + 1) + 33;
    srand(time(NULL));
    for (int i = 0; i < _window_h; i++)
    {
        for (int j = 0; j < _window_w; j++) {
            printf("\x1b[44m%c\x1b[0m",c);
            c= rand() % (47 - 33 + 1) + 33;
        }
        printf("\n");
    }
    _print_border(_window_w, _window_h);
}

void _print_border(int _window_w, int _window_h) {
    COORD positionCur = { _otstup,_interval }; //позиция x и y
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    for (int y = _interval; y <= _window_h - _interval; y++) {
        _set_cur_to_pos(hConsole, positionCur);
        if(y == _interval)  printf("┌");
        else if (y == _window_h - _interval)
            printf("└");
        else
        printf("│");
        for (int x = _otstup+1; x < _window_w - _otstup; x++) {
            if ((y == _interval) || (y == _window_h - _interval)) printf("─"); else
printf(" ");
        }
```

```
        if (y == _interval)  printf("┐");
        else if (y == _window_h - _interval)
            printf("┘");
        else
            printf("│");
        positionCur.Y++;
    }
}



int* _get_window_size(int* size) {
    HANDLE hWndConsole;
    size = (int*)calloc(2, sizeof(int));
    if (hWndConsole = GetStdHandle(-12))
    {
        CONSOLE_SCREEN_BUFFER_INFO consoleInfo;
        if (GetConsoleScreenBufferInfo(hWndConsole, &consoleInfo))
        {
            size[0] = consoleInfo.srWindow.Right - consoleInfo.srWindow.Left + 1;
            size[1] = consoleInfo.srWindow.Bottom - consoleInfo.srWindow.Top + 1;
        }
        else
            printf("Error: %d\n", GetLastError());
    }
    return size;
}



void _window(int _window_w, int _window_h, char* title) {
    int height = _window_h / 4; int width = _window_w / 4;
    COORD positionCur = { _otstup,_interval }; //позиция x и y
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    int _center_x, _center_y;
    _center_x = _window_w / 2;
    _center_y = _window_h / 2;
    positionCur.X = _center_x - width / 2;
    positionCur.Y = _center_y - height / 2;
    for (int y = 0; y < height; y++) {
        _set_cur_to_pos(hConsole, positionCur);
        for (int x = 0; x < width; x++) {
            if (y == 0) {
                if (x == 0) {
                    printf("┌");
                }
                else if (x == width - 1) { printf("┐"); }
                else   printf("─");

            }
            else
                if (y == height - 1)
                {
                    if (x == 0) {
                        printf("└");
                    }
                    else if (x == width - 1) {
                        printf("┘");
                    }
                    else printf("─");
                }
                else {
                    if (x == 0) {
                        printf("│");
```

```
                    }
                    else if (x == width - 1) { printf("│"); }
                    else printf(" ");
                }
            }
            positionCur.Y++;
        }
        positionCur.X = _center_x - width / 2 + 1;
        positionCur.Y = _center_y - height / 2 + 1;
        _set_cur_to_pos(hConsole, positionCur);
        int _temp_ots=0;
        if (title) _temp_ots = (width - 2 - u8_strlen(title)) / 2;
        for (int j = 0; j < _temp_ots; j++) {
            printf(" ");
        }
        if (title)  printf("%s", title);
        for (int j = 0; j < _temp_ots; j++) {
            printf(" ");
        }
        positionCur.X = _center_x - width / 2;
        positionCur.Y = _center_y - height / 2 + 2;
        _set_cur_to_pos(hConsole, positionCur);
        for (int j = 0; j < width; j++) {
            if (j == 0) printf("├");
            else if (j == width - 1) {
                printf("┤");
            }
            else printf("─");
        }
    }

    int _confirm_window(char * message)
    {
        SetConsoleOutputCP(65001);
        if (!message)  message = "Выполнить операцию ?";
        int* _size_n = NULL;
        _size_n = _get_window_size(_size_n);
        int _window_w = _size_n[0]; int _window_h = _size_n[1];
        _window(_window_w, _window_h, "Подтверждение");
        CONSOLE_SCREEN_BUFFER_INFO info_x;
        int height = _window_h / 4; int width = _window_w / 4;
        COORD positionCur = { _otstup,_interval }; //позиция x и y
        HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
        int _center_x = _window_w / 2;
        int _center_y = _window_h / 2;
        positionCur.X = _center_x - width / 2;
        positionCur.Y = _center_y - height / 2 + height / 4+2;
        int m_lenght = u8_strlen(message);
        int margin = 0;
        /*margin = width - 21;
        margin = margin / 2;
        positionCur.X += margin;
        _set_cur_to_pos(hConsole, positionCur);
        printf("%s");*/

        if (m_lenght < width - 2) {
            int margin = width - m_lenght;
            margin = margin / 2;
            positionCur.X += margin;
            _set_cur_to_pos(hConsole, positionCur);
            printf("%s", message);
        }
        else {
            positionCur.X++;
```

```
        _set_cur_to_pos(hConsole, positionCur);
        for (int i = 0; i < (width - 1) * 2; i++) {
            printf("%c", message[i]);
            _get_con_info(&info_x);
            if (info_x.dwCursorPosition.X - positionCur.X >= width - 5) {
                printf("..."); break;
            }
        }
    }
    int _selection = 1;
    positionCur.Y = _center_y - height / 2 + height / 4 + height / 3+2;
    while (1) {
        positionCur.X = _center_x - width / 2 + 3;
        _set_cur_to_pos(hConsole, positionCur);
        if (_selection) {
            printf(" \x1b[43mДА\x1b[0m ");
            positionCur.X = _center_x + width / 2-8;
            _set_cur_to_pos(hConsole, positionCur);
            printf(" НЕТ ");

        }
        else {
            printf(" ДА ");
            positionCur.X = _center_x + width / 2 - 8;
            _set_cur_to_pos(hConsole, positionCur);
            printf(" \x1b[43mНЕТ\x1b[0m ");

        }
        char c = getch();
        switch (c) {
        case 75://лево
            if (_selection != 1) _selection++;
            break;
        case 77://право
            if (_selection == 1) _selection--;
            break;
        case KEY_ENTER:
            return _selection;
            break;
        case KEY_ESC:
            return 0;
            break;
        }
    }

    return EXIT_SUCCESS;
}

void _in_window() {
    int* _size_n = NULL;
    _size_n = _get_window_size(_size_n);
    int _window_w = _size_n[0]; int _window_h = _size_n[1];
    _window(_window_w, _window_h,"Окно ввода");
    int height = _window_h / 4; int width = _window_w / 4;
    COORD positionCur = { _otstup,_interval }; //позиция x и y
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    int _center_x = _window_w / 2;
    int _center_y = _window_h / 2;
    positionCur.X = _center_x - width / 2;
    positionCur.Y = _center_y;
    int margin = width - 21;
    margin = margin / 2;
    positionCur.X += margin;
    _set_cur_to_pos(hConsole, positionCur);
```

```
    printf("Введите данные -> ");
}

void _message_window(char* message) {
    int* _size_n = NULL;
    _size_n = _get_window_size(_size_n);
    int _window_w = _size_n[0]; int _window_h = _size_n[1];
    _window(_window_w, _window_h,"Сообщение");
    CONSOLE_SCREEN_BUFFER_INFO info_x;  HANDLE hConsole =
GetStdHandle(STD_OUTPUT_HANDLE);
    int m_lenght = u8_strlen(message);
    int height = _window_h / 4; int width = _window_w / 4;
    COORD positionCur = { _otstup,_interval }; //позиция x и y

    int _center_x = _window_w / 2;
    int _center_y = _window_h / 2;
    positionCur.X = _center_x - width / 2;
    positionCur.Y = _center_y;
    if (m_lenght < width - 2) {
        int margin = width - m_lenght;
        margin = margin / 2;
        positionCur.X += margin;
        _set_cur_to_pos(hConsole, positionCur);
        printf("%s", message);
    }
    else {
        positionCur.X++;
        _set_cur_to_pos(hConsole, positionCur);
        for (int i = 0; i < (width - 1)*2; i++) {
            printf("%c",message[i]);
              _get_con_info(&info_x);
            if (info_x.dwCursorPosition.X - positionCur.X >= width - 5) {
                printf("..."); break;
            }
        }
      }


}

int _table_window(_tabel_metadata * table, abonent_t * _output_mass, int *
_info_count, int*  page, int * _table_focus_flag, abonent** root, sort_struct* sort)
{
    CONSOLE_SCREEN_BUFFER_INFO info_x;  HANDLE hConsole =
GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO structCursorInfo;
    GetConsoleCursorInfo(hConsole, &structCursorInfo);
    structCursorInfo.bVisible = FALSE;
    SetConsoleCursorInfo(hConsole, &structCursorInfo);
    static int row_selection[] = {1,1};
    int* _size_n = NULL;
    _size_n = _get_window_size(_size_n);
     int _window_w = _size_n[0]; int _window_h = _size_n[1];
    int _padding, _new_padding; int _mn_size_flag = 0; int _size_temp = 0;
    int _size_delta = 0; static int last_sel = 0;
    for (int i = 0; i < table->_col_count; i++) {
        _size_delta += table->_cols[i].size;
    }
    int height = _window_h - _interval * 2 - 10 ; int width = _window_w - _otstup * 2
- 4;
    _size_delta = (width -2) - _size_delta - table->_col_count*2-5 ;
    do {
    COORD positionCur = { _otstup+2,_interval+3}; //позиция x и y
    _set_cur_to_pos(hConsole, positionCur);
```

```
    for (int i = 0; i < width; i++) {
        if (i == 0) {
            printf("┌");
        }
        if (i == width - 1) {
            printf("┐");
        } else
        printf("─");
    }
    positionCur.Y++; //positionCur.X++;
    _set_cur_to_pos(hConsole, positionCur);
    printf("├");
    for (int i = 0; i < table->_col_count; i++) {
         _get_con_info(&info_x);
        _padding = info_x.dwCursorPosition.X - 1;
        if (i == sort->sort_f && i!=0)
           if (sort->sort_t == UP)
              printf(" \x1b[44m%s\x1b[0m ", table->_cols[i].name);
           else printf(" \x1b[43m%s\x1b[0m ", table->_cols[i].name);
        else printf(" %s ", table->_cols[i].name);
        if (table->_cols[i].resizebl) {
            if (!_mn_size_flag) {
                int _tmp_padding = _size_delta / 3;
                _size_delta -= _tmp_padding;
                _mn_size_flag = 1;
                if ((u8_strlen(table->_cols[i].name) + _tmp_padding) != table-
>_cols[i].size) {
                    table->_cols[i].size += _tmp_padding;
                }
            }
            else {
                if ((u8_strlen(table->_cols[i].name) + _size_delta) != table-
>_cols[i].size) {
                    table->_cols[i].size += _size_delta;
                }
            }
        }

        if (u8_strlen(table->_cols[i].name) != table->_cols[i].size) {
            int _sim_padding = table->_cols[i].size - u8_strlen(table-
>_cols[i].name);
            for (int i = 0; i < _sim_padding; i++) {
                printf(" ");
            }
        }

        printf("│");
         _get_con_info(&info_x);
        positionCur.X = info_x.dwCursorPosition.X - 1;
        _new_padding = info_x.dwCursorPosition.X - 1;
        positionCur.Y -= 1;
        _set_cur_to_pos(hConsole, positionCur);
        if (i == table->_col_count - 1) {
            printf("┐");
        }
        else
            printf("┬");
         _get_con_info(&info_x);
        positionCur.Y += 2; positionCur.X = _padding;
        _set_cur_to_pos(hConsole, positionCur);
        if (_padding == _otstup+2)
        {
            printf("├"); positionCur.X++;
        }
```

```
        else {
            positionCur.X++;
            _set_cur_to_pos(hConsole, positionCur);
        }
        for (int j = positionCur.X; j < _new_padding; j++) {
            printf("─");
        }
        if (i == table->_col_count - 1) {
            printf("┤");
        }
        else
            printf("┼");
        positionCur.Y -= 1; positionCur.X = _new_padding + 1;
        _set_cur_to_pos(hConsole, positionCur);
    }
    positionCur.Y += 2;
    positionCur.X = _otstup + 2;
    _set_cur_to_pos(hConsole, positionCur);

        if (_output_mass && *_info_count!=0) {

            int _col_inpage = height;
            int _diap[2] = { 0,0 };
            _diap[0] = ((*page) - 1) * _col_inpage;
            _diap[1] = _diap[0] + _col_inpage;
          //  if (row_selection[1]== last_sel)
            for (int j = _diap[0]; j < _diap[1]; j++)
            {
                if (*_info_count <= j) break;
                _set_cur_to_pos(hConsole, positionCur);
                if (*_table_focus_flag)
                    if (j == ((*page) - 1) * _col_inpage + row_selection[1]-1)
                        printf("\x1b[42m");
                printf("│");
                char buff[400] = { "" };
                sprintf(buff, "%d", j+1);
                SetConsoleOutputCP(1251); //-------
                printf("%s", buff);
                if (u8_strlen(buff) < 3)
                {
                    for (int l = 0; l < 3 - u8_strlen(buff); l++)
                        printf(" ");
                }
                SetConsoleOutputCP(65001); //-------
                printf("│");

                sprintf(buff, "%s %s %s", _output_mass[j].fio.surname,
_output_mass[j].fio.name, _output_mass[j].fio.secondname);
                if (u8_strlen(buff) > table->_cols[1].size + 2) {
                    sprintf(buff, "%s %c.%c", _output_mass[j].fio.surname,
_output_mass[j].fio.name[0], _output_mass[j].fio.secondname[0]);
                }
                SetConsoleOutputCP(1251); //-------
                printf("%s", buff);
                if (u8_strlen(buff) < table->_cols[1].size + 2)
                {
                    for (int l = 0; l < table->_cols[1].size + 2 - u8_strlen(buff);
l++)
                        printf(" ");
                }
                SetConsoleOutputCP(65001); //-------
                printf("│");
                SetConsoleOutputCP(1251); //-------
```

```
                sprintf(buff, "%s %s", _output_mass[j].autor.surname,
_output_mass[j].autor.inicial);
                if (u8_strlen(buff) > table->_cols[2].size + 2) {
                    for (int l = 0; l < table->_cols[2].size - 1; l++) {
                        printf("%c", buff[l]);
                    }
                    printf("...");
                }
                else
                    printf("%s", buff);
                if (u8_strlen(buff) < table->_cols[2].size + 2)
                {
                    for (int l = 0; l < table->_cols[2].size + 2 - u8_strlen(buff);
l++)
                        printf(" ");
                }
                SetConsoleOutputCP(65001); //-------
                printf("│");
                SetConsoleOutputCP(1251); //-------
                sprintf(buff, "%s", _output_mass[j].book_name);
                if (u8_strlen(buff) > table->_cols[3].size + 2) {
                    for (int l = 0; l < table->_cols[3].size - 1; l++) {
                        printf("%c", _output_mass[j].book_name[l]);
                    }
                    printf("...");
                }
                else
                    printf("%s", buff);
                if (u8_strlen(buff) < table->_cols[3].size + 2)
                {
                    for (int l = 0; l < table->_cols[3].size + 2 - u8_strlen(buff);
l++)
                        printf(" ");
                }
                SetConsoleOutputCP(65001); //-------
                printf("│");
                SetConsoleOutputCP(1251); //-------
                sprintf(buff, "%s", _output_mass[j].izd);
                if (u8_strlen(buff) > table->_cols[4].size + 2) {
                    for (int l = 0; l < table->_cols[4].size - 1; l++) {
                        printf("%c", _output_mass[j].izd[l]);
                    }
                    printf("...");
                }
                else
                    printf("%s", buff);
                if (u8_strlen(buff) < table->_cols[4].size + 2)
                {
                    for (int l = 0; l < table->_cols[4].size + 2 - u8_strlen(buff);
l++)
                        printf(" ");
                }
                SetConsoleOutputCP(65001); //-------
                printf("│");
                SetConsoleOutputCP(1251); //-------
                sprintf(buff, "%d.%d.%d", _output_mass[j].date_out.d,
_output_mass[j].date_out.m, _output_mass[j].date_out.y);
                if (u8_strlen(buff) > table->_cols[5].size + 2) {
                    for (int l = 0; l < table->_cols[5].size - 1; l++) {
                        printf("%c", buff[l]);
                    }
                    printf("...");
                }
                else
```

```
                printf("%s", buff);
                if (u8_strlen(buff) < table->_cols[5].size + 2)
                {
                    for (int l = 0; l < table->_cols[5].size + 2 - u8_strlen(buff);
l++)
                        printf(" ");
                }
                SetConsoleOutputCP(65001); //-------
                printf("│");
                SetConsoleOutputCP(1251); //-------
                sprintf(buff, "%.2f", _output_mass[j].cost);
                if (u8_strlen(buff) > table->_cols[6].size + 2) {
                    for (int l = 0; l < table->_cols[6].size - 1; l++) {
                        printf("%c", buff[l]);
                    }
                    printf("...");
                }
                else
                    printf("%s", buff);
                if (u8_strlen(buff) < table->_cols[6].size + 2)
                {
                    for (int l = 0; l < table->_cols[6].size + 2 - u8_strlen(buff);
l++)
                        printf(" ");
                }
                SetConsoleOutputCP(65001); //-------
                printf("│");
                if (j == ((*page) - 1) * _col_inpage + row_selection[1] - 1)
                    printf("\x1b[0m");
                positionCur.Y++;
            }
        }
        else {
        printf("│");
        for (int i = 0; i < (width - 12) / 2; i++)
            printf("-");
        SetConsoleOutputCP(65001); //-------
        printf(" Данных нет ");
        SetConsoleOutputCP(65001); //-------
        for (int i = 0; i < (width - 12) / 2 - 1; i++)
            printf("-");
        if ((width - 12) % 2 == 1) printf("-");
        printf("│");
        positionCur.Y++;
        *_table_focus_flag = 0;
        }

        _set_cur_to_pos(hConsole, positionCur);
        int _row_num = 0; int _padd_border = table->_cols[_row_num].size + 2;
        for (int i = 0; i < width; i++) {
            if (i == 0) {
                printf("└");
            }
            if (i == width - 1) {
                printf("┘");
            }
            else
                if (i == _padd_border) {
                    _row_num++;
                    if (_row_num < table->_col_count)
                        _padd_border += table->_cols[_row_num].size + 3;
                    printf("┴");
                }
                else
```

```
                printf("—");
        }
        COORD temp_cord = positionCur;
        temp_cord.Y++;
        _set_cur_to_pos(hConsole,temp_cord);
        char message[200] = "";
        //if(( * _info_count) != 0)
        sprintf(message, "Страница %d из %d",*page,(*_info_count/height)+1);
        printf(message);


        if (*_table_focus_flag)
        {
            char c = getch();

            if (c == KEY_ENTER) {
                _output_mass[((*page) - 1) * height + row_selection[1] - 1] =
*_in_info_window(table, &_output_mass[((*page) - 1) * height + row_selection[1] - 1],
0);
                tree_deleteNodeById(root, _output_mass[((*page) - 1) * height +
row_selection[1] - 1].id);
                _output_mass[((*page) - 1) * height + row_selection[1] - 1].id =
util_hashCodeFromFio(&_output_mass[((*page) - 1) * height + row_selection[1] -
1].fio);
                tree_add(root, &_output_mass[((*page) - 1) * height +
row_selection[1] - 1]);
                clear_table();
            }


            if (c == KEY_ESC || c == KEY_TAB)
            {
                *_table_focus_flag = 0;
                break;
            }

            if (c == KEY_DEL) {
                if (_confirm_window("Вы действительно хотите удалить запись ?")) {
                 // root =  tree_deleteNodeById(root, _output_mass[((*page) - 1) *
height + row_selection[1] - 1].id);
                    tree_deleteNodeById(root, _output_mass[((*page) - 1) * height +
row_selection[1] - 1].id);
                    _message_window("Запись успешно удалена");
                    int leafcount = tree_getNodeCount(*root, 0);
                    int  temp = 0;
                    _output_mass = _get_output_info(*root, _output_mass, &temp);
                    if (leafcount == 0)
                        _output_mass = NULL;
                    *_info_count = leafcount;
                    if (*_info_count < ((*page) * height)) {
                        clear_table();
                        if ((((*page) - 1) * height + row_selection[1] - 1) >
*_info_count) {
                            row_selection[1] = *_info_count - ((*page) - 1) * height;
                        }
                        if (((*_info_count) * (*page)) % height == 0){
                            if (*page > 1) { (*page)--; clear_table(); }
}

                 }
                }
              }

            if (c == KEY_HOME)
```

```
            {
                if (sort->sort_f > 0) {
                    sort->sort_f--;
                    _output_mass = _sort_output(_output_mass, _info_count, sort);
                }
            }
            if (c == KEY_END)
            {
                if (sort->sort_f < ZADANIE-1) {
                    sort->sort_f++;
                    _output_mass = _sort_output(_output_mass, _info_count, sort);
                }
            }
            if (c == KEY_PGUP)
            {
                sort->sort_t = UP;
                    _output_mass = _sort_output(_output_mass, _info_count, sort);
            }
            if (c == KEY_PGDOWN)
            {
                 sort->sort_t = DOWN;
                    _output_mass = _sort_output(_output_mass, _info_count, sort);

            }
            if (c == KEY_ARROW_LEFT)
            {
                if (*page > 1) { (*page)--; clear_table(); }

            }
            if (c == KEY_ARROW_RIGHT)
            {
                if (*_info_count > ((*page) * height)) {
                    (*page)++;
                    clear_table();
                    if ((((*page) - 1) * height + row_selection[1] - 1) >
*_info_count) {
                        row_selection[1] = *_info_count - ((*page) - 1) * height;
                    }
                }
            }
            int* temp = (int*)calloc(2, sizeof(int));
            temp = _get_curent_selection(c, row_selection, height, 1, 0);
            row_selection[1] = temp[1];
            if ((((*page) - 1) * height + row_selection[1] - 1) == *_info_count) {
                row_selection[1] = row_selection[1] - 1;
            }
        }
    }while (*_table_focus_flag);
     return EXIT_SUCCESS;
}


void _big_window(char* title) {
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    int* _size_n = NULL;
    _size_n = _get_window_size(_size_n);
    int _window_w = _size_n[0]; int  _window_h = _size_n[1];
    int _padding, _new_padding; int _mn_size_flag = 0; int _size_temp = 0; char
buff[200];
    int height = _window_h / 2;
    int width = _window_w / 2;
    int _center_x = _window_w / 2;
    int _center_y = _window_h / 2;
    positionCur.X = _center_x - width / 2;
```

```
    positionCur.Y = _center_y - height / 2;
    for (int y = 0; y < height; y++) {
        _set_cur_to_pos(hConsole, positionCur);
        for (int x = 0; x < width; x++) {
            if (y == 0) {
                if (x == 0) {
                    printf("┌");
                }
                else if (x == width - 1) { printf("┐"); }
                else   printf("─");

            }
            else
                if (y == height - 1)
                {
                    if (x == 0) {
                        printf("└");
                    }
                    else if (x == width - 1) {
                        printf("┘");
                    }
                    else printf("─");
                }
                else {
                    if (x == 0) {
                        printf("│");
                    }
                    else if (x == width - 1) { printf("│"); }
                    else printf(" ");
                }
        }
        positionCur.Y++;
    }

    positionCur.X = _center_x - width / 2 + 1;
    positionCur.Y = _center_y - height / 2 + 1;
    _set_cur_to_pos(hConsole, positionCur);
     int _temp_ots = 0;
     if (title) _temp_ots = (width - 2 - u8_strlen(title)) / 2;
    for (int j = 0; j < _temp_ots; j++) {
        printf(" ");
    }
    if(title)  printf("%s", title);
    for (int j = 0; j < _temp_ots; j++) {
        printf(" ");
    }
    positionCur.X = _center_x - width / 2;
    positionCur.Y = _center_y - height / 2 + 2;
    _set_cur_to_pos(hConsole, positionCur);
    for (int j = 0; j < width; j++) {
        if (j == 0) printf("├");
        else if (j == width - 1) {
            printf("┤");
        }
        else printf("─");
    }
}



abonent_t* _in_info_window(_tabel_metadata* table, abonent_t *_output_info,int
_cycle_in_flag) {
    CONSOLE_SCREEN_BUFFER_INFO info_x;  HANDLE hConsole =
GetStdHandle(STD_OUTPUT_HANDLE);
```

```
    int* _size_n = NULL; abonent_t *  _temp_info;
    if (_output_info ) {
        _temp_info = (abonent_t*)calloc(1, sizeof(abonent_t));
        *_temp_info = *_output_info;
    }
    else _temp_info = (abonent_t*)calloc(1, sizeof(abonent_t));
    _size_n = _get_window_size(_size_n);
    int _window_w = _size_n[0]; int  _window_h = _size_n[1];
    int _padding, _new_padding; int _mn_size_flag = 0; int _size_temp = 0; char
buff[200];
    int height = _window_h / 2; int width = _window_w / 2;
    int _center_x = _window_w / 2; int flag_clear = 0;
    int _center_y = _window_h / 2; char* title = NULL;
start:
    if (_cycle_in_flag) title = "Окно ввода информации"; else title = "Окно
редактирования информации";
    _big_window(title);
    int max_lenght = 0; int y_modifire = 1;
    if (height >= 20) y_modifire = 2;
    for (int i = 1; i < table->_col_count; i++) {
        if (u8_strlen(table->_cols[i].name) > max_lenght)
            max_lenght = u8_strlen(table->_cols[i].name);
    }

    char c;
    int* _men_position = (int*)calloc(2, sizeof(int));
    _men_position[0] = 1; _men_position[1] = 1;
    while (1) {
        if (flag_clear) { _big_window(title); flag_clear = 0; }
        positionCur.X = _center_x - width / 2 + 4;
        positionCur.Y = _center_y - height / 2 + 4;
        _set_cur_to_pos(hConsole, positionCur);
        for (int i = 1; i < table->_col_count; i++) {
            if (_men_position[1] == i) {
                // printf("\x1b[43m%s\x1b[0m", table->_cols[i].name);
                printf("%s", table->_cols[i].name);
            }
            else {
                printf("%s", table->_cols[i].name);
            }
            positionCur.Y += y_modifire;
            _set_cur_to_pos(hConsole, positionCur);
        }
        positionCur.X = _center_x - width / 2 + 4 + max_lenght + 1;
        positionCur.Y = _center_y - height / 2 + 4;
        _set_cur_to_pos(hConsole, positionCur);
        for (int i = 1; i < table->_col_count; i++) {
            if (_men_position[1] == i) {
               if(!_cycle_in_flag) printf("\x1b[43m --> \x1b[0m"); else printf(" -->
");
            }
            else {
                printf(" --> ");
            }
            positionCur.Y += y_modifire;
            _set_cur_to_pos(hConsole, positionCur);
        }
        _get_con_info(&info_x);
        int padding = info_x.dwCursorPosition.X; // получаем текущую позицию курсора,
сместив его на один символ назад
        padding -= _center_x;
        positionCur.X = _center_x - width / 2 + 4 + max_lenght + 6;
        positionCur.Y = _center_y - height / 2 + 4;
        _set_cur_to_pos(hConsole, positionCur);
```

```
for (int i = 1; i < table->_col_count; i++) {
    for (int j = 0; j < (width / 2 - padding - 8); j++) {
        printf("_");
    }
    positionCur.Y += y_modifire;
    _set_cur_to_pos(hConsole, positionCur);
}
positionCur.X = _center_x - width / 2 + 4 + max_lenght + 6;
positionCur.Y = _center_y - height / 2 + 4;
_set_cur_to_pos(hConsole, positionCur);
if (!_cycle_in_flag) {
    COORD fild_cords[6];
    SetConsoleOutputCP(1251); //-------
    if (u8_strlen(_temp_info->fio.surname) > 0)
        printf(" %s ", _temp_info->fio.surname);
    if (u8_strlen(_temp_info->fio.name) > 0)
        printf(" %s ", _temp_info->fio.name);
    if (u8_strlen(_temp_info->fio.secondname) > 0)
        printf(" %s ", _temp_info->fio.secondname);
    fild_cords[0] = positionCur;
    positionCur.Y += y_modifire;
    _set_cur_to_pos(hConsole, positionCur);
    if (u8_strlen(_temp_info->autor.surname) > 0)
        printf(" %s ", _temp_info->autor.surname);
    if (u8_strlen(_temp_info->autor.inicial) > 0)
        printf(" %s ", _temp_info->autor.inicial);
    fild_cords[1] = positionCur;
    positionCur.Y += y_modifire;
    _set_cur_to_pos(hConsole, positionCur);
    if (u8_strlen(_temp_info->book_name) > 0) {
        printf(" %s ", _temp_info->book_name);
    }
    fild_cords[2] = positionCur;
    positionCur.Y += y_modifire;
    _set_cur_to_pos(hConsole, positionCur);
    if (u8_strlen(_temp_info->izd) > 0) {
        printf(" %s ", _temp_info->izd);
    }
    fild_cords[3] = positionCur;
    positionCur.Y += y_modifire;
    _set_cur_to_pos(hConsole, positionCur);
    if (_temp_info->date_out.d > 0) {
        printf(" %d.", _temp_info->date_out.d);
        printf("%d.", _temp_info->date_out.m);
        printf("%d ", _temp_info->date_out.y);
    }
    fild_cords[4] = positionCur;
    positionCur.Y += y_modifire;
    _set_cur_to_pos(hConsole, positionCur);
    if (_temp_info->cost > 0) {
        printf(" %f ", _temp_info->cost);
    }
    fild_cords[5] = positionCur;
    positionCur.X = _center_x - width / 2 + 4;
    positionCur.Y = _center_y + height / 2 - 2;
    SetConsoleOutputCP(65001); //-------
    _set_cur_to_pos(hConsole, positionCur);
    if (_men_position[1] == table->_col_count) {
        printf("\x1b[43mСохранить\x1b[0m");
    }
    else printf("Сохранить");
    positionCur.X = _center_x + (width / 2 - u8_strlen("Отмена") - 3);
    _set_cur_to_pos(hConsole, positionCur);
    if (_men_position[1] == table->_col_count + 1) {
```

```
            printf("\x1b[43mОтмена\x1b[0m");
        }
        else printf("Отмена");

        c = getch();

        if (c == KEY_ENTER)
        {
            if (_men_position[1] == table->_col_count + 1)
            {
                if (_confirm_window("Отменить операцию ?"))
                {
                    return _output_info;
                }
                else
                {
                    flag_clear = 1;
                }
            }else if (_men_position[1] == table->_col_count)
            {
                if (_confirm_window("Сохранить данные ?"))
                {
                    return _temp_info;
                }
                else
                {
                    flag_clear = 1;
                }
            }
            else
            {
                switch (_men_position[1])
                {
                case 1: {
                    _set_cur_to_pos(hConsole, fild_cords[0]);
                    for (int i = 0; i < width / 2 - padding - 8; i++) {
printf("_"); }
                    _set_cur_to_pos(hConsole, fild_cords[0]);
                    input_string(_temp_info->fio.surname, 40, PERSONAL);
                    input_string(_temp_info->fio.name, 40, PERSONAL);
                    input_string(_temp_info->fio.secondname, 40, PERSONAL);
                    SetConsoleOutputCP(65001); //-------
                    break;
                }
                case 2: {
                    _set_cur_to_pos(hConsole, fild_cords[1]);
                    for (int i = 0; i < width / 2 - padding - 8; i++) {
printf("_"); }
                    _set_cur_to_pos(hConsole, fild_cords[1]);
                    input_string(_temp_info->autor.surname, 40, PERSONAL);
                    input_string(_temp_info->autor.inicial, 3, INICIAL);
                    SetConsoleOutputCP(65001); //-------
                    break;
                }
                case 3: {
                    _set_cur_to_pos(hConsole, fild_cords[2]);
                    for (int i = 0; i < width / 2 - padding - 8; i++) {
printf("_"); }
                    _set_cur_to_pos(hConsole, fild_cords[2]);
                    input_string(_temp_info->book_name, width / 2 - padding - 8,
NORMAL);
                    SetConsoleOutputCP(65001); //-------
                    break;
                }
```

```
                    case 4: {
                        _set_cur_to_pos(hConsole, fild_cords[3]);
                        for (int i = 0; i < width / 2 - padding - 8; i++) {
printf("_"); }
                        _set_cur_to_pos(hConsole, fild_cords[3]);
                        input_string(_temp_info->izd, width / 2 - padding - 8,
NORMAL);
                        SetConsoleOutputCP(65001); //-------
                        break;
                    }
                    case 5: {
                        _set_cur_to_pos(hConsole, fild_cords[4]);
                        for (int i = 0; i < width / 2 - padding - 8; i++) {
printf("_"); }
                        _set_cur_to_pos(hConsole, fild_cords[4]);
                        in_date(&_temp_info->date_out.d, &_temp_info->date_out.m,
&_temp_info->date_out.y);
                        SetConsoleOutputCP(65001); //-------
                        break;
                    }
                    case 6: {
                        _set_cur_to_pos(hConsole, fild_cords[5]);
                        for (int i = 0; i < width / 2 - padding - 8; i++) {
printf("_"); }
                        _set_cur_to_pos(hConsole, fild_cords[5]);
                        input_float(&_temp_info->cost);
                        //scanf("%f", &_temp_info->cost);
                        SetConsoleOutputCP(65001); //-------
                        break;
                    }
                    default:
                        break;
                    }
                }
            }
            if (c == KEY_ESC)
            {
                return _output_info;
                break;
            }

            _men_position = _get_curent_selection(c, _men_position, table->_col_count
+ 1, 1, 1);
        }
        else {
          //
            char mes[] = "\x1b[45mESC\x1b[0m:Выход  \x1b[45mENTER\x1b[0m:Ввод
\x1b[45mСТРЕЛКИ\x1b[0m: Переключение поля"; //51
            positionCur.Y = _center_y + height / 2 - 2;
            positionCur.X = _center_x - 50/2;
            _set_cur_to_pos(hConsole, positionCur);
            printf("%s", mes);
            int cur_step = 1; int cur_key = 0; int step_compl = 0;
            CONSOLE_SCREEN_BUFFER_INFO con_inf;
            CONSOLE_CURSOR_INFO structCursorInfo;
            GetConsoleCursorInfo(hConsole, &structCursorInfo);
            structCursorInfo.bVisible = TRUE;
            SetConsoleCursorInfo(hConsole, &structCursorInfo);
            positionCur.X = _center_x - width / 2 + 4 + max_lenght + 6;
            positionCur.Y = _center_y - height / 2 + 4;
            _set_cur_to_pos(hConsole, positionCur);
            COORD lastcord = {0,0};
            SetConsoleOutputCP(1251); //-------
            while (cur_step <= 9) {
```

```
                _get_con_info(&con_inf);
                lastcord = con_inf.dwCursorPosition;
                switch (cur_step)
                {
                case 1: {
                    cur_key = input_string(_temp_info->fio.surname, 40, PERSONAL);
                    switch (cur_key)
                    {
                    case KEY_ENTER:
                        cur_step++;
                        step_compl++;

                        break;
                    case KEY_ARROW_UP:
                        _set_cur_to_pos(hConsole, positionCur);
                        for (int j = 0; j < (width / 2 - padding - 8); j++) {
                            printf("_");
                        }
                        _set_cur_to_pos(hConsole, positionCur);
                        break;
                    case KEY_ARROW_DOWN:
                        if (cur_step < step_compl) {
                            cur_step++;
                            printf(" ");
                        }
                        else {
                            _set_cur_to_pos(hConsole, lastcord);
                        }
                        break;
                    case KEY_ESC:
                        if (_confirm_window("Вы действительно хотите отменить
ввод?")) {
                            SetConsoleOutputCP(65001); //-------
                            return NULL;
                        }
                        else { SetConsoleOutputCP(65001); goto start; }
                        break;
                    default:
                        break;
                    }
                  break;
                }
                case 2: {
                    cur_key = input_string(_temp_info->fio.name, 40, PERSONAL);
                    switch (cur_key)
                    {
                    case KEY_ENTER:
                        cur_step++;
                        step_compl++;
                        break;
                    case KEY_ARROW_UP:
                        if (cur_step > 1) {
                            cur_step=1;
                        }
                        _set_cur_to_pos(hConsole, positionCur);
                        for (int j = 0; j < (width / 2 - padding - 8); j++) {
                            printf("_");
                        }
                        _set_cur_to_pos(hConsole, positionCur);
                        break;
                    case KEY_ARROW_DOWN:
                        if (cur_step < step_compl) {
                            cur_step++;
                            printf(" ");
```

```
                        }
                        else {
                            _set_cur_to_pos(hConsole, lastcord);
                        }
                        break;
                    case KEY_ESC:
                        if (_confirm_window("Вы действительно хотите отменить
ввод?")) {
                            SetConsoleOutputCP(65001); //-------
                            return NULL;
                        }
                        else { SetConsoleOutputCP(65001); goto start; }
                        break;
                    default:
                        break;
                    }
                    break;
                }
                case 3: {
                    cur_key = input_string(_temp_info->fio.secondname, 40, PERSONAL);
                    switch (cur_key)
                    {
                    case KEY_ENTER:
                        cur_step++;
                        step_compl++;
                        positionCur.Y += y_modifire;
                        _set_cur_to_pos(hConsole, positionCur);
                        break;
                    case KEY_ARROW_UP:
                        if (cur_step > 1) {
                            cur_step = 1;
                        }
                        _set_cur_to_pos(hConsole, positionCur);
                        for (int j = 0; j < (width / 2 - padding - 8); j++) {
                            printf("_");
                        }
                        _set_cur_to_pos(hConsole, positionCur);
                        break;
                    case KEY_ARROW_DOWN:
                        if (cur_step < step_compl) {
                            cur_step++;
                            printf(" ");
                            positionCur.Y += y_modifire;
                            _set_cur_to_pos(hConsole, positionCur);
                        }
                        else {
                            _set_cur_to_pos(hConsole, lastcord);
                        }
                        break;
                    case KEY_ESC:
                        if (_confirm_window("Вы действительно хотите отменить
ввод?")) {
                            SetConsoleOutputCP(65001); //-------
                            return NULL;
                        }
                        else { SetConsoleOutputCP(65001); goto start; }
                        break;
                    default:
                        break;
                    }

                    break;
                }
                case 4: {
```

```
                    cur_key = input_string(_temp_info->autor.surname, 40, PERSONAL);
                    switch (cur_key)
                    {
                    case KEY_ENTER:
                        cur_step++;
                        step_compl++;
                        break;
                    case KEY_ARROW_UP:
                        if (cur_step > 1) {
                            cur_step=1;
                            positionCur.Y -= y_modifire;
                            _set_cur_to_pos(hConsole, positionCur);
                            for (int j = 0; j < (width / 2 - padding - 8); j++) {
                                printf("_");
                            }
                            _set_cur_to_pos(hConsole, positionCur);
                        }
                        break;
                    case KEY_ARROW_DOWN:
                        if (cur_step < step_compl) {
                            printf(" ");
                            cur_step++;
                        }
                        else {
                            _set_cur_to_pos(hConsole, lastcord);
                        }
                        break;
                    case KEY_ESC:
                        if (_confirm_window("Вы действительно хотите отменить
ввод?")) {
                            SetConsoleOutputCP(65001); //-------
                            return NULL;
                        }
                        else { SetConsoleOutputCP(65001); goto start; }
                        break;
                    default:
                        break;
                    }
                   break;
                }
                case 5: {
                    cur_key = input_string(_temp_info->autor.inicial, 3, INICIAL);
                    switch (cur_key)
                    {
                    case KEY_ENTER:
                        cur_step++;
                        step_compl++;
                        positionCur.Y += y_modifire;
                        _set_cur_to_pos(hConsole, positionCur);
                        break;
                    case KEY_ARROW_UP:
                        if (cur_step > 1) {
                            cur_step=4;
                            _set_cur_to_pos(hConsole, positionCur);
                            for (int j = 0; j < (width / 2 - padding - 8); j++) {
                                printf("_");
                            }
                            _set_cur_to_pos(hConsole, positionCur);
                        }
                        break;
                    case KEY_ARROW_DOWN:
                        if (cur_step < step_compl) {
                            cur_step++; printf(" ");
                            positionCur.Y += y_modifire;
```

```
                        _set_cur_to_pos(hConsole, positionCur);
                    }
                    else {
                        _set_cur_to_pos(hConsole, lastcord);
                    }
                    break;
                case KEY_ESC:
                    if (_confirm_window("Вы действительно хотите отменить
ввод?")) {
                        SetConsoleOutputCP(65001); //-------
                        return NULL;
                    }
                    else { SetConsoleOutputCP(65001); goto start; }
                    break;
                default:
                    break;
                }
                break;
            }
            case 6: {
                cur_key = input_string(_temp_info->book_name, 100, NORMAL);
                switch (cur_key)
                {
                case KEY_ENTER:
                    cur_step++;
                    step_compl++;
                    _get_con_info(&con_inf);
                    positionCur.Y += y_modifire;
                    _set_cur_to_pos(hConsole, positionCur);
                    break;
                case KEY_ARROW_UP:
                    if (cur_step > 1) {
                        cur_step=4;
                        positionCur.Y -= y_modifire;
                        _set_cur_to_pos(hConsole, positionCur);
                        for (int j = 0; j < (width / 2 - padding - 8); j++) {
                            printf("_");
                        }
                        _set_cur_to_pos(hConsole, positionCur);
                    }
                    break;
                case KEY_ARROW_DOWN:
                    if (cur_step < step_compl) {
                        cur_step++;
                        positionCur.Y += y_modifire;
                        _set_cur_to_pos(hConsole, positionCur);
                    }
                    else {
                        _set_cur_to_pos(hConsole, lastcord);
                    }
                    break;
                case KEY_ESC:
                    if (_confirm_window("Вы действительно хотите отменить
ввод?")) {
                        SetConsoleOutputCP(65001); //-------
                        return NULL;
                    }
                    else { SetConsoleOutputCP(65001); goto start; }
                    break;
                default:
                    break;
                }
                break;
            }
```

```
                case 7: {
                    cur_key = input_string(_temp_info->izd, 60, NORMAL);
                    switch (cur_key)
                    {
                    case KEY_ENTER:
                        cur_step++;
                        step_compl++;
                        _get_con_info(&con_inf);
                        positionCur.Y += y_modifire;
                        _set_cur_to_pos(hConsole, positionCur);
                        break;
                    case KEY_ARROW_UP:
                        if (cur_step > 1) {
                            cur_step--;
                            positionCur.Y -= y_modifire;
                            _set_cur_to_pos(hConsole, positionCur);
                        }
                        break;
                    case KEY_ARROW_DOWN:
                        if (cur_step < step_compl) {
                            cur_step++;
                            positionCur.Y += y_modifire;
                            _set_cur_to_pos(hConsole, positionCur);
                        }
                        else {
                            _set_cur_to_pos(hConsole, lastcord);
                        }
                        break;
                    case KEY_ESC:
                        if (_confirm_window("Вы действительно хотите отменить
ввод?")) {
                            SetConsoleOutputCP(65001); //-------
                            return NULL;
                        }
                        else { SetConsoleOutputCP(65001); goto start; }
                        break;
                    default:
                        break;
                    }
                    break;
                }
                case 8: {
                    cur_key = in_date(&_temp_info->date_out.d, &_temp_info-
>date_out.m, &_temp_info->date_out.y);
                    switch (cur_key)
                    {
                    case KEY_ENTER:
                        cur_step++;
                        step_compl++;
                        _get_con_info(&con_inf);
                        positionCur.Y += y_modifire;
                        _set_cur_to_pos(hConsole, positionCur);
                        break;
                    case KEY_ARROW_UP:
                        if (cur_step > 1) {
                            cur_step--;
                            positionCur.Y -= y_modifire;
                            _set_cur_to_pos(hConsole, positionCur);
                        }
                        break;
                    case KEY_ARROW_DOWN:
                        if (cur_step < step_compl) {
                            cur_step++;
                            positionCur.Y += y_modifire;
```

```
                        _set_cur_to_pos(hConsole, positionCur);
                    }
                    else {
                        _set_cur_to_pos(hConsole, lastcord);
                    }
                    break;
                case KEY_ESC:
                    if (_confirm_window("Вы действительно хотите отменить
ввод?")) {
                        SetConsoleOutputCP(65001); //-------
                        return NULL;
                    }
                    else { SetConsoleOutputCP(65001); goto start; }
                    break;
                default:
                    break;
                }
                break;
            }
            case 9: {
                cur_step++;
                step_compl++;
                input_float(&_temp_info->cost);
                break;
            }
            default:
                break;
            }
        }
        SetConsoleOutputCP(65001);
        _message_window("Запись успешно добавлена");
        Sleep(2000);
        structCursorInfo.bVisible = FALSE;
        SetConsoleCursorInfo(hConsole, &structCursorInfo);
        return _temp_info;
    }
    }
}



int print_help(char * help_message) {
    int* _size_now = NULL; //текущий размер окна
    _size_now = _get_window_size(_size_now);
    CONSOLE_SCREEN_BUFFER_INFO info_x;  HANDLE hConsole =
GetStdHandle(STD_OUTPUT_HANDLE);
    _get_con_info(&info_x);
    COORD positionCur = { _otstup,_size_now[1] - _interval - 2 };
    _set_cur_to_pos(hConsole, positionCur);
    printf("├");
    for (int i = 0; i < _size_now[0] - _otstup*2 - 1; i++) {
        printf("─");
    }
    printf("┤");
    positionCur.Y++; positionCur.X++;
    _set_cur_to_pos(hConsole, positionCur);
    for (int i = 0; i < _size_now[0] - _otstup * 2 - 1; i++)
        printf(" ");
    _set_cur_to_pos(hConsole, positionCur);
    printf("%s", help_message);
    return 0;
}
```

```
abonent_t* _sort_output(abonent_t* _output_mass, int* filds_count, sort_struct*
sorts)
{
    if (!sorts) return _output_mass;
    switch (sorts->sort_f)
    {
    case DEF:
        return _output_mass;
        break;
    case FIO:
                if (sorts->sort_t == UP){
                    for (int write = 0; write < *filds_count; write++) {
                        for (int sort = 0; sort < *filds_count - 1; sort++) {
                            if (_output_mass[sort].id > _output_mass[sort + 1].id) {
                                abonent_t temp = _output_mass[sort + 1];
                                _output_mass[sort + 1] = _output_mass[sort];
                                _output_mass[sort] = temp;
                            }
                        }
                    }
                  }
                 else {
                    for (int write = 0; write < *filds_count; write++) {
                        for (int sort = 0; sort < *filds_count - 1; sort++) {
                            if (_output_mass[sort].id < _output_mass[sort + 1].id) {
                                abonent_t temp = _output_mass[sort + 1];
                                _output_mass[sort + 1] = _output_mass[sort];
                                _output_mass[sort] = temp;
                            }
                        }
                    }
                }
        break;
    case AUTHOR:
        if (sorts->sort_t == UP) {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (strcmp(_output_mass[sort].autor.surname, _output_mass[sort +
1].autor.surname) == 1) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        else {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (strcmp(_output_mass[sort].autor.surname, _output_mass[sort +
1].autor.surname) == -1) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        break;
    case BOOK_NAME:
        if (sorts->sort_t == UP) {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
```

```
                    if (strcmp(_output_mass[sort].book_name, _output_mass[sort +
1].book_name) == 1) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        else {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (strcmp(_output_mass[sort].book_name, _output_mass[sort +
1].book_name) == -1) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        break;
    case DATE_OUT:
        if (sorts->sort_t == UP) {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (_output_mass[sort].date_out.y > _output_mass[sort +
1].date_out.y) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                    else if (_output_mass[sort].date_out.m > _output_mass[sort +
1].date_out.m && _output_mass[sort].date_out.y == _output_mass[sort + 1].date_out.y)
                    {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                    else if (_output_mass[sort].date_out.m == _output_mass[sort +
1].date_out.m && _output_mass[sort].date_out.y == _output_mass[sort + 1].date_out.y
&& _output_mass[sort].date_out.d > _output_mass[sort + 1].date_out.d) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        else {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (_output_mass[sort].date_out.y < _output_mass[sort +
1].date_out.y) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                    else if (_output_mass[sort].date_out.m < _output_mass[sort +
1].date_out.m && _output_mass[sort].date_out.y == _output_mass[sort + 1].date_out.y)
                    {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
```

```
                    }
                    else if (_output_mass[sort].date_out.m == _output_mass[sort +
1].date_out.m && _output_mass[sort].date_out.y == _output_mass[sort + 1].date_out.y
&& _output_mass[sort].date_out.d < _output_mass[sort + 1].date_out.d) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        break;

    case IZD:
        if (sorts->sort_t == UP) {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (strcmp(_output_mass[sort].izd, _output_mass[sort + 1].izd) ==
1) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        else {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (strcmp(_output_mass[sort].izd, _output_mass[sort + 1].izd) ==
-1) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        break;
    case COST:
        if (sorts->sort_t == UP) {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (_output_mass[sort].cost > _output_mass[sort + 1].cost) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        else {
            for (int write = 0; write < *filds_count; write++) {
                for (int sort = 0; sort < *filds_count - 1; sort++) {
                    if (_output_mass[sort].cost < _output_mass[sort + 1].cost) {
                        abonent_t temp = _output_mass[sort + 1];
                        _output_mass[sort + 1] = _output_mass[sort];
                        _output_mass[sort] = temp;
                    }
                }
            }
        }
        break;
    case ZADANIE:
```

```
        if (sorts->sort_t == DEF) return _output_mass;
        break;
    default:
        break;
    }
    return _output_mass;
}



void dolgiWindow(abonent ** root) {
    _big_window("Должники");
    CONSOLE_SCREEN_BUFFER_INFO info_x;  HANDLE hConsole =
GetStdHandle(STD_OUTPUT_HANDLE);
    int* _size_n = NULL;
    _size_n = _get_window_size(_size_n);
    int _window_w = _size_n[0]; int  _window_h = _size_n[1];
    int _padding, _new_padding; int _mn_size_flag = 0; int _size_temp = 0; char
buff[200];
    int height = _window_h / 2; int width = _window_w / 2;
    int _center_x = _window_w / 2; int flag_clear = 0;
    int _center_y = _window_h / 2;
    int refresh_flag = 1;
    time_t  now = time(0);
    struct tm* ltm = localtime(&now);
    int d, y, m;
    y = ltm->tm_year + 1900;
    m = ltm->tm_mon+1;
    d = ltm->tm_mday;
    dolgi_pers_t_obr* dolgniki = (dolgi_pers_t_obr*)calloc(sizeof(dolgi_pers_t_obr),
1);
    dolgniki->count = 0;
    dolgniki = _get_dolgi_info(*root, dolgniki, d, m, y);
    COORD positionCur;
    positionCur.X = _center_x - width / 2 + 4;
    positionCur.Y = _center_y - height / 2 + 4;
    int* _men_position = (int*)calloc(2, sizeof(int));
    _men_position[0] = 1; _men_position[1] = 1;
    _set_cur_to_pos(hConsole, positionCur);
    _tabel_metadata * table = (_tabel_metadata*)calloc(sizeof(_tabel_metadata), 1);
    table->_col_count = 2;
    table->_cols = (_table_col*)calloc(sizeof(_table_col), table->_col_count);
    //-------------------------------------------------------------------------------
------------//
    table->_cols[0].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[0].name, "ФИО абонента-должника");
    table->_cols[0].resizebl = 1;
    table->_cols[0].size = u8_strlen(table->_cols[0].name);
    //-------------------------------------------------------------------------------
------------//
    table->_cols[1].name = (char*)calloc(sizeof(char), 60);
    strcpy(table->_cols[1].name, "Кол-во книг");
    table->_cols[1].resizebl = 0;
    table->_cols[1].size = u8_strlen(table->_cols[1].name);
    //-------------------------------------------------------------------------------
------------//
    COORD PosCur = { _center_x - width / 2 + 4,_center_y - height / 2 + 3 };
//позиция x и y
    int _size_delta = 0;
    for (int i = 0; i < table->_col_count; i++) {
        _size_delta += table->_cols[i].size;
    }
    _size_delta = (width - 2) - _size_delta - ((table->_col_count )* 2) - 10;
    _set_cur_to_pos(hConsole, PosCur);
```

```
    int size_w = 0;
    for (int i = 0; i < table->_col_count; i++) {
        size_w += table->_cols[i].size;
    }



    for (int i = 0; i < size_w+ (table->_col_count)+3 + _size_delta; i++) {
        if (i == 0) {
            printf("┌");
        }
        if (i == width - 1) {
            printf("┐");
        }
        else
            printf("─");
    }
    PosCur.Y++; //positionCur.X++;
    _set_cur_to_pos(hConsole, PosCur);
    printf("│");
    for (int i = 0; i < table->_col_count; i++) {
        _get_con_info(&info_x);
        _padding = info_x.dwCursorPosition.X - 1;
        printf(" %s ", table->_cols[i].name);
        if (table->_cols[i].resizebl) {
            {
                if ((u8_strlen(table->_cols[i].name) + _size_delta) != table-
>_cols[i].size) {
                    table->_cols[i].size += _size_delta;
                }
            }
        }
        if (u8_strlen(table->_cols[i].name) != table->_cols[i].size) {
            int _sim_padding = table->_cols[i].size - u8_strlen(table-
>_cols[i].name);
            for (int i = 0; i < _sim_padding; i++) {
                printf(" ");
            }
        }

        printf("│");
        _get_con_info(&info_x);
        PosCur.X = info_x.dwCursorPosition.X - 1;
        _new_padding = info_x.dwCursorPosition.X - 1;
        PosCur.Y -= 1;
        _set_cur_to_pos(hConsole, PosCur);
        if (i == table->_col_count - 1) {
            printf("┐");
        }
        else
            printf("┬");
        _get_con_info(&info_x);
        PosCur.Y += 2; PosCur.X = _padding;
        _set_cur_to_pos(hConsole, PosCur);
        if (_padding == _center_x - width / 2 + 4 )
        {
            printf("├"); PosCur.X++;
        }
        else {
            PosCur.X++;
            _set_cur_to_pos(hConsole, PosCur);
        }
        for (int j = PosCur.X; j < _new_padding; j++) {
            printf("─");
```

```
        }
        if (i == table->_col_count - 1) {
            printf("┤");
        }
        else
            printf("┼");
        PosCur.Y -= 1; PosCur.X = _new_padding + 1;
        _set_cur_to_pos(hConsole, PosCur);
    }
    PosCur.Y += 2;
    PosCur.X = _center_x - width / 2 + 4;
    int page = 1;
    int cor_X, cor_Y; cor_X = PosCur.X; cor_Y = PosCur.Y;
        SetConsoleOutputCP(65001); //-------
        while (1) {
            PosCur.X = cor_X; PosCur.Y = cor_Y;
            if (refresh_flag) {
                if ((dolgniki->count)) {
                    int _col_inpage = height - 10;
                    int _diap[2] = { 0,0 };
                    _diap[0] = ((page)-1) * _col_inpage;
                    _diap[1] = _diap[0] + _col_inpage;
                    for (int j = _diap[0]; j < _diap[1]; j++)
                    {
                        if (dolgniki->count <= j) {
                            break;
                        }
                        _set_cur_to_pos(hConsole, PosCur);
                        char buff[400] = { "" };
                        SetConsoleOutputCP(65001); //-------
                        printf("│");

                        sprintf(buff, "%s %s %s", dolgniki->info_mass[j].fio.surname,
dolgniki->info_mass[j].fio.name, dolgniki->info_mass[j].fio.secondname);
                        if (u8_strlen(buff) > table->_cols[0].size + 2) {
                            sprintf(buff, "%s %c.%c", dolgniki-
>info_mass[j].fio.surname, dolgniki->info_mass[j].fio.name[0], dolgniki-
>info_mass[j].fio.secondname[0]);
                        }
                        SetConsoleOutputCP(1251); //-------
                        printf("%s", buff);
                        if (u8_strlen(buff) < table->_cols[0].size + 2)
                        {
                            for (int l = 0; l < table->_cols[0].size + 2 -
u8_strlen(buff); l++)
                                printf(" ");
                        }
                        SetConsoleOutputCP(65001); //-------
                        printf("│");
                        SetConsoleOutputCP(1251); //-------
                        sprintf(buff, "%d", dolgniki->info_mass[j].count_dolg_books);
                        if (u8_strlen(buff) > table->_cols[1].size + 2) {
                            for (int l = 0; l < table->_cols[1].size - 1; l++) {
                                printf("%c", buff[l]);
                            }
                            printf("...");
                        }
                        else
                            printf("%s", buff);
                        if (u8_strlen(buff) < table->_cols[1].size + 2)
                        {
                            for (int l = 0; l < table->_cols[1].size + 2 -
u8_strlen(buff); l++)
                                printf(" ");
```

```
            }
            SetConsoleOutputCP(65001); //-------
            printf("│");
            SetConsoleOutputCP(1251); //-------
            PosCur.Y++;
        }
        refresh_flag = 0;
    }
    else
    {
        _set_cur_to_pos(hConsole, PosCur);
        printf("│");
        for (int i = 0; i < (width - 21) / 2; i++)
            printf("-");
        SetConsoleOutputCP(65001); //-------
        printf(" Данных нет ");
        SetConsoleOutputCP(65001); //-------
        for (int i = 0; i < (width - 21) / 2 - 1; i++)
            printf("-");
        if ((width - 6) % 2 == 1) printf("-");
        printf("│");
        PosCur.Y++;
        refresh_flag = 0;
    }
    SetConsoleOutputCP(65001); //-------
    _set_cur_to_pos(hConsole, PosCur);
    int _row_num = 0; int _padd_border = table->_cols[_row_num].size + 2;
    for (int i = 0; i < width - 10; i++) {
        if (i == 0) {
            printf("└");
        }
        if (i == width - 11) {
            printf("┘");
        }
        else
            if (i == _padd_border) {
                _row_num++;
                if (_row_num < table->_col_count)
                    _padd_border += table->_cols[_row_num].size + 3;
                printf("┴");
            }
            else
                printf("─");
    }
    if (_center_y + height / 2 - 2 - PosCur.Y > 1) {
        for (; _center_y + height / 2 - 2 - PosCur.Y > 1;) {
            PosCur.Y++; _set_cur_to_pos(hConsole, PosCur);
            for (int i = PosCur.X; i < _center_x + width / 2 - 4; i++) {
                printf(" ");
            }
        }
   }
}

positionCur.X = _center_x - width / 2 + 4;
positionCur.Y = _center_y + height / 2 - 2;
_set_cur_to_pos(hConsole, positionCur);
if (_men_position[0] == 1) {
    printf("\x1b[43mПред.страница\x1b[0m");
}
else printf("Пред.страница");
positionCur.X = 3+  width / 2 + 4 + u8_strlen("Пред.страница");
_set_cur_to_pos(hConsole, positionCur);
if (_men_position[0] == 2) {
```

```
            printf("\x1b[43mСлед.страница\x1b[0m");
        }
        else printf("След.страница");
        positionCur.X = _center_x + (width / 2 - u8_strlen("Выход") - 3);
        _set_cur_to_pos(hConsole, positionCur);
        if (_men_position[0] == 3) {
            printf("\x1b[43mВыход\x1b[0m");
        }
        else printf("Выход");
        char c = getch();

        if (c == KEY_ENTER)
        {
            if (_men_position[0] == 1)
            {
                if (page > 1) { page--; refresh_flag = 1; }
            }
            if (_men_position[0] == 2)
            {
                if (dolgniki->count>((page)  * (height - 10))) {
                    page++;
                    refresh_flag = 1;
                }
            }
            if (_men_position[0] == 3)
            {
                return;
            }
        }
        if (c == KEY_ESC)

            return;
            _men_position = _get_curent_selection(c, _men_position, 2, 3, 0);

    }
}
```