Министерство образования Республики Беларусь

|  |  |
| --- | --- |
| Учреждение образования  БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ | |
| Факультет компьютерного проектирования  Кафедра инженерной психологии и эргономики  Дисциплина: Основы конструирования программ | |
|  |  |
|  |  |
| **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  к курсовой работе  на тему | |
| **РАЗРАБОТКА ПРОГРАММЫ УЧЕТА КНИГ В БИБЛИОТЕКИ** | |
|  | |
| Выполнил: | |
| Проверил: | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |

Минск 2022

ОГЛАВЛЕНИЕ

[1. ТРЕБОВАНИЯ К ПРОГРАММЕ 3](#_Toc102396424)

[1.1 Индивидуальное задание 3](#_Toc102396425)

[1.2 Исходные данные 3](#_Toc102396426)

[1.3 Функциональные требования 3](#_Toc102396427)

[1.4 Требования к программной реализации 5](#_Toc102396428)

[2.КОНСТРУИРОВАНИЕ ПРОГРАММЫ 7](#_Toc102396429)

[2.1 Разработка модульной структуры программы 7](#_Toc102396430)

[2.2 Выбор способа организации данных 9](#_Toc102396431)

[2.3 Разработка перечня пользовательских функций программы 10](#_Toc102396432)

[3 РАЗРАБОТКА АЛГОРИТМОВ РАБОТЫ ПРОГРАММЫ 15](#_Toc102396433)

[3.1 Алгоритм функции main 15](#_Toc102396434)

[3.2 Алгоритм функции authorization 15](#_Toc102396435)

[3.3 Алгоритм функции administratorMenu 17](#_Toc102396436)

[4 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ 18](#_Toc102396437)

[4.1 Авторизация 18](#_Toc102396438)

[4.2 Модуль администратора 19](#_Toc102396439)

[4.3 Модуль пользователя 20](#_Toc102396440)

[4.4 Исключительные ситуации 22](#_Toc102396441)

[Приложение А 25](#_Toc102396442)

# 1. ТРЕБОВАНИЯ К ПРОГРАММЕ

## 1.1 Индивидуальное задание

Для книг, хранящихся в библиотеке, задаются: регистрационный номер книги, автор, название, год издания, издательство, количество страниц, номер читательского билета (шесть цифр) последнего читателя, отметка о нахождении книги у читателя или в библиотеке в текущий момент. Индивидуальное задание: вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года (год вводится с клавиатуры). Вывести список книг, находящихся в текущий момент у читателей. Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя.

## 1.2 Исходные данные

1. Язык программирования С++.
2. Среда разработки Microsoft Visual Studio.
3. Вид приложения – консольное.
4. Парадигма программирования – процедурная.
5. Способ организации данных – структуры (struct).
6. Способ хранения данных – файлы.
7. Каждая логически завершенная подзадача программы должна быть реализована в виде отдельной функции.
8. Построение программного кода должно соответствовать соглашению о коде «С++ Code Convention».
9. К защите курсовой работы представляются: консольное приложение и пояснительная записка.
10. Текст пояснительной записки оформляется в соответствии со стандартом предприятия СТП 01–2017.

## 1.3 Функциональные требования

**Первым этапом работы** **программы** является авторизация – предоставление прав доступа.

В рамках данного этапа необходимо считать данные из файла с учетными записями пользователей следующего вида:

1. login;
2. saltedHashPassword (результат хеширования пароля с «солью»);
3. salt («соль»);
4. role (данное поле служит для разделения в правах администраторов и пользователей);
5. access (данное поле служит для подтверждения или блокировки администратором учетных записей).

После ввода пользователем своих персональных данных (логина и пароля) и сверки со считанной из файла информацией необходимо предусмотреть возможность входа: − в качестве администратора (в этом случае, например, role = 1); − в качестве пользователя (в этом случае, например, role = 0).

При отсутствии файл с учетными записями пользователей, необходимо программно создать его и записать учетные данные администратора.

Ввод пароля в форме авторизации маскировать с помощью символов «звёздочки» \*.

Регистрация новых пользователей осуществляется двумя способами:

– первый – администратором в режиме работы с учетными записями пользователей добавляет нового пользователя. При этом поле access принимает значение 1;

– второй – самим пользователем путем ввода желаемых логина и пароля, и ожидания подтверждения администратором новой учетной записи. При этом поле access принимает значение 0.

Для хранения пароля использовать метод хеширования с «солью».

**Вторым этапом работы** **программы** является собственно работа с данными, которая становится доступной только после прохождения авторизации.

Для работы с данными должны быть предусмотрены два функциональных модуля: модуль администратора и модуль пользователя.

**Модуль администратора** включает следующие подмодули (с указанием функциональных возможностей):

1. Управление учетными записями пользователей:
   * просмотр всех учетных записей;
   * добавление новой учетной записи;
   * редактирование учетной записи;
   * удаление учетной записи.
2. Работа с данными:
   1. режим редактирования:
      * просмотр всех книг в библиотеки;
      * добавление новой книги;
      * удаление книги;
      * редактирование книги;
   2. режим обработки данных:
      * просмотр всех книг в библиотеки;
      * вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года;
      * вывести данные по книга находящихся на выдачи;
      * поиск книг по автору;
      * поиск книг по году издания;
      * поиск книг по издательству;
      * поиск книг, изданных после заданного года;
      * сортировка по году издания;
      * сортировка по фамилии автора;
      * сортировка по числу страниц.

**Модуль пользователя** включает подмодуль работы с данными со следующими функциональными возможностями:

* + - просмотр всех книг в библиотеки;
    - вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года;
    - вывести данные по книга находящихся на выдачи;
    - поиск книг по автору;
    - поиск книг по году издания;
    - поиск книг по издательству;
    - поиск книг, изданных после заданного года;
    - сортировка по году издания;
    - сортировка по фамилии автора;
    - сортировка по числу страниц.

Для реализации перечисленных модулей/подмодулей необходимо создавать меню с соответствующими пунктами.

В программе необходимо предусмотреть:

1. Обработку исключительных ситуаций:
   * введенные пользователем данные не соответствуют формату поля (например, символы в числовом поле);
   * введенные пользователем данные нелогичны (книги без названия, число страниц отрицательно или 0, отрицательный год издания, год издания больше текущего года);
   * отсутствие файла с базой данных;
   * ничего не найдено по результатам поиска;
   * удаление или редактирование не существующих записей.
2. Возможность возврата назад (навигация);
3. Запрос на выполнение необратимых действий, а именно подтверждение удаления, вида «Вы действительно хотите удалить книгу (пользователя)?»
4. Запрос на выполнение действий по редактированию записей, вида «Вы действительно хотите изменить книгу (пользователя)?»
5. Обратную связь с пользователем, например, вывод сообщения об успешности удаления/редактирования записи.

## 1.4 Требования к программной реализации

1. Все переменные и константы должны иметь осмысленные имена в рамках тематики работы. Переменным рекомендуется присваивать имена, состоящие из букв нижнего регистра; для формирования составного имени используется нижнее подчеркивание (например, number\_of\_students) или «верблюжья нотация» (например, flagExit).
2. Имена функций должны быть осмысленными, начинаться с буквы нижнего регистра, строиться по принципу глагол + существительное (например, addAccount, findStudentBySurname). Если функция выполняет проверку и возвращает результат типа bool, то ее название должно начинаться с глагола is (например, isNumberNumeric, isLoginUnique).
3. Не допускается использование оператора прерывания goto.
4. Код не должен содержать неименованных числовых констант («магических» чисел), неименованных строковых констант (например, имен файлов и др.). Подобного рода информацию следует представлять как глобальные константы. По правилам качественного стиля программирования тексты всех информационных сообщений, выводимых пользователю в ответ на его действия, также оформляются как константы.
5. Код необходимо комментировать (как минимум в части объявления структур, массивов/векторов, прототипов функций, нетривиальной логики).
6. Код не должен дублироваться – для этого существуют функции!
7. Одна функция решает только одну задачу (например, не допускается в одной функции считывать данные из файла и выводить их на консоль – это две разные функции). При этом внутри функции возможен вызов других функций.
8. Выполнение операций чтения/записи в файл должно быть сведено к минимуму (т. е. после однократной выгрузки данных из файла в массив/вектор дальнейшая работа ведется с этим массивом/вектором, а не происходит многократное считывание данных из файла в каждой функции).
9. Следует избегать глубокой вложенности условных и циклических конструкций: вложенность блоков должна быть не более трех.
10. Следует избегать длинных функций: текст функции должен умещаться на один экран (размер текста не должен превышать 25–50 строк).
11. Следует выносить код логически независимых модулей в отдельные .cpp файлы и подключать их с помощью заголовочных .h файлов.

# 2.КОНСТРУИРОВАНИЕ ПРОГРАММЫ

## 2.1 Разработка модульной структуры программы

Модульная структура программы представлена на рисунке 2.1, а описание модулей в таблице 2.1.



Рисунок 2.1 – Модульная структура программы

Таблица 2.1 – Описание модулей программы

|  |  |
| --- | --- |
| Наименование модуля | Описание |
| Чтение данных из файлов | При запуске программы считывает данные файлов, хранящих информацию о пользователях и книгах библиотеки. При завершении работы осуществляет перезапись данных файлов |
| Авторизация и регистрация | Осуществляется авторизация существующих пользователей или регистрация нового |
| Модуль администратора | Содержит функционал необходимый для работы администратора |
| Работа с учетными записями | Содержит функционал для просмотра и редактирования данных пользователей |
| Просмотр всех записей | Даёт возможность просмотра всех записей пользователей |
| Добавление новой записи | Позволяет осуществлять добавление новых пользователей |
| Редактирование записи | Позволяет осуществлять изменение информации о имеющихся пользователях |
| Удаление записи | Удаление имеющегося пользователя |
| Работа с данными библиотечных книг | Содержит функционал для просмотра, редактирования и обработки данных о книгах находящихся в библиотеки |
| Режим редактирования данных | Содержит функционал для просмотра и редактирования данных о книгах находящихся в библиотеки |
| Просмотр всех книг | Даёт возможность просмотра всех библиотечных книг |
| Добавление новой книги | Позволяет осуществлять добавление новых книг |
| Редактирование книги | Позволяет осуществлять изменение информации о имеющихся книгах |
| Удаление книги | Удаление имеющейся книги |
| Режим обработки данных | Содержит функционал для просмотра и обработки данных о книгах находящихся в библиотеки |
| Поиск | Содержит функционал осуществляющий поиск книг по различным параметрам |
| Поиск по автору | Выполняет поиск книг написанных заданным автором |
| Поиск по году издания | Выполняет поиск книг изданных в заданному году |
| Поиск по издательству | Выполняет поиск книг изданных в заданном издательстве |
| Изданных после заданного года | Выполняет поиск книг изданных после заданному году |
| Сортировка | Содержит функционал осуществляющий сортировку книг по различным параметрам |
| Сортировка по автору | Выполняет сортировку книг по их авторам в алфавитном порядке |
| Сортировка по году издания | Выполняет сортировку книг по их году издания в порядке увеличения |
| Сортировка по числу страниц | Выполняет сортировку книг по числу страниц порядке увеличения |
| Сортировка по автору, изданных после заданного года | Выполняет сортировку книг по их авторам в алфавитном порядке и поиск книг изданных после заданного года |
| На выдаче | Выполняет поиск книг которые находятся на выдаче |
| Модуль пользователя | Содержит функционал необходимый для работы пользователя |

## 2.2 Выбор способа организации данных

В качестве способа описания входных данных выбраны структуры (struct). Для описания данных пользователей используется структура типа User имеющая поля:

* std::string login – хранит логин пользователя;
* std::string saltedHashPassword – хранить значение хешированного пароля с «солью»;
* std::string salt – хранить значение «соли» к паролю;
* userRole role – хранить уровень доступа (администратор или обычный пользователь);
* bool access – указывает пользователь активен или заблокирован.

Тип поля role – это пользовательское перечисление типа userRole хранящая два значения:

* SIMPLE\_USER – простой пользователь;
* ADMINISTRATOR – администратор.

Для описания данных библиотечных книг используется структура типа Book имеющая поля:

* unsigned int registrationNumber – хранит регистрационный номер книги;
* std::string author – хранит автора книги;
* std::string name – хранит название книги;
* unsigned int yearPuplication – хранит год публикации книги;
* std::string publishingHouse – хранит издательство книги;
* unsigned int quantityPage – хранит число страниц в книги;
* unsigned int lastReaderTicketNumber – хранит номер читательского билета последнего читателя;
* bookStatus status – хранит статус книги (на выдаче или в библиотеки).

Тип поля status – это пользовательское перечисление типа bookStatus хранящая два значения:

* BOOK\_IN\_LIBRARY – книга в библиотеки;
* BOOK\_ON\_ISSUE – книга на выдаче.

В качестве способа объединения данных используются векторы с локальной областью видимости.

## 2.3 Разработка перечня пользовательских функций программы

**checkFunction**:

* bool checkingCorrectnessInputForSequences(std::string const input, int const maxValue) – проверяет корректность ввода при выборе пунктов из последовательности (например пунктов меню);
* bool checkingCorrectnessInputUnsignetInt(std::string const input) – проверяет корректность ввода без знаковых чисел;
* bool checkingCorrectnessInputString(std::string const input) – проверяет корректность ввода строковых значений;
* bool confirmationAction(std::string const message) – функция служит для подтверждения действий пользователя.

**drawTable**:

* void drawTableRaw(const std::vector<size\_t>& columnSize, const std::vector<std::string>& data, bool drawLowerBorder = false) – функция выводит в консоль одну из строк таблицы;
* void printDataInCell(size\_t const cellSize, std::string const data) – функция выводит значения в одну из ячеек таблицы;
* void repetSimvol(std::string const simvol, size\_t const repetNumber) – функция служит для повторения одинаковых символов заданное число раз;
* std::string cropString(std::string const originalString, size\_t const maxLen) – функция обрезает исходную строку под заданный размер, отсутствующие символы заменяются «…»;
* T mySum(const std::vector<T>& inputVector) – шаблон функции суммирования элементов вектора.

**editingLibrary**:

* void addNewBookByAdministrator(std::vector<Book>& books) – функция выполняет запрос на добавлении новой книги в базу данных;
* void updateBookByAdministrator(std::vector<Book>& books) – функция выполняет запрос на редактирование информации о книге из базы данных;
* void deleteBookByAdministrator(std::vector<Book>& books) – функция выполняет запрос на удаление книги из базы данных;
* bool setBookParameters(const std::vector<Book>& books, Book& newBook, std::string const welcomeMessage) – функция служит для ввода всей информации о книге;
* int unsigned enteryNewBookRegistrationNumber(std::string const welcomeMessage) – функция служит для ввода регистрационного номера книги;
* std::string enteryNewBookAuthor(std::string const welcomeMessage) – функция служит для ввода автора книги;
* std::string enteryNewBookName(std::string const welcomeMessage) – функция служит для ввода названия книги;
* int unsigned enteryNewBookYearPuplication(std::string const welcomeMessage) – функция служит для ввода года издания книги;
* std::string enteryNewBookPublishingHouse(std::string const welcomeMessage) – функция служит для ввода издательства книги;
* int unsigned enteryNewBookQuantityPage(std::string const welcomeMessage) – функция служит для ввода количества страниц в книги;
* int unsigned enteryNewBookLastReader(std::string const welcomeMessage) – функция служит для ввода номера читательского билета последнего читателя книги;
* bookStatus enteryNewBookStatus(std::string const welcomeMessage) – функция служит для ввода статуса книги (в библиотеки или на выдаче);
* int unsigned enteryUnsignetIntMeaning(std::string const welcomeMessage) – ввод без знакового числового значения;
* std::string enteryStringMeaning(std::string const welcomeMessage) – ввод строкового значения;
* bool bookExists(const std::vector<Book> & books, unsigned int const registrationNumber) – проверка на то существует ли книга с таким регистрационным номером в базе данных;
* int findBookIndex(const std::vector<Book>& books, unsigned int const registrationNumber) – вывод индекса под которым хранится книга с заданным регистрационным номером;
* int getYear() – возвращает текущий год(по системной дате).

**editingUsers**:

* void creatFirstUser(std::vector<User>& users) – создает первого пользователя(пользователя по умолчанию);
* void addUsersByAdministrator(std::vector<User>& users) – функция выполняет запрос на добавлении нового пользователя в базу данных администратором;
* void addUsersBySimpleUser(std::vector<User>& users) – функция выполняет запрос на добавлении нового пользователя в базу данных простым пользователем;
* User setUserLoginAndPassword(const std::vector<User>& users, std::string const welcomeMessage) – функция служит для установки логина и пароля для пользователя;
* std::string enteryNewUserLogin(const std::vector<User>& users, std::string const welcomeMessage) – функция служит для ввода логина пользователя;
* std::string enteryNewUserPasswor(std::string const welcomeMessage) – функция служит для ввода пароля пользователя;
* userRole enteryNewUserRole(std::string const welcomeMessage) – функция служит для ввода прав пользователя(пользователь или администратор);
* bool enteryNewUserAccess(std::string const welcomeMessage, userRole const role) – функция служит для ввода статуса пользователя(активен или заблокирован);
* int findUserIndex(const std::vector<User>& users, std::string const login) – вывод индекса под которым хранится пользователь с заданным логином;
* bool loginExists(const std::vector<User>& users, std::string const login) – проверка на существования пользователя с заданным логином;
* void updateUserDataByAdministrator(std::vector<User>& users, std::string const activUserLogin) – функция выполняет запрос на изменение информации о пользователи администратором;
* User setAllUserData(const std::vector<User>& users, std::string const welcomeMessage) – функция служит для ввода всей информации о пользователе;
* bool updateUserData(const std::vector<User>& users, User& updatedUser, std::string const activUserLogin) – функция служит для изменения данных о пользователе в базе данных;
* void deleteUserDataByAdministrator(std::vector<User>& users, std::string const activUserLogin) – функция выполняет запрос на удаление пользователя администратором;
* bool deleteUser (std::vector<User>& const users, size\_t const indexDeletedUser, std::string const activUserLogin) – функция служит для проверки возможности удаления пользователя из базы данных;
* void activationUserByAdministrator(std::vector<User>& users) – функция выполняет запрос на активацию пользователя администратором;
* void blockUserByAdministrator(std::vector<User>& users, std::string const activUserLogin) – функция выполняет запрос на блокировку пользователя администратором;
* void printUsers(const std::vector<User> &users) – вывод данных о всех пользователях.

**forMainFunction**:

* void setRusSettings() – устанавливает русскую локализацию для консоли;
* void downloadData(std::string const libraryFileName, std::vector<Book>& books, std::string const usersFileName, std::vector<User>& users) – функция предназначена для загрузки требуемых файлов в вектора;
* void writeData(std::string const libraryFileName, const std::vector<Book>& books, std::string const usersFileName, const std::vector<User>& users) – функция предназначена для загрузки в файлы данных из соответствующих векторов.

**menu**:

* void inputMenu(std::vector<User>& users, std::vector<Book>& booksInLibrary) – меню входа в систему;
* void simpleUserMenu(std::vector<Book>& booksInLibrary, User const activUser) – главное меню пользователя;
* void administratorMenu(std::vector<User>& users, std::vector<Book>& booksInLibrary, User const activUser) – главное меню администратора;
* void menuEditingUsers(std::vector<User>& users, User const activUser) – меню изменения параметров пользователей;
* void menuEditingLibrary(std::vector<Book>& booksInLibrary) – меню изменения параметров библиотечных книг;
* void menuProcessingLibrary(std::vector<Book>& booksInLibrary) – меню обработки библиотечных книг;
* void menuSortedBooks(std::vector<Book>& booksInLibrary) – меню типов сортировки книг;
* void menuSearcBooks(const std::vector<Book>& booksInLibrary) – меню поиска книг;
* bool isActivatedUser(User const user) – проверка на активацию пользователя;

**procesingLibrary**:

* void searchQueryByAuthor(const std::vector<Book>& books) – запрос на поиск книги по автору;
* void searchQueryByYearPuolication(const std::vector<Book>& books) – запрос на поиск книги по году издания;
* void searchQueryByYearPuolicationMoreThen(const std::vector<Book>& books) – запрос на поиск книги изданной после заданного года;
* void searchQueryByPublishingHouse(const std::vector<Book>& books) – запрос на поиск книги по издательству;
* void searchQueryBooksOnIssue(const std::vector<Book>& books) – – запрос на вывод книг находящихся на выдаче;
* void findBooksByAuthor(const std::vector<Book>& books, std::string const author, std::vector<Book>& outBooks) – функция поиска книги по автору;
* void findBooksByYearPuolication(const std::vector<Book>& books, unsigned int const yearPuolication, std::vector<Book>& outBooks) – функция поиска книги по году издания;
* void findBooksByYearPuolicationMoreThen(const std::vector<Book>& books, unsigned int const minYearPuolication, std::vector<Book>& outBooks) – функция поиска книги изданной после заданного года;
* void findBooksByPublishingHouse(const std::vector<Book>& books, std::string const publishingHouse, std::vector<Book>& outBooks) – функция поиска книги по издательству;
* void findBooksOnIssue(const std::vector<Book>& books, std::vector<Book>& outBooks) – функция поиска книг находящихся на выдаче;
* void sortBooksByYearPublication(std::vector<Book> &books) – сортировка книг по году издания;
* void sortBooksByAuthor(std::vector<Book> &books) – сортировка книг по автору;
* void sortBooksByQuantityPage(std::vector<Book> &books) – сортировка книг по числу страниц;
* bool sortByYearPublication(Book firstBook, Book secondBook) – сортировка по полю года издания;
* bool sortByAuthor(Book firstBook, Book secondBook) – сортировка по полю автора;
* bool sortByQuantityPage(Book firstBook, Book secondBook) – сортировка по полю числа страниц;
* void printBooks(const std::vector<Book>& books) – вывод данных по всем книгам.

**userPassword**:

* bool authorization(const std::vector<User>& users, User& newActivUser) – функция авторизации;
* std::string createSaltForUser(std::string const userName) – функция создания «соли» к паролю для пользователя;
* std::string generateSalt(size\_t const saltLen) – функция генерации «соли»;
* std::string convertUserPasswordToHash(std::string const passwordWithSalt, std::string const userName) – функция перевода пароля с «солью» в хеш;
* std::string readPassword() – функция для чтения пароля с консоли, заменяет символы на «звёздочки» \*;
* std::string hashFunction(std::string const toHash, int const randomSeed) – функция хеширования.

**workingWithLibraryFile**:

* bool fileWithLibraryIsOpen(std::string const fileName) – проверка на наличие файла с базой данных библиотечных книг;
* void downloadLibraryData(std::string const fileName, std::vector<Book>& booksInLibrary) – загрузка данных по библиотечным книга из файла в вектор;
* void readLibraryDataFromFile(std::ifstream& fileWithData, std::vector<Book>& booksInLibrary) – чтение данных из файла базы данных о библиотечных книгах;
* void writeLibraryDataToFile(std::string const fileName, std::vector<Book> const\* booksInLibrary) – запись из вектора в файл данных о библиотечных книгах.

**workingWithUserFile**:

* void downloadUsersData(std::string const fileName, std::vector<User>& users) – загрузка данных по пользователям из файла в вектор;
* void readUsersDataFromFile(std::ifstream& fileWithData, std::vector<User>& users) – чтение данных из файла базы данных о пользователях;
* void writeUserDataToFile(std::string const fileName, std::vector<User> const\* users) – запись из вектора в файл данных о пользователях.

# 3 РАЗРАБОТКА АЛГОРИТМОВ РАБОТЫ ПРОГРАММЫ

## 3.1 Алгоритм функции main

Алгоритм функции main представлен на рисунке 3.1.



Рисунок 3.1 – Алгоритм функции main

## 3.2 Алгоритм функции authorization

Алгоритм функции authorization представлен на рисунке 3.2.



Рисунок 3.1 – Алгоритм функции authorization

## 3.3 Алгоритм функции administratorMenu

Алгоритм функции administratorMenu представлен на рисунке 3.3.



Рисунок 3.3 – Алгоритм функции administratorMenu

# 4 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

## 4.1 Авторизация

При запуске программы пользователя встречает входное меню входа в систему, представлена на рисунке 4.1.

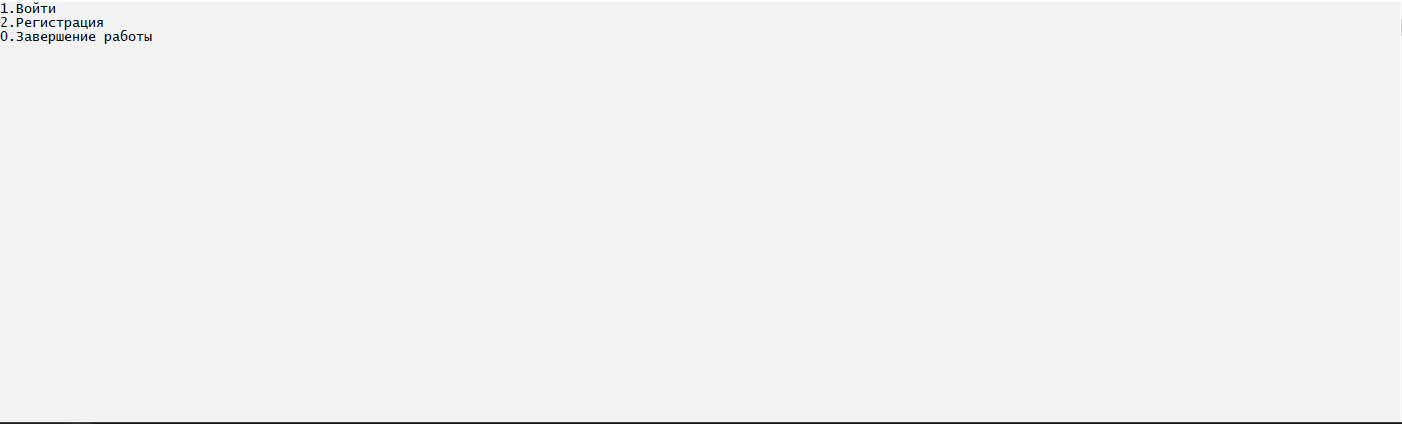


Рисунок 4.1 – Входное меню

При выборе пункта 1 «Войти» у пользователя появится запрос на ввод логина и пароля, рисунок 4.2.

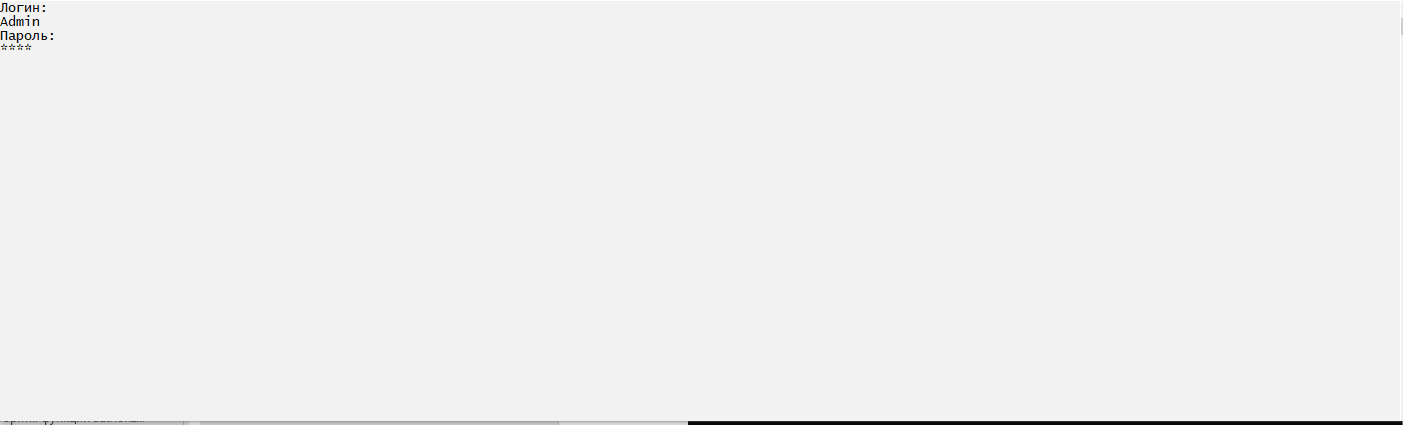


Рисунок 4.2 – Ввод логина и пароля

При успешной авторизации и если пользователь активирован, то ему откроется меню администратора или меню пользователя. Если же пользователь не прошёл авторизацию, то выведется сообщение об ошибке, рисунок 4.3.

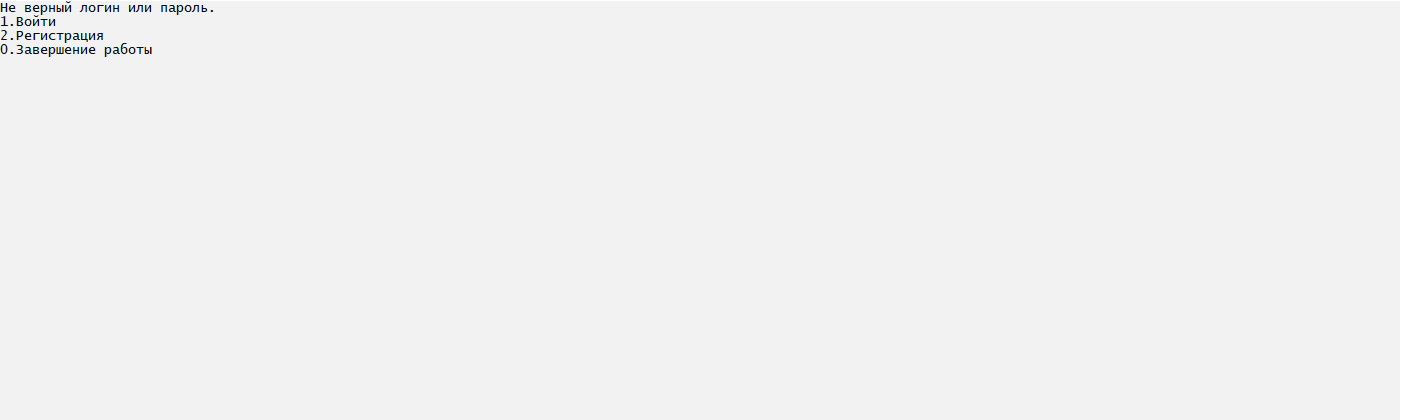


Рисунок 4.3 – Сообщение об ошибке при авторизации

Если же пользователь успешно прошёл авторизацию, но при этом ещё не был активирован администратором, то пользователю будет об этом сообщено, рисунок 4.4.

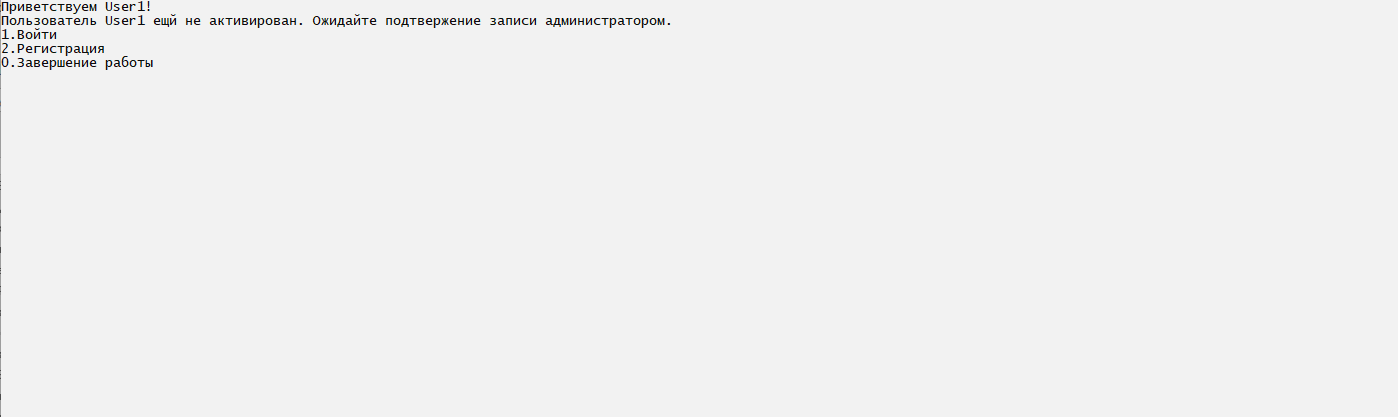


Рисунок 4.4 – Сообщение об отсутствии активации

## 4.2 Модуль администратора

Главное меню администратора имеет следующий вид, рисунок 4.5.

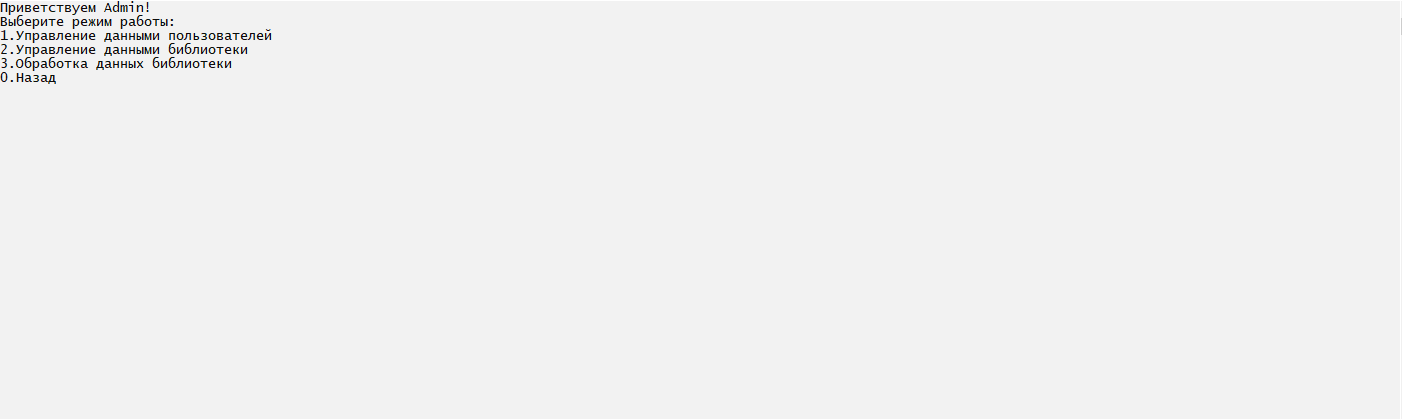


Рисунок 4.5 – Главное меню администратора

При выборе пункта 1 администратор переходит в меню управления данными пользователей, рисунок 4.6.

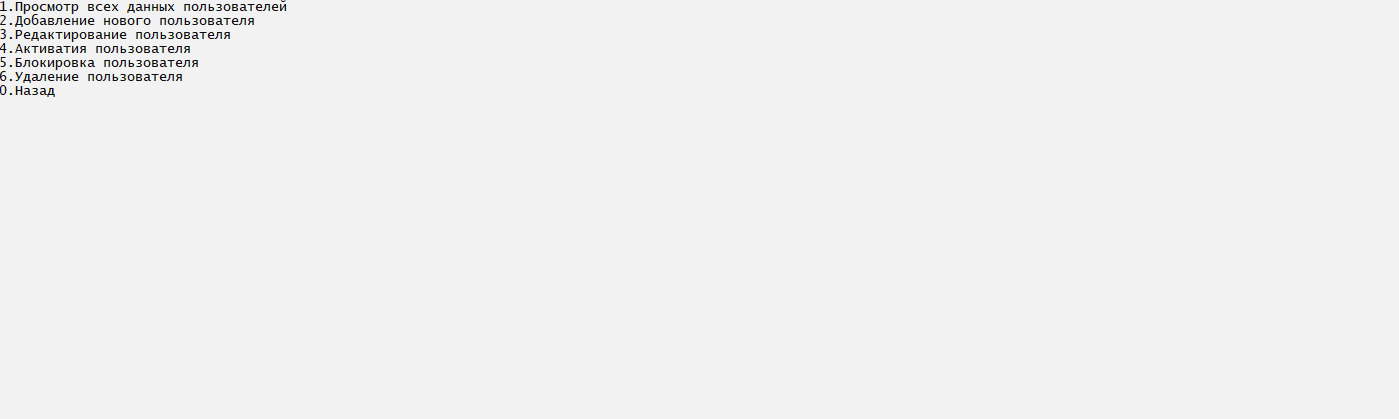


Рисунок 4.6 –Меню управления данными пользователей

При выборе пункта 2 главного меню администратора, администратор переходит в меню управления данными библиотечных книг, рисунок 4.7.

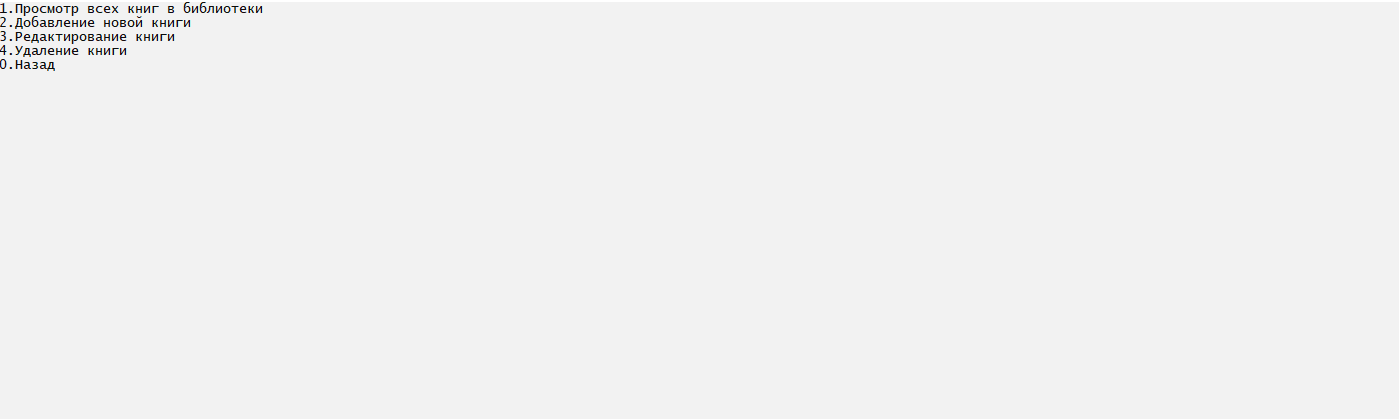


Рисунок 4.7 –Меню управления данными библиотечных книг

## 4.3 Модуль пользователя

Главное меню пользователя имеет следующий вид, рисунок 4.8.

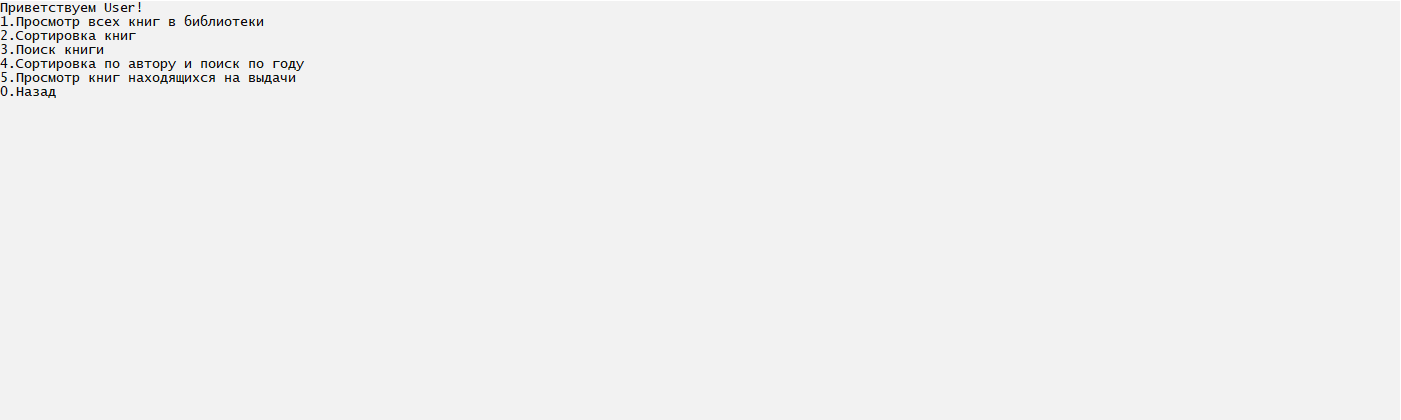


Рисунок 4.8 – Главное меню пользователя

При выборе 1 пункта меню пользователь увидит все имеющиеся в библиотеки книги, результат на рисунке 4.9.

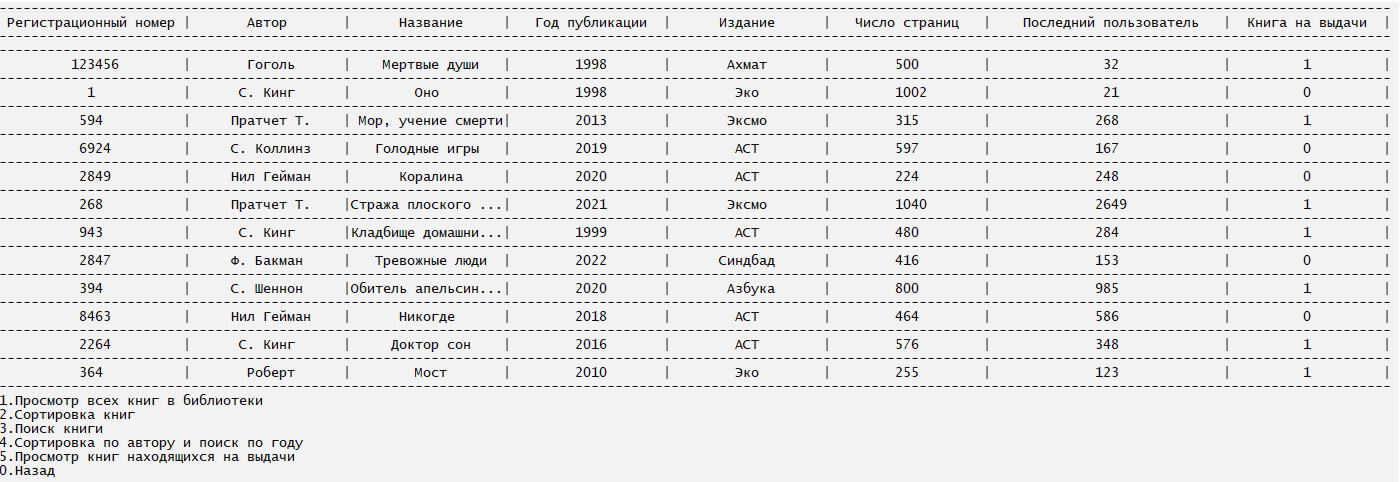


Рисунок 4.9 – Просмотр всех книг в библиотеки

При выборе 2 пункта меню пользователь перейдет в меню возможных вариантов сортировки книг, рисунок 4.10.

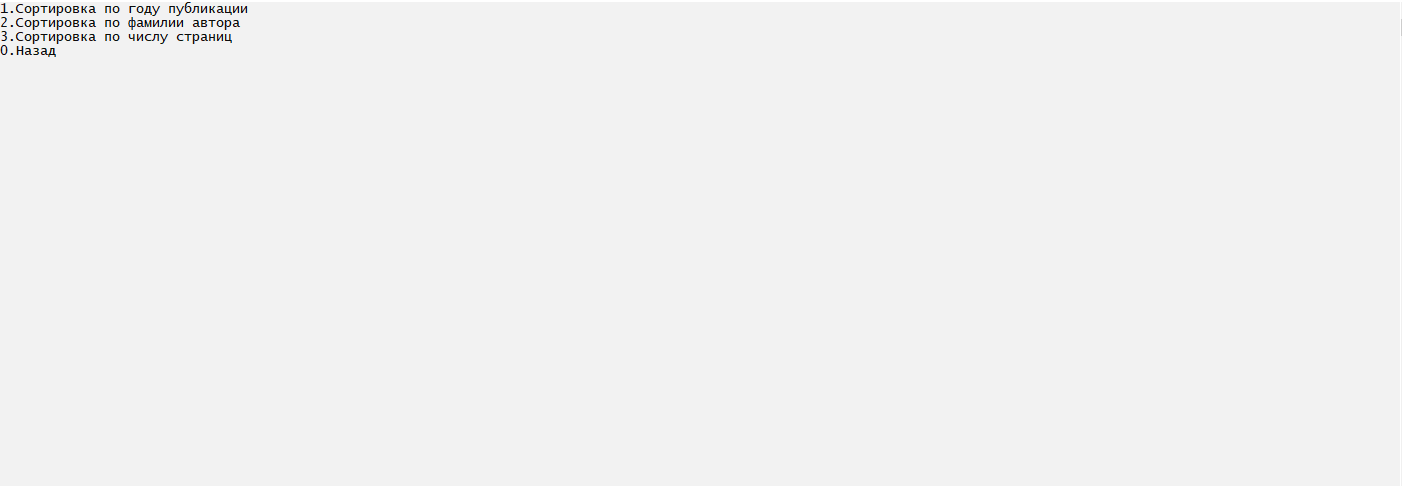


Рисунок 4.10 – Меню возможных сортировок книг

При выборе 3 пункта меню пользователь перейдет в меню возможных вариантов поиска книг, рисунок 4.11.

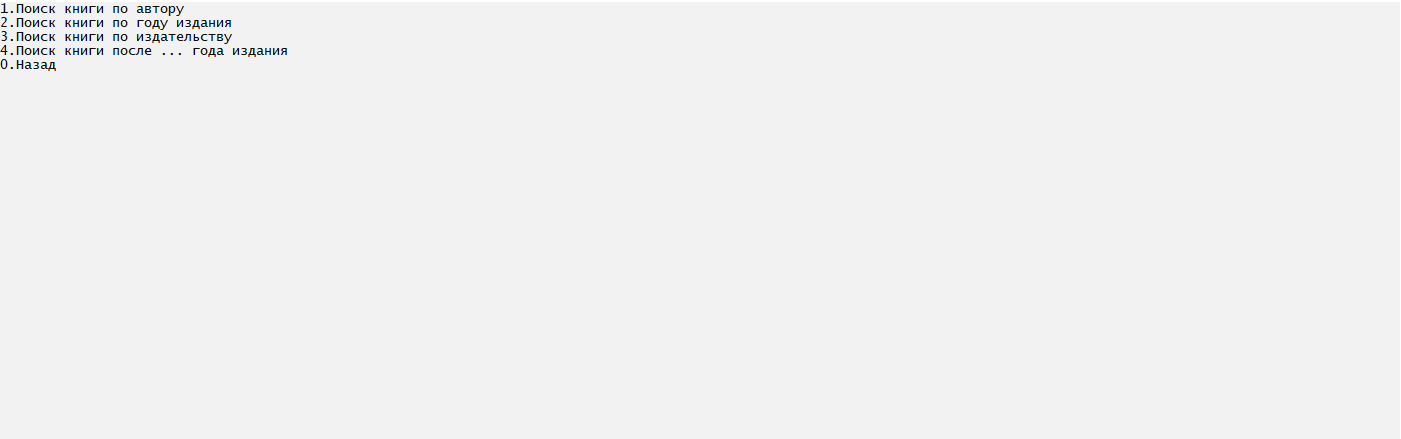


Рисунок 4.11 – Меню возможных вариантов поиска книг

При выборе 4 пункта меню у пользователя запросят год издания, после которого он бы хотел видеть книги, рисунок 4.12.

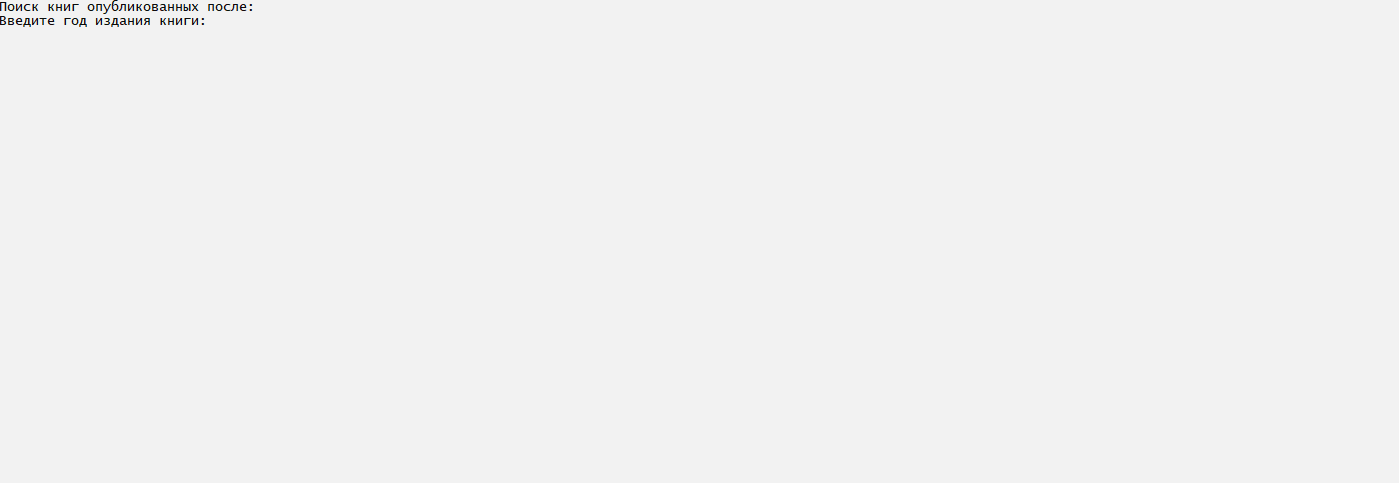


Рисунок 4.12 – Запрос на ввод года издания

После чего пользователь увидит все имеющиеся в библиотеки книги, отсортированные в алфавитном порядке по фамилии автора выпущенных после требуемого года, результат на рисунке 4.13.

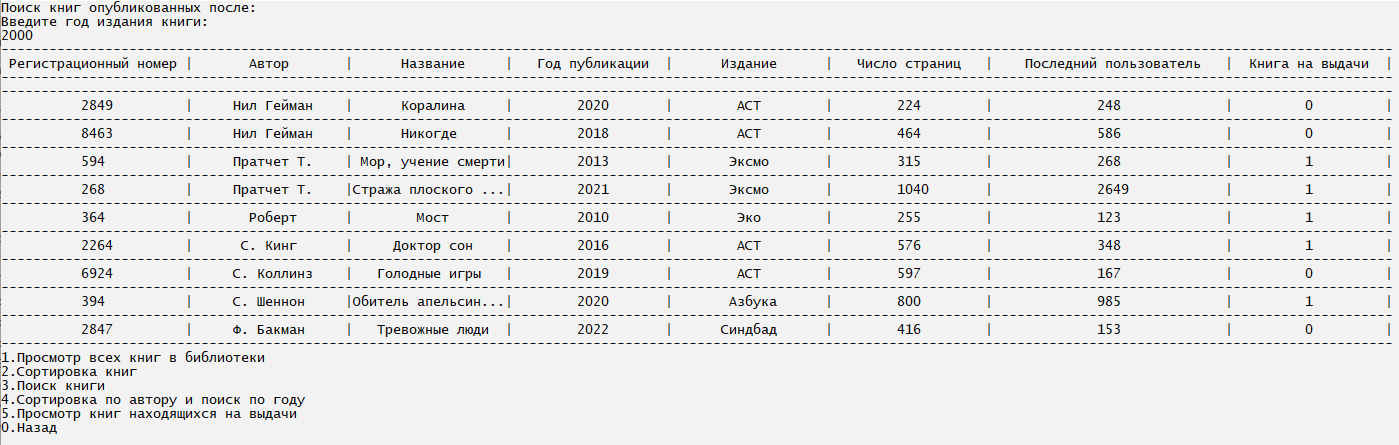


Рисунок 4.13 – Конечный результат запроса 4 пользовательского меню

При выборе пункта 5 пользователь увидит все книги, которые в данный момент находятся на выдаче, рисунок 4.14.

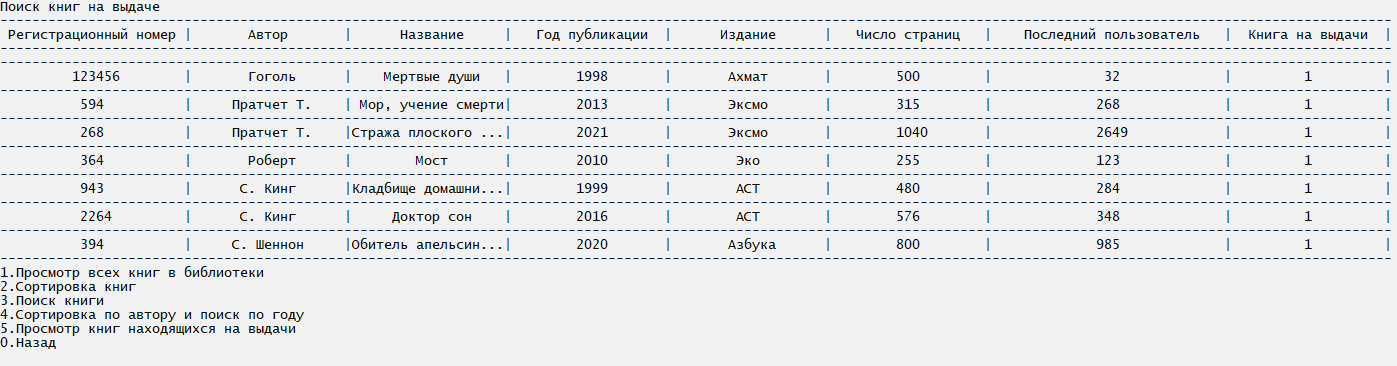


Рисунок 4.14 – Вывод книг, находящихся на выдаче

## 4.4 Исключительные ситуации

Программой предусмотрена обработка исключительных ситуаций, например, таких как ввод не существующих пунктов меню или ввод вместо числового значения строки, в таких случаях пользователь получит сообщение об некорректном вводе, рисунок 4.15.

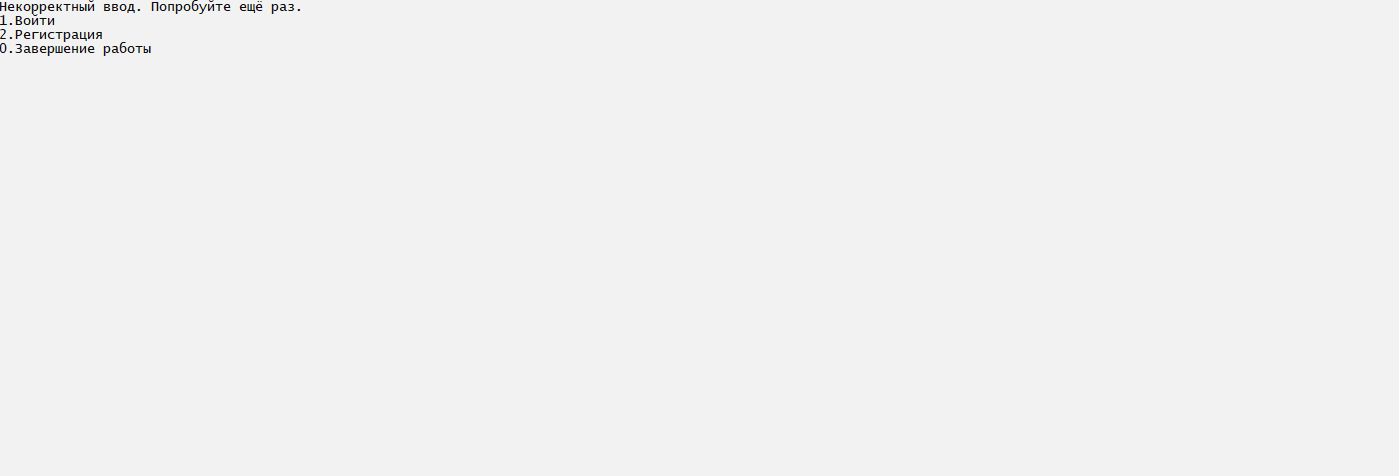


Рисунок 4.15 – Сообщение об некорректном вводе

Кроме того, администратор не может вводить нелогичные данные типа отрицательный год издания или год изданию превышающий текущий, в данном случаи он получит следующие сообщение, рисунок 4.16.

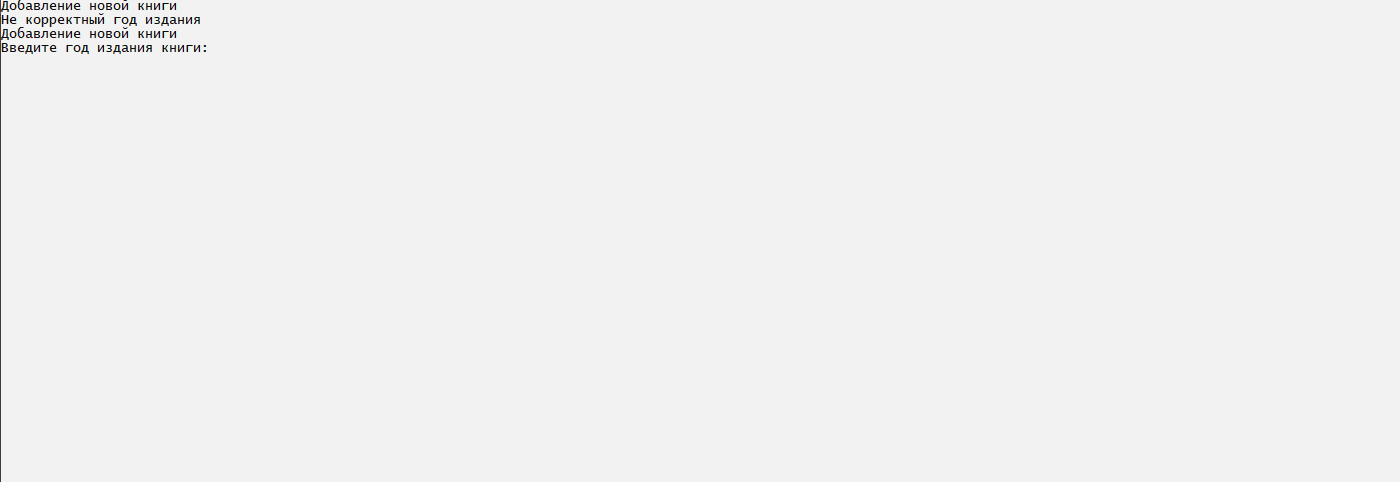


Рисунок 4.16 – Сообщение об некорректном годе издания

Если при запуске программы не будет обнаружен файл с базой данных библиотеки работа программы завершится с сообщением, рисунок 4.17.

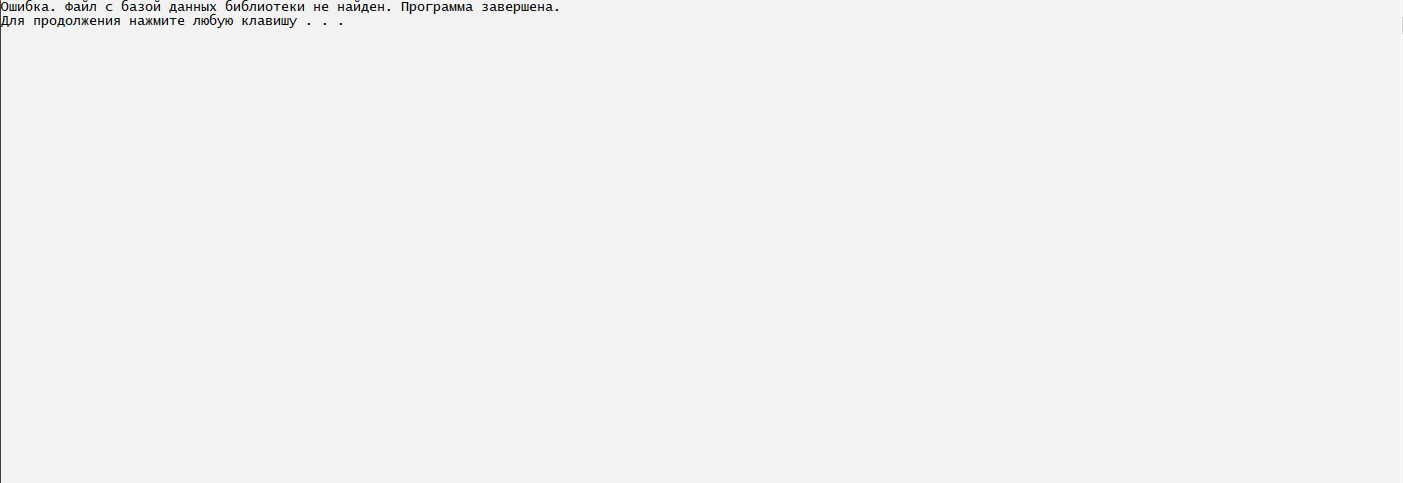


Рисунок 4.17 – Сообщение об отсутствии файла с базой данных библиотеки

В случаи если по результату запроса на поиск книг ничего не будет найдено, пользователь увидит сообщение, рисунок 4.18.

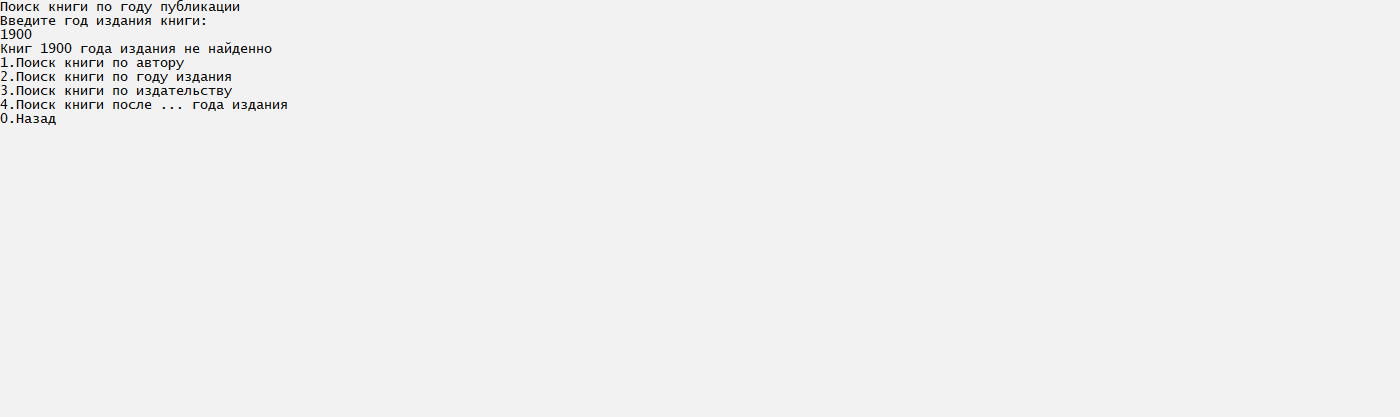


Рисунок 4.18 – Сообщение об отсутствии результатов поиска

Похожие сообщения администратор будет получать при попытке удалить или редактировать пользователей и книги которых не существует, рисунок 4.19.

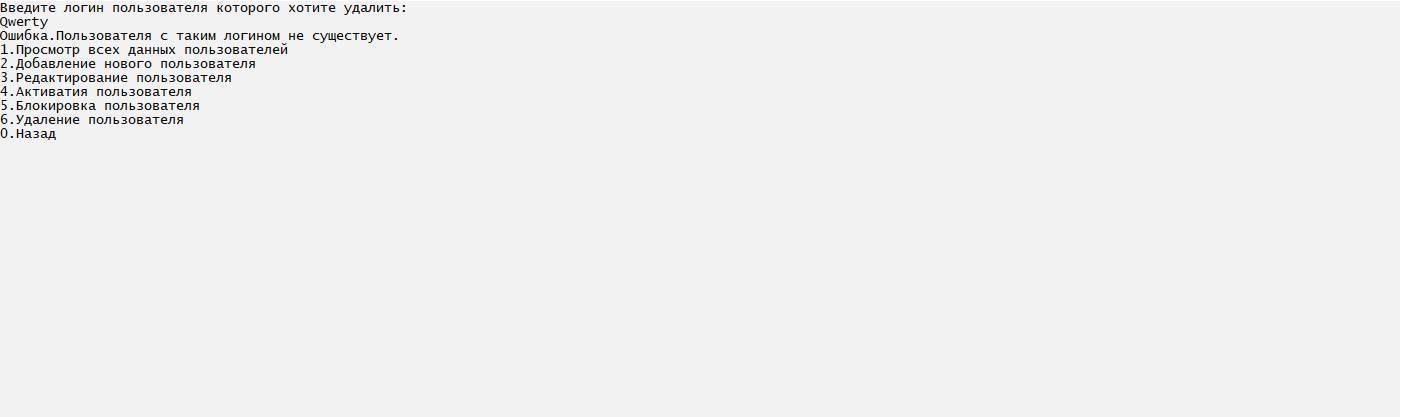


Рисунок 4.18 – Сообщение об отсутствии требуемого пользователя

# Приложение А

**main.cpp**

#include "forMainFunction.h"

#include "menu.h"

int main()

{

setRusSettings();

std::string const FILE\_NAME\_WITH\_LIBRARY\_DB{ "libraryDB.txt" };// константа для хранения имени файла с БД библиотечных книг

std::vector<Book> booksInLibrary{};// объявление вектора для хранения информации библиотечных книг

std::string const FILE\_NAME\_WITH\_USERS{ "users1.txt" };// константа для хранения имени файла с БД пользователей

std::vector<User> users{};// объявление вектора для хранения информации пользователей

if (fileWithLibraryIsOpen(FILE\_NAME\_WITH\_LIBRARY\_DB))

{

downloadData(FILE\_NAME\_WITH\_LIBRARY\_DB, booksInLibrary, FILE\_NAME\_WITH\_USERS, users);

inputMenu(users, booksInLibrary);

writeData(FILE\_NAME\_WITH\_LIBRARY\_DB, booksInLibrary, FILE\_NAME\_WITH\_USERS, users);

}

else

{

std::string const dbError{ "Ошибка. Файл с базой данных библиотеки не найден. Программа завершена." };//константа хранящая сообщение об ошибке подключения к БД

std::cout << dbError << std::endl;

system("pause");//пауза, предотвращает преждевременное закрытие консоли

}

}

**Book.h**

#pragma once

#include <iostream>

//перечисление содержашие возможный статус книи

enum class bookStatus

{

BOOK\_IN\_LIBRARY, //книга в библиотеки

BOOK\_ON\_ISSUE //книга на выдаче

};

//структура хранящая данные о книге

struct Book

{

unsigned int registrationNumber{}; //регистрационный номер

std::string author{}; //автор

std::string name{}; //название книги

unsigned int yearPuplication{}; //год издания

std::string publishingHouse{}; //мздательство

unsigned int quantityPage{}; //число страниц

unsigned int lastReaderTicketNumber{}; //номер читательского билета последнего читателя

bookStatus status{}; //статус книги

};

**User.h**

#pragma once

#include<iostream>

//перечеслинеи определяющие права пользователя

enum class userRole

{

SIMPLE\_USER, //простой пользователь

ADMINISTRATOR //администратор

};

//структура хранящая данные пользователя

struct User

{

std::string login{}; //логин

std::string saltedHashPassword{}; //хэш паспорта с "солью"

std::string salt{}; //"соль"

userRole role{ userRole::SIMPLE\_USER }; //права пользователя

bool access{ false }; //пользователь активирован(true) или блокирован(false)

};

**checkFunction.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

//проверяет корректность ввода при выборе пунктов из последовательности (например пунктов меню)

//на вход подается значение которое надо проверить и максимальное возможное значение

//если ввод был корректен функция вернёт true иначе false

bool checkingCorrectnessInputForSequences(std::string const input, int const maxValue);

//проверяет корректность ввода без знаковых чисел

//на вход подается значение которое надо проверить

//если ввод был корректен функция вернёт true иначе false

bool checkingCorrectnessInputUnsignetInt(std::string const input);

//проверяет корректность ввода строковых значений

//на вход подается значение которое надо проверить

//если ввод был корректен функция вернёт true иначе false

bool checkingCorrectnessInputString(std::string const input);

//функция служит для подтверждения действий пользователя

//на вход подается сообщение об подтверждении какого-либо действия

//если пользователь подтверждает действие функция вернёт true иначе false

bool confirmationAction(std::string const message);

**checkFunction.cpp**

#include "checkFunction.h"

bool checkingCorrectnessInputForSequences(std::string const input, int const maxValue)

{

std::string const trueAnswer{ "0123456789" };//перечень разрешонных для ввода символов

size\_t index{ input.find\_first\_not\_of(trueAnswer) };//поиск индексо первого символа не соответствующего перечню

if (index == std::string::npos)//если индекс пустой

{

int inputToInt{ std::atoi(input.c\_str()) };

if (inputToInt <= maxValue)//проверка, что введённое число не выходи за заданные пределы

return true;

}

return false;

}

bool checkingCorrectnessInputUnsignetInt(std::string const input)

{

std::string const trueAnswer{ "0123456789" };//перечень разрешонных для ввода символов, т.е. что введены только числа

size\_t index{ input.find\_first\_not\_of(trueAnswer) };//поиск индексо первого символа не соответствующего перечню

if (index == std::string::npos)//если индекс пустой

return true;

else

return false;

}

bool checkingCorrectnessInputString(std::string const input)

{

std::string const falseAnswer{ " " }; //искомые символ

size\_t index{ input.find\_first\_not\_of(falseAnswer) }; //поиск индексо первого символа не соответствующего перечню искомых

if (index == std::string::npos) //если индекс пустой, т.е. ввод состоит только из пробелов

return false;

else

return true;

}

bool confirmationAction(std::string const message)

{

system("cls");//очистка консоли

for (; ; )

{

std::string const addToMessage{ "\n0.Отмена\n1.Да" };

std::cout << message + addToMessage << std::endl;

std::string answer{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, answer); //чтение строки с наличием " "

int const maxValue{ 1 };

if (checkingCorrectnessInputForSequences(answer, maxValue)) //если ввод корректен

{

return std::atoi(answer.c\_str()); //возврот 0 или 1

}

else

{

system("cls");//очистка консоли

std::string const errorInput{ "Некорректный ввод. Введите 0(отмена) или 1(ОК)" };

std::cout << errorInput << std::endl;

}

}

}

**drawTable.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <stdlib.h>

//функция выводит в консоль одну из строк таблицы

//на вход подается вектор содержаший размеры каждого столбца, вектор со значениями которые надо поместить в столбцы и

//флаг указывающий на необходимость рисования нижней границы строки

void drawTableRaw(const std::vector<size\_t>& columnSize, const std::vector<std::string>& data, bool drawLowerBorder = false);

//функция выводит значения в одну из ячеек таблицы

//на вход подаётся размер ячейки и значение которое туда надо поместить

void printDataInCell(size\_t const cellSize, std::string const data);

//функция служит для повторения одинаковых символов заданное число раз

//на вход подаётся символ который надо повторить и число раз сколько его надо вывести

void repetSimvol(std::string const simvol, size\_t const repetNumber);

//функция обрезает исходную строку под заданный размер, отсутствующие символы заменяются «…»

//на вход подаётся строка которую надо проверить и максимальная допустимая длина

std::string cropString(std::string const originalString, size\_t const maxLen);

//шаблон функции суммирования элементов вектора

//на вход подается вектор элементы которого надо суммировать

//функция возврощает сумму элементов вектора

template <typename T>

T mySum(const std::vector<T>& inputVector)

{

T outSumm{};

for (auto element : inputVector)

{

outSumm += element;

}

return outSumm;

}

**drawTable.cpp**

#include "drawTable.h"

void drawTableRaw(const std::vector<size\_t>& columnSize, const std::vector<std::string>& data, bool drawLowerBorder)

{

size\_t rawSize{ mySum<size\_t>(columnSize) }; //определения ширины таблицы

repetSimvol("-", rawSize); //вывод верхней границы строки

std::cout << std::endl;

for (size\_t i{}; i < columnSize.size(); ++i)

{

printDataInCell(columnSize[i], data[i]); //ввод ячеек

}

std::cout << std::endl;

if (drawLowerBorder) //если требутся ввести нижную границу

{

repetSimvol("-", rawSize); //вывод нижней границы строки

std::cout << std::endl;

}

}

void printDataInCell(size\_t const cellSize, std::string const data)

{

if (data.size() < cellSize) //если значение помешается в ячейку

{

size\_t leftIndent{ (cellSize - data.size()) / 2 + data.size() }; //определение смешения слева

size\_t rightIndent{ cellSize - leftIndent }; //определение смешения справ

std::cout << std::setw(leftIndent) << data << std::setw(rightIndent) << "|"; //установка смещения слева ввод значеня, дополнительное смещение

}

else

{

size\_t maxNameLen{ cellSize - 1 }; //максимальный размер строки

std::string cropData{ cropString(data, maxNameLen) }; //обрезание строки

std::cout << cropData << "|";//вывод обрезанного значения

}

}

void repetSimvol(std::string const simvol, size\_t const repetNumber)

{

for (size\_t i{}; i < repetNumber; ++i)

{

std::cout << simvol;

}

}

std::string cropString(std::string const originalString, size\_t const maxLen)

{

if (originalString.size() < maxLen) //если исходная строка меньше допустимой длины

return originalString;

else

{

std::string cropSimvol{ "..." }; //символ продолжения

std::string crop{ originalString.begin(), originalString.begin() + maxLen - cropSimvol.size() }; //обрезания строки, от 0 индеса до индекса с номером maxLen-3 (3-размер символа продолжения)

return crop + cropSimvol;

}

}

**editingLibrary.h**

#pragma once

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include "Book.h"

#include "checkFunction.h"

//функция выполняет запрос на добавлении новой книги в базу данных

//принимает на вход ссылку на вектор книг

void addNewBookByAdministrator(std::vector<Book>& books);

//функция выполняет запрос на редактирование информации о книге из базы данных

//принимает на вход ссылку на вектор книг

void updateBookByAdministrator(std::vector<Book>& books);

//функция выполняет запрос на удаление книги из базы данных

//принимает на вход ссылку на вектор книг

void deleteBookByAdministrator(std::vector<Book>& books);

//функция служит для ввода всей информации о книге

//принимает на вход ссылку на вектор книг, ссылку на новую книгу, и

//сообщение которое должно выводится при вводе данных книги

//выходным параметрои является ссылка на новую книгу, которая будет хранить все введённые параметры

//если ввод был корректен функция вернёт true иначе false

bool setBookParameters(const std::vector<Book>& books, Book& newBook, std::string const welcomeMessage);

//функция служит для ввода регистрационного номера книги

//принимает на вход сообщение которое должно выводится при вводе

//возвращает число удовлетворяющие требованием к регистрационному номеру

int unsigned enteryNewBookRegistrationNumber(std::string const welcomeMessage);

//функция служит для ввода автора книги

//принимает на вход сообщение которое должно выводится при вводе

//возвращает строку удовлетворяющие требованием к автору

std::string enteryNewBookAuthor(std::string const welcomeMessage);

//функция служит для ввода названия книги

//принимает на вход сообщение которое должно выводится при вводе

//возвращает строку удовлетворяющие требованием к названию

std::string enteryNewBookName(std::string const welcomeMessage);

//функция служит для ввода года издания книги

//принимает на вход сообщение которое должно выводится при вводе

//возвращает число удовлетворяющие требованием к году издания

int unsigned enteryNewBookYearPuplication(std::string const welcomeMessage);

//функция служит для ввода издательства книги

//принимает на вход сообщение которое должно выводится при вводе

//возвращает строку удовлетворяющие требованием к издательству

std::string enteryNewBookPublishingHouse(std::string const welcomeMessage);

//функция служит для ввода количества страниц в книги

//принимает на вход сообщение которое должно выводится при вводе

//возвращает число удовлетворяющие требованием к количеству страниц

int unsigned enteryNewBookQuantityPage(std::string const welcomeMessage);

//функция служит для ввода номера читательского билета последнего читателя книги

//принимает на вход сообщение которое должно выводится при вводе

//возвращает число удовлетворяющие требованием к номеру читательского билета

int unsigned enteryNewBookLastReader(std::string const welcomeMessage);

//функция служит для ввода статуса книги (в библиотеки или на выдаче)

//принимает на вход сообщение которое должно выводится при вводе

//возвращает одно из значений статуса

bookStatus enteryNewBookStatus(std::string const welcomeMessage);

//ввод без знакового числового значения

//принимает на вход сообщение которое должно выводится при вводе

//возвращает без знаковое число

int unsigned enteryUnsignetIntMeaning(std::string const welcomeMessage);

//ввод строкового значения

//принимает на вход сообщение которое должно выводится при вводе

//возвращает строку

std::string enteryStringMeaning(std::string const welcomeMessage);

//проверка на то существует ли книга с таким регистрационным номером в базе данных

//на вход принимает ссылку на вектор книг и искомый регистрационный номер

//если книга с таким номером найдена функция вернёт true иначе false

bool bookExists(const std::vector<Book> & books, unsigned int const registrationNumber);

//вывод индекса под которым хранится книга с заданным регистрационным номером

//на вход принимает ссылку на вектор книг и искомый регистрационный номер

//если книга с таким номером найдена функция вернёт её индекс иначе -1

int findBookIndex(const std::vector<Book>& books, unsigned int const registrationNumber);

//возвращает текущий год(по системной дате)

int getYear();

**editingLibrary.cpp**

#include "editingLibrary.h"

void addNewBookByAdministrator(std::vector<Book>& books)

{

std::string const welcomeMessage{ "Добавление новой книги" };

Book newBook{};

if (setBookParameters(books, newBook, welcomeMessage)) //если данные книги были введены успешно

{

std::string const question{ "Вы действительно хотите добавить новую книгу?" };

if (confirmationAction(question)) //если подтвердили добавление книги

{

books.push\_back(newBook);

std::string const bookAdded{ "Новая книга успешно добавлена" };

std::cout << bookAdded << std::endl;

}

else

{

std::string const bookAddedCanel{ "Добавление книги отменено" };

std::cout << bookAddedCanel << std::endl;

}

}

else

{

std::string const errorMessage{ "Книга с таким регистрационным номером уже имеется." };

std::cout << errorMessage << std::endl;

}

}

void updateBookByAdministrator(std::vector<Book>& books)

{

system("cls");//очистка консоли

std::string const welcomeMessage{ "Введите регистрационный номер книги которую хотите редактировать:" };

std::cout << welcomeMessage << std::endl;

std::string registrationNumberUpdatedBook{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, registrationNumberUpdatedBook);//чтение строки с наличием " "

int indexUpdatedBook{ findBookIndex(books, std::atoi(registrationNumberUpdatedBook.c\_str())) }; //поиск индекса книги с введенным регистрационным номером

if (indexUpdatedBook < 0) //индекс -1(книга не найдена)

{

std::string const registrationNumberError{ "Ошибка.Книги с таким регистрационным номером не существует." };

std::cout << registrationNumberError << std::endl;

}

else

{

std::string const updateBookMessage{ "Редактирование книги" };

Book updatedBookBufer{ books[indexUpdatedBook] };//сохроняем в буфере книгу которую хотим изменить

Book updatedBook{ };

books.erase(books.begin() + indexUpdatedBook); //временно удаляем её из общего спитска

if (setBookParameters(books, updatedBook, updateBookMessage))//если данные книги введены успешно

{

system("cls"); //очистка кансоли

std::string const question{ "Вы действительно хотите изменить дынные в книги с регистрационным номером " + registrationNumberUpdatedBook + "?" };

if (confirmationAction(question)) // подтверждение изменения книги

{

books.emplace(books.begin() + indexUpdatedBook, updatedBook);// на место временно удалённой записи помешаем новую

system("cls");//очистка консоли

std::string const updateSuccessful{ "Данные книги с регистрационным номером " + registrationNumberUpdatedBook + " успешно изменены." };

std::cout << updateSuccessful << std::endl; //сообщение об успешном изменении

}

else //если изменения отменены

{

books.emplace(books.begin() + indexUpdatedBook, updatedBookBufer); //на место временно удалённой записи помещаем её буфер

system("cls");//очистка консоли

std::string const updateCanel{ "Изменения отменены." };

std::cout << updateCanel << std::endl;

}

}

else

{

books.emplace(books.begin() + indexUpdatedBook, updatedBookBufer);//на место временно удалённой записи помещаем её буфер

system("cls");//очистка консоли

std::string const updateError{ "Ошибка. Книга с таким индексом существует." };

std::cout << updateError << std::endl;

}

}

}

void deleteBookByAdministrator(std::vector<Book>& books)

{

system("cls");//очистка консоли

std::string const welcomeMessage{ "Введите регистрационный номер книги которую хотите удалить:" };

std::cout << welcomeMessage << std::endl;

std::string registrationNumberDeletedBook{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, registrationNumberDeletedBook);//чтение строки с наличием " "

int indexDeletedBook{ findBookIndex(books, std::atoi(registrationNumberDeletedBook.c\_str())) }; // поиск индекса книги с введенным регистрационным номером

if (indexDeletedBook < 0) //индекс -1(книга не найдена)

{

std::string const registrationNumberError{ "Ошибка.Книги с таким регистрационным номером не существует." };

std::cout << registrationNumberError << std::endl;

}

else

{

system("cls");//очистка консоли

std::string const question{ "Вы действительно хотите удалить книгу с регистрационным номером " + registrationNumberDeletedBook + "?" };

if (confirmationAction(question)) //подтвеждение удаления

{

books.erase(books.begin() + indexDeletedBook); //удаление книги

system("cls");//очистка консоли

std::string const deletedSuccessful{ "Книга с регистрационным номером " + registrationNumberDeletedBook + " успешно удалена." };

std::cout << deletedSuccessful << std::endl;

}

else

{

system("cls");//очистка консоли

std::string const deletedCanel{ "Удаление отменено." };

std::cout << deletedCanel << std::endl;

}

}

}

bool setBookParameters(const std::vector<Book>& books, Book &newBook, std::string const welcomeMessage)

{

unsigned int registrationNumber{ enteryNewBookRegistrationNumber(welcomeMessage) }; //ввод регистрационного номера

if (bookExists(books, registrationNumber)) //если книга с таким номером есть

{

return false; //завершение

}

else//продолжение ввода данных

{

newBook.registrationNumber = registrationNumber;

newBook.author = enteryNewBookAuthor(welcomeMessage);

newBook.name = enteryNewBookName(welcomeMessage);

newBook.yearPuplication = enteryNewBookYearPuplication(welcomeMessage);

newBook.publishingHouse = enteryNewBookPublishingHouse(welcomeMessage);

newBook.quantityPage = enteryNewBookQuantityPage(welcomeMessage);

newBook.lastReaderTicketNumber = enteryNewBookLastReader(welcomeMessage);

newBook.status = enteryNewBookStatus(welcomeMessage);

return true;

}

}

int unsigned enteryNewBookRegistrationNumber(std::string const welcomeMessage)

{

system("cls");//очистка консоли

std::string const messageInvitationEnterBookName{ "Введите регистрационный номер книги:" };

unsigned int newRegistrationNumber{ enteryUnsignetIntMeaning(welcomeMessage + "\n" + messageInvitationEnterBookName) }; //проверка корректности ввода

return newRegistrationNumber;

}

std::string enteryNewBookAuthor(std::string const welcomeMessage)

{

system("cls");//очистка консоли

std::string const messageInvitationEnterBookAthor{ "Введите автора книги:" };

size\_t minLen{ 0 };

for (; ; )

{

std::string newBookAthor{ enteryStringMeaning(welcomeMessage + "\n" + messageInvitationEnterBookAthor) };//проверка корректности ввода

if (newBookAthor.size() < minLen) //проверка на наличие автора

{

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageError{ "У книги должен быть автор" };

std::cout << messageError << std::endl;

}

else

{

return newBookAthor;

}

}

}

std::string enteryNewBookName(std::string const welcomeMessage)

{

//проверки аналогичны функции enteryNewBookAuthor

system("cls");//очистка консоли

std::string const messageInvitationEnterBookName{ "Введите название книги:" };

size\_t minNameLen{ 0 };

for (; ; )

{

std::string newBookName{ enteryStringMeaning(welcomeMessage + "\n" + messageInvitationEnterBookName) };

if (newBookName.size() < minNameLen)

{

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageError{ "У книги должно быть название" };

std::cout << messageError << std::endl;

}

else

{

return newBookName;

}

}

}

int unsigned enteryNewBookYearPuplication(std::string const welcomeMessage)

{

system("cls");//очистка консоли

std::string const messageInvitationEnterBookYearPuplication{ "Введите год издания книги:" };

int maxYear{ getYear() };//определение текущего года

for (; ; )

{

unsigned int newYearPuplication{ enteryUnsignetIntMeaning(welcomeMessage + "\n" + messageInvitationEnterBookYearPuplication) }; //проверка корректности ввода

if (newYearPuplication > maxYear) //проверка, что год не привышает текущий

{

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageError{ "Не корректный год издания" };

std::cout << messageError << std::endl;

}

else

{

return newYearPuplication;

}

}

}

std::string enteryNewBookPublishingHouse(std::string const welcomeMessage)

{

system("cls");//очистка консоли

std::string const messageInvitationEnterBookPublishingHouse{ "Введите издательство книги:" };

std::string newPublishingHouse{ enteryStringMeaning(welcomeMessage + "\n" + messageInvitationEnterBookPublishingHouse) };

return newPublishingHouse;

}

int unsigned enteryNewBookQuantityPage(std::string const welcomeMessage)

{

system("cls");//очистка консоли

std::string const messageInvitationEnterBookQuantityPage{ "Введите число страниц книги:" };

size\_t minPage{ 0 };

for (; ; )

{

unsigned int newPublishingHouse{ enteryUnsignetIntMeaning(welcomeMessage + "\n" + messageInvitationEnterBookQuantityPage) };

if (newPublishingHouse <= minPage) //проверка на не нулевое число страниц

{

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageError{ "Книга не может быть без страниц" };

std::cout << messageError << std::endl;

}

else

{

return newPublishingHouse;

}

}

}

int unsigned enteryNewBookLastReader(std::string const welcomeMessage)

{

system("cls");//очистка консоли

std::string const messageInvitationEnterLastReader{ "Введите номер читательского билета последнего читателя книги:" };

unsigned int newLastReader{ enteryUnsignetIntMeaning(welcomeMessage + "\n" + messageInvitationEnterLastReader) };

return newLastReader;

}

bookStatus enteryNewBookStatus(std::string const welcomeMessage)

{

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageInvitationChooseStatus{ "Выберите статус книги(ввести 0 или 1):\n0.Книга в библиотеки\n1.Книга на выдаче" };

std::cout << messageInvitationChooseStatus << std::endl;

for (; ; )

{

std::string statusInput{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, statusInput);//чтение строки с наличием " "

int const maxValue{ 1 };

if (checkingCorrectnessInputForSequences(statusInput, maxValue)) //проверка на корректность ввода

{

return bookStatus(std::atoi(statusInput.c\_str()));

}

else

{

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageIncorrectInput{ "Не верный ввод, попробуйте ещё раз." };

std::cout << messageIncorrectInput << std::endl;

std::cout << messageInvitationChooseStatus << std::endl;

}

}

}

unsigned int enteryUnsignetIntMeaning(std::string const welcomeMessage)

{

std::cout << welcomeMessage << std::endl;

for (; ; )

{

std::string newInt{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, newInt);//чтение строки с наличием " "

if (checkingCorrectnessInputUnsignetInt(newInt) && newInt.size()>0)

{

return std::atoi(newInt.c\_str());

}

else

{

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageError{ "Не корректный ввод. Попробуйте ещё раз" };

std::cout << messageError << std::endl;

}

}

}

std::string enteryStringMeaning(std::string const welcomeMessage)

{

std::cout << welcomeMessage << std::endl;

for (; ; )

{

std::string newString{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, newString);//чтение строки с наличием " "

if (checkingCorrectnessInputString(newString))

{

return newString;

}

else

{

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageError{ "Не корректный ввод. Попробуйте ещё раз" };

std::cout << messageError << std::endl;

}

}

}

bool bookExists(const std::vector<Book>& books, unsigned int const registrationNumber)

{

if (findBookIndex(books, registrationNumber) >= 0)

return true;

else

return false;

}

int findBookIndex(const std::vector<Book>& books, unsigned int const registrationNumber)

{

for (size\_t i{ 0 }; i < books.size(); ++i)

{

if (books[i].registrationNumber == registrationNumber)

return i;

}

return -1;

}

int getYear()

{

struct tm newtime;

time\_t now = time(0);

localtime\_s(&newtime, &now);//определение текущего времени

int startYear{ 1900 }; //отсчёт осуществляется с 1900, т.е. для используемой библиотеки он является 0

return startYear + newtime.tm\_year; // прибавляем 1900 к текущему году

}

**editingUsers.h**

#pragma once

#include <vector>

#include <iomanip>

#include <stdlib.h>

#include "User.h"

#include "userPassword.h"

#include "drawTable.h"

//создает первого пользователя(пользователя по умолчанию)

//принимает ссылку на вектор пользователей

void creatFirstUser(std::vector<User>& users);

//функция выполняет запрос на добавлении нового пользователя в базу данных администратором

//принимает ссылку на вектор пользователей

void addUsersByAdministrator(std::vector<User>& users);

//функция выполняет запрос на добавлении нового пользователя в базу данных простым пользователем

//принимает ссылку на вектор пользователей

void addUsersBySimpleUser(std::vector<User>& users);

//функция служит для установки логина и пароля для пользователя

//принимает ссылку на вектор пользователей и сообщение которое должно выводится при вводе данных

//возврощает пользователя с введённым паролем и логином

User setUserLoginAndPassword(const std::vector<User>& users, std::string const welcomeMessage);

//функция служит для ввода логина пользователя

//принимает ссылку на вектор пользователей и сообщение которое должно выводится при вводе данных

//возвращает строку удовлетворяющие требованием к логину

std::string enteryNewUserLogin(const std::vector<User>& users, std::string const welcomeMessage);

//функция служит для ввода пароля пользователя

//принимает сообщение которое должно выводится при вводе данных

//возвращает строку удовлетворяющие требованием к паролю

std::string enteryNewUserPasswor(std::string const welcomeMessage);

//функция служит для ввода прав пользователя(пользователь или администратор)

//принимает сообщение которое должно выводится при вводе данных

//возвращает права пользователя

userRole enteryNewUserRole(std::string const welcomeMessage);

//функция служит для ввода статуса пользователя(активен или заблокирован)

//принимает сообщение которое должно выводится при вводе данных и

//уровень прав пользователя для которого изменяют статус

//если пользователя активировали функция вернёт true иначе false

bool enteryNewUserAccess(std::string const welcomeMessage, userRole const role);

//вывод индекса под которым хранится пользователь с заданным логином

//на вход принимает ссылку на вектор пользователей и искомый логин

//если пользователь с таким логином найден функция вернёт его индекс иначе -1

int findUserIndex(const std::vector<User>& users, std::string const login);

//проверка на существования пользователя с заданным логином

//на вход принимает ссылку на вектор пользователей и искомый логин

//если пользователь с таким логином найден функция вернёт true иначе false

bool loginExists(const std::vector<User>& users, std::string const login);

//функция выполняет запрос на изменение информации о пользователи администратором

//принимает ссылку на вектор пользователей и логин активного пользователя

void updateUserDataByAdministrator(std::vector<User>& users, std::string const activUserLogin);

//функция служит для ввода всей информации о пользователе

//принимает ссылку на вектор пользователей и сообщение которое должно выводится при вводе данных

//возврощает пользователя с введённой информаицей

User setAllUserData(const std::vector<User>& users, std::string const welcomeMessage);

//функция служит для изменения данных о пользователе в базе данных

//принимает ссылку на вектор пользователей, сыслку на пользователя парметры которого планируется изменить и

//логин активного пользователя

//если обнавление данных прошло успешно функция вернёт true иначе false

bool updateUserData(const std::vector<User>& users, User& updatedUser, std::string const activUserLogin);

//функция выполняет запрос на удаление пользователя администратором

//принимает ссылку на вектор пользователей и логин активного пользователя

void deleteUserDataByAdministrator(std::vector<User>& users, std::string const activUserLogin);

//функция служит для удаления пользователя из базы данных

//принимает ссылку на вектор пользователей, индекс проверяемого пользователя и логин активного пользователя

//если пользователя удалён функция вернёт true иначе false

bool deleteUser(std::vector<User>& const users, size\_t const indexDeletedUser, std::string const activUserLogin);

//функция выполняет запрос на активацию пользователя администратором

//принимает ссылку на вектор пользователей

void activationUserByAdministrator(std::vector<User>& users);

//функция выполняет запрос на блокировку пользователя администратором

//принимает ссылку на вектор пользователей и логин активного пользователя

void blockUserByAdministrator(std::vector<User>& users, std::string const activUserLogin);

//вывод данных о всех пользователях

//принимает ссылку на вектор пользователей

void printUsers(const std::vector<User> &users);

**editingUsers.cpp**

#include "editingUsers.h"

void creatFirstUser(std::vector<User>& users)

{

users.push\_back(User{});//добавление в вектор

std::string const loginFirstUser{ "Admin" };

users[0].login = loginFirstUser;

std::string const saltFirstUser{ createSaltForUser(loginFirstUser) }; //генерация "соли"

users[0].salt = saltFirstUser;

std::string const standartPassword{ "1111" };//стандартный пароль

users[0].saltedHashPassword = convertUserPasswordToHash(standartPassword + saltFirstUser, loginFirstUser);//хэширование пароля

users[0].role = userRole::ADMINISTRATOR;

users[0].access = true;

}

void addUsersByAdministrator(std::vector<User>& users)

{

std::string const messageNewUserRegistration{ "Регистрация нового пользователя" };

User newUser{ setUserLoginAndPassword(users, messageNewUserRegistration) }; //установка пользователю логина и пароля

userRole newUserRole{ enteryNewUserRole(messageNewUserRegistration) }; //установка прав доступа

newUser.role = newUserRole;

newUser.access = true; //если пользователь создан администратором, он сразу является активным

users.push\_back(newUser);

system("cls");//очистка кансоли

std::string const messageNewUserCreate{ "Новый пользователь " + newUser.login + " успешно создан." };

std::cout << messageNewUserCreate << std::endl;

}

void addUsersBySimpleUser(std::vector<User>& users)

{

std::string const messageNewUserRegistration{ "Регистрация нового пользователя" };

User newUser{ setUserLoginAndPassword(users, messageNewUserRegistration) }; //установка логина и пароля пользователю

users.push\_back(newUser); //добавление в вектор

system("cls");//очистка кансоли

std::string const messageNewUserCreate{ "Новый пользователь " + newUser.login + " успешно создан. Ожидайте подтверждения администратором." };

std::cout << messageNewUserCreate << std::endl;

}

User setUserLoginAndPassword(const std::vector<User>& users, std::string const welcomeMessage)

{

User outUser{};

system("cls");//очистка кансоли

std::string newUserLogin{ enteryNewUserLogin(users, welcomeMessage) };//ввод логина

outUser.login = newUserLogin;

system("cls");//очистка кансоли

std::string originalPassword{ enteryNewUserPasswor(welcomeMessage) };//ввод пароля

std::string newUserSalt{ createSaltForUser(outUser.login) }; //генерация соли

outUser.salt = newUserSalt;

outUser.saltedHashPassword = convertUserPasswordToHash(originalPassword + newUserSalt, newUserLogin);//хеширование пароля

return outUser;

}

std::string enteryNewUserLogin(const std::vector<User>& users, std::string const welcomeMessage)

{

std::cout << welcomeMessage << std::endl;

std::string const messageInvitationEnterLogin{ "Введите желаемый логин:" };

std::cout << messageInvitationEnterLogin << std::endl;

size\_t minLoginLen{ 3 }; //установка минимальной длины логина

for ( ; ; )

{

std::string newLogin{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, newLogin);//чтение строки с наличием " "

if (loginExists(users, newLogin)) //проверка на существование введённого логина

{

//логин уже занят

system("cls");//очистка консоли

std::cout << welcomeMessage << std::endl;

std::string const messageLoginDusy{ "Логин " + newLogin + " уже занят." };

std::cout << messageLoginDusy << std::endl;

std::string const messageInvitationEnterOtherLogin{ "Введите другой логин:" };

std::cout << messageInvitationEnterOtherLogin << std::endl;

}

else if (newLogin.size() < minLoginLen) //проверка соответствия минимальной длине

{

system("cls");//очистка кансоли

std::cout << welcomeMessage << std::endl;

std::string const messageLoginShort{ "Логин " + newLogin + " слишком короткий. Логин должен состоять не мене чем из " + std::to\_string(minLoginLen) + " символов." };

std::cout << messageLoginShort << std::endl;

std::string const messageInvitationEnterOtherLogin{ "Введите другой логин:" };

std::cout << messageInvitationEnterOtherLogin << std::endl;

}

else

{

return newLogin;

}

}

}

std::string enteryNewUserPasswor(std::string const welcomeMessage)

{

std::cout << welcomeMessage << std::endl;

std::string const messageInvitationEnterPassword{ "Введите пароль:" };

std::cout << messageInvitationEnterPassword << std::endl;

size\_t minPasswordLen{ 3 };//минимальная длина пароля

for ( ; ; )

{

std::string firstPasswordInput{ readPassword() };//ввод пароля первый раз

std::cout << std::endl;

if (firstPasswordInput.size() < minPasswordLen)//проверка соответствия минимальной длине

{

system("cls");//очистка кансоли

std::cout << welcomeMessage << std::endl;

std::string const messageLoginShort{ "Пароль слишком короткий. Пароль должен состоять не мене чем из " + std::to\_string(minPasswordLen) + " символов." };

std::cout << messageLoginShort << std::endl;

std::string const messageInvitationEnterOtherLogin{ "Введите другой логин:" };

std::cout << messageInvitationEnterOtherLogin << std::endl;

}

else

{

std::string const messageInvitationRepeatPassword{ "Повторите пароль:" };

std::cout << messageInvitationRepeatPassword << std::endl;

std::string secondPasswordInput{ readPassword() };//повторение пароля

std::cout << std::endl;

if (firstPasswordInput == secondPasswordInput) //пароли совпали

{

return firstPasswordInput;

}

else

{

system("cls");//очистка кансоли

std::cout << welcomeMessage << std::endl;

std::string const messageIncorrectPassword{ "Не верный пароль, попробуйте ещё раз." };

std::cout << messageIncorrectPassword << std::endl;

std::cout << messageInvitationEnterPassword << std::endl;

}

}

}

}

userRole enteryNewUserRole(std::string const welcomeMessage)

{

system("cls");//очистка кансоли

std::cout << welcomeMessage << std::endl;

std::string const messageInvitationChooseRole{ "Выберите права пользователя(ввести 0 или 1):\n0.Простой пользователь\n1.Администратор" };

std::cout << messageInvitationChooseRole << std::endl;

for (; ; )

{

std::string roleInput{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, roleInput);//чтение строки с наличием " "

int const maxValue{ 1 };//можно ввести 0 и 1

if (checkingCorrectnessInputForSequences(roleInput, maxValue)) //проверка кооректности ввода

{

return userRole(std::atoi(roleInput.c\_str()));

}

else

{

system("cls");//очистка кансоли

std::cout << welcomeMessage << std::endl;

std::string const messageIncorrectInput{ "Не верный ввод, попробуйте ещё раз." };

std::cout << messageIncorrectInput << std::endl;

std::cout << messageInvitationChooseRole << std::endl;

}

}

}

bool enteryNewUserAccess(std::string const welcomeMessage, userRole const role)

{

if (role == userRole::ADMINISTRATOR)

{

return 1;

}

system("cls");//очистка кансоли

std::cout << welcomeMessage << std::endl;

std::string const messageInvitationChooseRole{ "Блокироватьь или активироват пользователя(ввести 0 или 1):\n0.Блокировать пользователь\n1.Активировать пользователь" };

std::cout << messageInvitationChooseRole << std::endl;

for (; ; )

{

std::string accessInput{};

\_flushall();

std::getline(std::cin, accessInput);

int const maxValue{ 1 }; //можно ввести 0 и 1

if (checkingCorrectnessInputForSequences(accessInput, maxValue))//проверка корректности ввода

{

return std::atoi(accessInput.c\_str()); //преобразование из string в int

}

else

{

system("cls");//очистка кансоли

std::cout << welcomeMessage << std::endl;

std::string const messageIncorrectInput{ "Не верный ввод, попробуйте ещё раз." };

std::cout << messageIncorrectInput << std::endl;

std::cout << messageInvitationChooseRole << std::endl;

}

}

}

User setAllUserData(const std::vector<User>& users, std::string const welcomeMessage)

{

User newUser{ setUserLoginAndPassword(users, welcomeMessage) };

userRole newUserRole{ enteryNewUserRole(welcomeMessage) };

bool newUserAccess{ enteryNewUserAccess(welcomeMessage, newUserRole) };

newUser.role = newUserRole;

newUser.access = newUserAccess;

return newUser;

}

bool updateUserData(const std::vector<User>& users, User & updatedUser, std::string const activUserLogin)

{

std::string const informMassege{ "Обновление данных пользователя " + updatedUser.login };

User newUser{ setAllUserData(users, informMassege) };

if (updatedUser.login == activUserLogin) // если пользователь пытается изменить свои данные

{

if (updatedUser.access != newUser.access || updatedUser.role != newUser.role) // и изменил уровень прав или заблокировал себя

return false;//запретить изменения

}

updatedUser = newUser; //обновить данные пользователя

return true;//разрешить изменения

}

void updateUserDataByAdministrator(std::vector<User>& users, std::string const activUserLogin)

{

system("cls");//очистка кансоли

std::string const welcomeMessage{ "Введите логин пользователя которого хотите редактировать:" };

std::cout << welcomeMessage << std::endl;

std::string loginUpdatedUser{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, loginUpdatedUser);//чтение строки с наличием " "

int indexUpdatedUser{ findUserIndex(users, loginUpdatedUser) }; //поиск индекса пользователя по логину

if (indexUpdatedUser < 0)//индекс -1(пользователь не найден)

{

std::string const loginError{ "Ошибка.Пользователя с таким логином не существует." };

std::cout << loginError << std::endl;

}

else

{

User updatedUserBufer{ users[indexUpdatedUser] };// буфер хранящий данные изменяемого пользователя

User updatedUser{ };// место для хранения вновь введённых данных

users.erase(users.begin() + indexUpdatedUser);//временное удаление пользователя из вектора

if (updateUserData(users, updatedUser, activUserLogin)) //если данные были ведены успешно

{

system("cls");//очистка кансоли

std::string const question { "Вы действительно хотите изменить дынные пользователя " + loginUpdatedUser + "?"};

if (confirmationAction(question))//если подтвержденно изменения пользователя

{

users.emplace(users.begin() + indexUpdatedUser, updatedUser);//на место временно удаленного пользователя вернуть обнавлённые данные

system("cls");//очистка кансоли

std::string const updateSuccessful{ "Данные пользователя " + loginUpdatedUser + " успешно изменены." };

std::cout << updateSuccessful << std::endl;

}

else

{

users.emplace(users.begin() + indexUpdatedUser, updatedUserBufer);//на место временно удаленного пользователя вернуть буферные данные

system("cls");//очистка кансоли

std::string const updateCanel{ "Изменения отменены." };

std::cout << updateCanel << std::endl;

}

}

else

{

users.emplace(users.begin() + indexUpdatedUser, updatedUserBufer);//на место временно удаленного пользователя вернуть буферные данные

system("cls");//очистка кансоли

std::string const updateError{ "Ошибка. Запрещенно менять собственный уровень доступа." };

std::cout << updateError << std::endl;

}

}

}

void deleteUserDataByAdministrator(std::vector<User>& users, std::string const activUserLogin)

{

system("cls");//очистка кансоли

std::string const welcomeMessage{ "Введите логин пользователя которого хотите удалить:" };

std::cout << welcomeMessage << std::endl;

std::string loginDeletedUser{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, loginDeletedUser);//чтение строки с наличием " "

int indexDeletedUser{ findUserIndex(users, loginDeletedUser) }; //поиск индекса пользователя по логину

if (indexDeletedUser < 0) //индекс -1 логин не найден

{

std::string const loginError{ "Ошибка.Пользователя с таким логином не существует." };

std::cout << loginError << std::endl;

}

else

{

system("cls");//очистка кансоли

std::string const question{ "Вы действительно хотите удалить пользователя " + loginDeletedUser + "?" };

if (confirmationAction(question))//подтверждение удаления пользователя

{

if (deleteUser(users, indexDeletedUser, activUserLogin)) //пользователь был удалён

{

system("cls");//очистка кансоли

std::string const deleteSuccessful{ "Данные пользователя " + loginDeletedUser + " успешно удалены." };

std::cout << deleteSuccessful << std::endl;

}

else //пользователь не может быть удалён

{

system("cls");//очистка кансоли

std::string const deleteError{ "Ошибка. Запрещенно удалять собственного пользователя." };

std::cout << deleteError << std::endl;

}

}

else//отмена удаления

{

system("cls");//очистка кансоли

std::string const deleteCanel{ "Удаление отменены." };

std::cout << deleteCanel << std::endl;

}

}

}

bool deleteUser(std::vector<User>& const users, size\_t const indexDeletedUser, std::string const activUserLogin)

{

if(users[indexDeletedUser].login == activUserLogin) // проверка на то, что бы пользователь не удалил сам себя

return false;//удаление не завершено

else

{

users.erase(users.begin() + indexDeletedUser); //удаления пользователя

return true; //успешно удален

}

}

void activationUserByAdministrator(std::vector<User>& users)

{

system("cls");//очистка кансоли

std::string const welcomeMessage{ "Введите логин пользователя которого хотите активировать:" };

std::cout << welcomeMessage << std::endl;

std::string loginActivatedUser{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, loginActivatedUser);//чтение строки с наличием " "

int indexActivatedUser{ findUserIndex(users, loginActivatedUser) };//поиск индекса пользователя по логину

if (indexActivatedUser < 0)//индекс -1, пользователь не найден

{

std::string const loginError{ "Ошибка.Пользователя с таким логином не существует." };

std::cout << loginError << std::endl;

}

else

{

system("cls");//очистка кансоли

std::string const question{ "Вы действительно хотите активировать пользователя " + loginActivatedUser + "?" };

if (confirmationAction(question)) //подтверждение активации пользователя

{

users[indexActivatedUser].access = true;//актиация пользователя

system("cls");//очистка кансоли

std::string const activatedSuccessful{ "Пользователь " + loginActivatedUser + " успешно активирован." };

std::cout << activatedSuccessful << std::endl;

}

else//отмена активации

{

system("cls");//очистка кансоли

std::string const activatedCanel{ "Активация отменена." };

std::cout << activatedCanel << std::endl;

}

}

}

void blockUserByAdministrator(std::vector<User>& users, std::string const activUserLogin)

{

system("cls");//очистка кансоли

std::string const welcomeMessage{ "Введите логин пользователя которого хотите заблокировать:" };

std::cout << welcomeMessage << std::endl;

std::string loginBlockedUser{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, loginBlockedUser);//чтение строки с наличием " "

int indexBlockedUser{ findUserIndex(users, loginBlockedUser) };//поиск индекса пользователя по логину

if (indexBlockedUser < 0)//индекс -1, пользователь не найден

{

std::string const loginError{ "Ошибка.Пользователя с таким логином не существует." };

std::cout << loginError << std::endl;

}

else

{

system("cls");//очистка кансоли

if (loginBlockedUser == activUserLogin)//если пользователь пытается сам себя заблокировать

{

std::string const blockedError{ "Ошибка. Пользователь не может заблокировать себя." };

std::cout << blockedError << std::endl;

return; //прерывание функции

}

std::string const question{ "Вы действительно хотите блокировать пользователя " + loginBlockedUser + "?" };

if (confirmationAction(question)) //подверждение блокировки

{

users[indexBlockedUser].access = false; //блокировка пользователя

system("cls");//очистка кансоли

std::string const blockedSuccessful{ "Пользователь " + loginBlockedUser + " успешно заблокирован." };

std::cout << blockedSuccessful << std::endl;

}

else //отмена блокировки

{

system("cls");//очистка кансоли

std::string const blockedCanel{ "Блокировка отменена." };

std::cout << blockedCanel << std::endl;

}

}

}

bool loginExists(const std::vector<User>& users, std::string const login)

{

if (findUserIndex(users, login) >= 0)

return true;

return false;

}

int findUserIndex(const std::vector<User>& users, std::string const login)

{

for (size\_t i{}; i < users.size(); ++i)

{

if (users[i].login == login)

return i;

}

return -1;

}

void printUsers(const std::vector<User> & users)

{

std::vector<size\_t> columnSize{ 20,30,30,20,20 };//размеры калонок

std::vector<std::string> columnNames{ "Логин","Хэш пароля","Соль пароля","Права","Статус" };//заголовки калонок

drawTableRaw(columnSize, columnNames, true);//рисование заголовка

for (size\_t i{ 0 }; i < users.size(); ++i)//рисование данных

{

std::vector<std::string> userData{};

userData.push\_back(users[i].login);

userData.push\_back(users[i].saltedHashPassword);

userData.push\_back(users[i].salt);

userData.push\_back(std::to\_string(static\_cast<int>(users[i].role))); //конвертация перечесления в int затем string

userData.push\_back(std::to\_string(users[i].access)); //ковертаци bool в string

if(i < users.size() - 1)

drawTableRaw(columnSize, userData);

else

drawTableRaw(columnSize, userData, true);//рисование нижней границы для последней строки

}

}

**forMainFunction.h**

#pragma

#include <Windows.h>

#include "workingWithLibraryFile.h"

#include "workingWithUserFile.h"

//устанавливает русскую локализацию для консоли

void setRusSettings();

//функция предназначена для загрузки требуемых файлов в вектора

//функция принимает имя файла с БД библиотечных книг, ссылку на вектор для хранения книг,

//имя файла с БД пользователей, ссылку на вектор для хранения пользователей

void downloadData(std::string const libraryFileName, std::vector<Book>& books, std::string const usersFileName, std::vector<User>& users);

//функция предназначена для загрузки в файлы данных из соответствующих векторов

//функция принимает имя файла в который надо записать данные библиотечных книг, ссылку на вектор с данными книг,

//имя файла в который надо записать данные пользователей, ссылку на вектор с данными пользователей

void writeData(std::string const libraryFileName, const std::vector<Book>& books, std::string const usersFileName, const std::vector<User>& users);

**forMainFunction.cpp**

#include "forMainFunction.h"

void setRusSettings()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "Russian");

}

void downloadData(std::string const libraryFileName, std::vector<Book>& books, std::string const usersFileName, std::vector<User>& users)

{

downloadLibraryData(libraryFileName, books);

downloadUsersData(usersFileName, users);

}

void writeData(std::string const libraryFileName, const std::vector<Book>& books, std::string const usersFileName, const std::vector<User>& users)

{

writeUserDataToFile(usersFileName, &users);

writeLibraryDataToFile(libraryFileName, &books);

}

**menu.h**

#pragma once

#include <iostream>

#include "editingUsers.h"

#include "editingLibrary.h"

#include "procesingLibrary.h"

#include "userPassword.h"

#include "checkFunction.h"

//меню входа в систему

//принимает ссылку на вектор пользователей и ссылку на вектор библиотечных книг

void inputMenu(std::vector<User>& users, std::vector<Book>& booksInLibrary);

//главное меню пользователя

//принимает ссылку на вектор библиотечных книг и активного пользователя

void simpleUserMenu(std::vector<Book>& booksInLibrary, User const activUser);

//главное меню администратора

//принимает ссылку на вектор пользователей, ссылку на вектор библиотечных книг и активного пользователя

void administratorMenu(std::vector<User>& users, std::vector<Book>& booksInLibrary, User const activUser);

//меню изменения параметров пользователей

//принимает ссылку на вектор пользователей и активного пользователя

void menuEditingUsers(std::vector<User>& users, User const activUser);

//меню изменения параметров библиотечных книг

//принимает ссылку на вектор библиотечных книг

void menuEditingLibrary(std::vector<Book>& booksInLibrary);

//меню обработки библиотечных книг

//принимает ссылку на вектор библиотечных книг

void menuProcessingLibrary(std::vector<Book>& booksInLibrary);

//меню типов сортировки книг

//принимает ссылку на вектор библиотечных книг

void menuSortedBooks(std::vector<Book>& booksInLibrary);

//меню поиска книг

//принимает ссылку на вектор библиотечных книг

void menuSearcBooks(const std::vector<Book>& booksInLibrary);

//проверка на активацию пользователя

//если пользователь активирован функция вернёт true иначе false

bool isActivatedUser(User const user);

**menu.cpp**

#include "menu.h"

void inputMenu(std::vector<User>& users, std::vector<Book>& booksInLibrary)

{

for (; ; )

{

std::string const message{ "1.Войти\n2.Регистрация\n0.Завершение работы" };

std::cout << message << std::endl;

std::string answer{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, answer);//чтение строки с наличием " "

int const maxAnswer{ 2 }; //возможные пункты 0,1,2

if (checkingCorrectnessInputForSequences(answer, maxAnswer))//проверка корректности ввода

{

switch (std::atoi(answer.c\_str())) //конвертация string из в int

{

case 1:

{

User activUser{}; // авторизораванный пользователь

if (authorization(users, activUser))

{

system("cls");//очистка консоли

std::string const welcomeMessage{ "Приветствуем " + activUser.login + "!"};

std::cout << welcomeMessage << std::endl;

if (activUser.role == userRole::SIMPLE\_USER)

{

simpleUserMenu(booksInLibrary, activUser);//меню пользователя

}

else

{

administratorMenu(users, booksInLibrary, activUser);//меню администратора

}

}

else

{

system("cls");//очистка консоли

std::string const authorizationError{ "Не верный логин или пароль." };

std::cout << authorizationError << std::endl;

}

break;

}

case 2:

{

addUsersBySimpleUser(users);//добавления пользователя самостоятельно

break;

}

case 0:

{

return;// выход из функции, завершение программы

break;

}

}

}

else

{

system("cls");//очистка консоли

std::string const errorInput{ "Некорректный ввод. Попробуйте ещё раз." };

std::cout << errorInput << std::endl;

}

}

}

void simpleUserMenu(std::vector<Book>& booksInLibrary, User const activUser)

{

if (isActivatedUser(activUser))//если пользователь активирован

{

menuProcessingLibrary(booksInLibrary);//меню обработки данных библиотеки

}

else

{

std::string userNotActivatedMessage{ "Пользователь " + activUser.login + " ещй не активирован. Ожидайте подтвержение записи администратором." };

std::cout << userNotActivatedMessage << std::endl;

}

}

void administratorMenu(std::vector<User>& users, std::vector<Book>& booksInLibrary, User const activUser)

{

if (isActivatedUser(activUser))//если пользователь активен

{

for (; ; )

{

std::string const welcomeMessage{ "Выберите режим работы:\n1.Управление данными пользователей\n2.Управление данными библиотеки\n3.Обработка данных библиотеки\n0.Назад" };

std::cout << welcomeMessage << std::endl;

std::string answer{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, answer);//чтение строки с наличием " "

int const maxAnswer{ 3 };//возможные пункты 0,1,2,3

if (checkingCorrectnessInputForSequences(answer, maxAnswer))//проверка корректности ввода

{

system("cls");//очистка консоли

switch (std::atoi(answer.c\_str())) //конвертация string из в int

{

case 1: {

menuEditingUsers(users, activUser);//меню работы с пользователями

break;

}

case 2: {

menuEditingLibrary(booksInLibrary);//меню работы книгами

break;

}

case 3: {

menuProcessingLibrary(booksInLibrary);//меню обработки книг

break;

}

case 0: {

return;// выход из функции, возвращение в предыдушие меню

}

}

}

else

{

system("cls");//очистка консоли

std::string const errorInput{ "Некорректный ввод. Попробуйте ещё раз." };

std::cout << errorInput << std::endl;

}

}

}

else

{

std::string userNotActivatedMessage{ "Пользователь " + activUser.login + " ещй не активирован. Ожидайте подтвержение записи администратором." };

std::cout << userNotActivatedMessage << std::endl;

}

}

void menuEditingUsers(std::vector<User>& users, User const activUser)

{

for (; ; )

{

std::string const welcomeMessage{ "1.Просмотр всех данных пользователей\n2.Добавление нового пользователя\n"

"3.Редактирование пользователя\n4.Активатия пользователя\n5.Блокировка пользователя\n"

"6.Удаление пользователя\n0.Назад" };

std::cout << welcomeMessage << std::endl;

std::string answer{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, answer);//чтение строки с наличием " "

int const maxAnswer{ 6 };//возможные пункты 0,1,2,3,4,5,6

if (checkingCorrectnessInputForSequences(answer, maxAnswer))//проверка корректности ввода

{

system("cls");//очистка консоли

switch (std::atoi(answer.c\_str()))//конвертация string из в int

{

case 1: {

printUsers(users);//вывести пользователей

break;

}

case 2: {

addUsersByAdministrator(users);//добавить пользователя

break;

}

case 3: {

updateUserDataByAdministrator(users, activUser.login); //редактировать пользователя

break;

}

case 4: {

activationUserByAdministrator(users);//активировать пользователя

break;

}

case 5: {

blockUserByAdministrator(users, activUser.login);//блокировать пользователя

break;

}

case 6: {

deleteUserDataByAdministrator(users, activUser.login);//удалить пользователя

break;

}

case 0: {

return;// выход из функции, возвращение в предыдушие меню

}

}

}

else

{

system("cls");//очистка консоли

std::string const errorInput{ "Некорректный ввод. Попробуйте ещё раз." };

std::cout << errorInput << std::endl;

}

}

}

void menuEditingLibrary(std::vector<Book>& booksInLibrary)

{

for (; ; )

{

std::string const welcomeMessage{ "1.Просмотр всех книг в библиотеки\n2.Добавление новой книги\n"

"3.Редактирование книги\n4.Удаление книги\n0.Назад" };

std::cout << welcomeMessage << std::endl;

std::string answer{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, answer);//чтение строки с наличием " "

int const maxAnswer{ 4 };//возможные пункты 0,1,2,3,4

if (checkingCorrectnessInputForSequences(answer, maxAnswer))//проверка корректности ввода

{

system("cls");//очистка консоли

switch (std::atoi(answer.c\_str()))//конвертация string из в int

{

case 1: {

printBooks(booksInLibrary);//вывести книги

break;

}

case 2: {

addNewBookByAdministrator(booksInLibrary);//добаввить книгу

break;

}

case 3: {

updateBookByAdministrator(booksInLibrary);//редактировать книгу

break;

}

case 4: {

deleteBookByAdministrator(booksInLibrary);//удалить книгу

break;

}

case 0: {

return;// выход из функции, возвращение в предыдушие меню

}

}

}

else

{

system("cls");//очистка консоли

std::string const errorInput{ "Некорректный ввод. Попробуйте ещё раз." };

std::cout << errorInput << std::endl;

}

}

}

void menuProcessingLibrary(std::vector<Book>& booksInLibrary)

{

for (; ; )

{

std::string const welcomeMessage{ "1.Просмотр всех книг в библиотеки\n2.Сортировка книг\n"

"3.Поиск книги\n4.Сортировка по автору и поиск по году\n5.Просмотр книг находящихся на выдачи\n0.Назад" };

std::cout << welcomeMessage << std::endl;

std::string answer{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, answer);//чтение строки с наличием " "

int const maxAnswer{ 5 };//возможные пункты 0,1,2,3,4,5

if (checkingCorrectnessInputForSequences(answer, maxAnswer))//проверка корректности ввода

{

system("cls");//очистка консоли

switch (std::atoi(answer.c\_str()))//конвертация string из в int

{

case 1: {

printBooks(booksInLibrary);//вывести книги

break;

}

case 2: {

menuSortedBooks(booksInLibrary);//меню сортировки

break;

}

case 3: {

menuSearcBooks(booksInLibrary);//меню поиска

break;

}

case 4: {//сортировка по фамилии автора и вывод книг изданных после заданного года

sortBooksByAuthor(booksInLibrary);//сортировка по автору

searchQueryByYearPuolicationMoreThen(booksInLibrary);//поиск книг изданных после .. года

break;

}

case 5: {

searchQueryBooksOnIssue(booksInLibrary);//вывод книг на выдаче

break;

}

case 0: {

return;// выход из функции, возвращение в предыдушие меню

}

}

}

else

{

system("cls");//очистка консоли

std::string const errorInput{ "Некорректный ввод. Попробуйте ещё раз." };

std::cout << errorInput << std::endl;

}

}

}

void menuSortedBooks(std::vector<Book>& booksInLibrary)

{

for (; ; )

{

std::string const welcomeMessage{ "1.Сортировка по году издания\n2.Сортировка по фамилии автора\n"

"3.Сортировка по числу страниц\n0.Назад" };

std::cout << welcomeMessage << std::endl;

std::string answer{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, answer);//чтение строки с наличием " "

int const maxAnswer{ 3 };//возможные пункты 0,1,2,3

if (checkingCorrectnessInputForSequences(answer, maxAnswer))//проверка корректности ввода

{

system("cls");//очистка консоли

switch (std::atoi(answer.c\_str()))//конвертация string из в int

{

case 1: {

sortBooksByYearPublication(booksInLibrary);//сортировка по году публикации

break;

}

case 2: {

sortBooksByAuthor(booksInLibrary);//сортировка по автору

break;

}

case 3: {

sortBooksByQuantityPage(booksInLibrary); //сортирова по числу страниц

break;

}

case 0: {

return;// выход из функции, возвращение в предыдушие меню

}

}

}

else

{

system("cls");//очистка консоли

std::string const errorInput{ "Некорректный ввод. Попробуйте ещё раз." };

std::cout << errorInput << std::endl;

}

}

}

void menuSearcBooks(const std::vector<Book>& booksInLibrary)

{

for (; ; )

{

std::string const welcomeMessage{ "1.Поиск книги по автору\n2.Поиск книги по году издания\n"

"3.Поиск книги по издательству\n4.Поиск книги после ... года издания\n0.Назад" };

std::cout << welcomeMessage << std::endl;

std::string answer{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, answer);//чтение строки с наличием " "

int const maxAnswer{ 4 };//возможные пункты 0,1,2,3,4

if (checkingCorrectnessInputForSequences(answer, maxAnswer))//проверка корректности ввода

{

system("cls");//очистка консоли

switch (std::atoi(answer.c\_str()))//конвертация string из в int

{

case 1: {

searchQueryByAuthor(booksInLibrary);//поиск книг по автору

break;

}

case 2: {

searchQueryByYearPuolication(booksInLibrary); //поиск книг по году издания

break;

}

case 3: {

searchQueryByPublishingHouse(booksInLibrary);//поиск книг по издательству

break;

}

case 4: {

searchQueryByYearPuolicationMoreThen(booksInLibrary);//поиск книг изданных после ... года

break;

}

case 0: {

return;// выход из функции, возвращение в предыдушие меню

}

}

}

else

{

system("cls");//очистка консоли

std::string const errorInput{ "Некорректный ввод. Попробуйте ещё раз." };

std::cout << errorInput << std::endl;

}

}

}

bool isActivatedUser(User const user)

{

return user.access;

}

**procesingLibrary.h**

#pragma once

#include <iostream>

#include <string>

#include <algorithm>

#include "editingLibrary.h"

#include "drawTable.h"

#include "Book.h"

//запрос на поиск книги по автору

//принимает ссылку на вектор книг

void searchQueryByAuthor(const std::vector<Book>& books);

//запрос на поиск книги по году издания

//принимает ссылку на вектор книг

void searchQueryByYearPuolication(const std::vector<Book>& books);

//запрос на поиск книги изданной после заданного года

//принимает ссылку на вектор книг

void searchQueryByYearPuolicationMoreThen(const std::vector<Book>& books);

//запрос на поиск книги по издательству

//принимает ссылку на вектор книг

void searchQueryByPublishingHouse(const std::vector<Book>& books);

//запрос на вывод книг находящихся на выдаче

//принимает ссылку на вектор книг

void searchQueryBooksOnIssue(const std::vector<Book>& books);

//функция поиска книги по автору

//принимает ссылку на вектор книг, искомого автора, ссылку на вектор книг удовлетворяющих запросу

void findBooksByAuthor(const std::vector<Book>& books, std::string const author, std::vector<Book>& outBooks);

//функция поиска книги по году издания

//принимает ссылку на вектор книг, искомый год издания, ссылку на вектор книг удовлетворяющих запросу

void findBooksByYearPuolication(const std::vector<Book>& books, unsigned int const yearPuolication, std::vector<Book>& outBooks);

//функция поиска книги изданной после заданного года

//принимает ссылку на вектор книг, требуемый год издания, ссылку на вектор книг удовлетворяющих запросу

void findBooksByYearPuolicationMoreThen(const std::vector<Book>& books, unsigned int const minYearPuolication, std::vector<Book>& outBooks);

//функция поиска книги по издательству

//принимает ссылку на вектор книг, искомого издательства, ссылку на вектор книг удовлетворяющих запросу

void findBooksByPublishingHouse(const std::vector<Book>& books, std::string const publishingHouse, std::vector<Book>& outBooks);

//функция поиска книг находящихся на выдаче

//принимает ссылку на вектор книг и ссылку на вектор книг удовлетворяющих запросу

void findBooksOnIssue(const std::vector<Book>& books, std::vector<Book>& outBooks);

//сортировка книг по году издания

//принимает ссылку на вектор книг

//возращает входной вектор отсортированный по году издания

void sortBooksByYearPublication(std::vector<Book> &books);

//сортировка книг по автору

//принимает ссылку на вектор книг

//возращает входной вектор отсортированный по автор

void sortBooksByAuthor(std::vector<Book> &books);

//сортировка книг по числу страниц

//принимает ссылку на вектор книг

//возращает входной вектор отсортированный по числу страниц

void sortBooksByQuantityPage(std::vector<Book> &books);

//сортировка по полю года издания

//входные параметры сравниваемые книги

//если год издания первой книги меньше второй функция вернёт true иначе false

bool sortByYearPublication(Book firstBook, Book secondBook);

//сортировка по полю автора

//входные параметры сравниваемые книги

//если автор первой книги в алфавитном порядке находится перед автором второй функция вернёт true иначе false

bool sortByAuthor(Book firstBook, Book secondBook);

//сортировка по полю числа страниц

//входные параметры сравниваемые книги

//если число страниц первой книги меньше второй функция вернёт true иначе false

bool sortByQuantityPage(Book firstBook, Book secondBook);

//вывод данных по всем книгам

//принимает ссылку на вектор книг

void printBooks(const std::vector<Book>& books);

**procesingLibrary.cpp**

#include "procesingLibrary.h"

void searchQueryByAuthor(const std::vector<Book>& books)

{

std::string const welcomeMessage{ "Поиск книги по автору" };

std::string author{ enteryNewBookAuthor(welcomeMessage) };

std::vector<Book> booksByAuthor(0);//пустой вектор книг, для хранения результатов поиск

findBooksByAuthor(books, author, booksByAuthor);//поиск книг по автору

if (booksByAuthor.size() > 0)//если вектор booksByAuthor не пустой

{

printBooks(booksByAuthor);//вывод найденных книг

}

else

{

std::string const booksNotFound{ "Книг искомого автора " + author + " не найденно" };

std::cout << booksNotFound << std::endl;

}

}

void searchQueryByYearPuolication(const std::vector<Book>& books)

{

std::string const welcomeMessage{ "Поиск книги по году публикации" };

unsigned int yearPuplication{ enteryNewBookYearPuplication(welcomeMessage) };

std::vector<Book> booksByYearPublication(0);//пустой вектор книг, для хранения результатов поиск

findBooksByYearPuolication(books, yearPuplication, booksByYearPublication);//поиск книг по году издания

if (booksByYearPublication.size() > 0)//если вектор booksByYearPublication не пустой

{

printBooks(booksByYearPublication);//вывод найденных книг

}

else

{

std::string const booksNotFound{ "Книг " + std::to\_string(yearPuplication) + " года издания не найденно" };

std::cout << booksNotFound << std::endl;

}

}

void searchQueryByYearPuolicationMoreThen(const std::vector<Book>& books)

{

std::string const welcomeMessage{ "Поиск книг опубликованных после:" };

unsigned int yearPuplication{ enteryNewBookYearPuplication(welcomeMessage) };

std::vector<Book> booksByYearPublication(0);//пустой вектор книг, для хранения результатов поиск

findBooksByYearPuolicationMoreThen(books, yearPuplication, booksByYearPublication);//поиск книг изданных после

if (booksByYearPublication.size() > 0)//если вектор booksByYearPublication не пустой

{

printBooks(booksByYearPublication);//вывод найденных книг

}

else

{

std::string const booksNotFound{ "Книг изданных после " + std::to\_string(yearPuplication) + " года не найденно" };

std::cout << booksNotFound << std::endl;

}

}

void searchQueryByPublishingHouse(const std::vector<Book>& books)

{

std::string const welcomeMessage{ "Поиск книги по издателю" };

std::string publishingHouse{ enteryNewBookPublishingHouse(welcomeMessage) };

std::vector<Book> booksByPublishingHouse(0);//пустой вектор книг, для хранения результатов поиск

findBooksByPublishingHouse(books, publishingHouse, booksByPublishingHouse);//поиск книг по издательству

if (booksByPublishingHouse.size() > 0)//если вектор booksByPublishingHouse не пустой

{

printBooks(booksByPublishingHouse);//вывод найденных книг

}

else

{

std::string const booksNotFound{ "Книг издательства " + publishingHouse + " не найденно" };

std::cout << booksNotFound << std::endl;

}

}

void searchQueryBooksOnIssue(const std::vector<Book>& books)

{

std::string const welcomeMessage{ "Поиск книг на выдаче" };

std::cout << welcomeMessage << std::endl;

std::vector<Book> booksOnIssue(0);//пустой вектор книг, для хранения результатов поиск

findBooksOnIssue(books, booksOnIssue);//поиск книг на выдаче

if (booksOnIssue.size() > 0)//если вектор booksOnIssue не пустой

{

printBooks(booksOnIssue);//вывод найденных книг

}

else

{

std::string const booksNotFound{ "Книг на выдаче нет" };

std::cout << booksNotFound << std::endl;

}

}

void findBooksByAuthor(const std::vector<Book>& books, std::string const author, std::vector<Book>& outBooks)

{

for (auto book : books)

{

if (book.author == author)

outBooks.push\_back(book);//добавление книги в выходной вектор

}

}

void findBooksByYearPuolication(const std::vector<Book>& books, unsigned int const yearPuolication, std::vector<Book>& outBooks)

{

for (auto book : books)

{

if (book.yearPuplication == yearPuolication)

outBooks.push\_back(book);//добавление книги в выходной вектор

}

}

void findBooksByYearPuolicationMoreThen(const std::vector<Book>& books, unsigned int const minYearPuolication, std::vector<Book>& outBooks)

{

for (auto book : books)

{

if (book.yearPuplication > minYearPuolication)

outBooks.push\_back(book);//добавление книги в выходной вектор

}

}

void findBooksByPublishingHouse(const std::vector<Book>& books, std::string const publishingHouse, std::vector<Book>& outBooks)

{

for (auto book : books)

{

if (book.publishingHouse == publishingHouse)

outBooks.push\_back(book);//добавление книги в выходной вектор

}

}

void findBooksOnIssue(const std::vector<Book>& books, std::vector<Book>& outBooks)

{

for (auto book : books)

{

if (book.status == bookStatus::BOOK\_ON\_ISSUE)

outBooks.push\_back(book);//добавление книги в выходной вектор

}

}

void sortBooksByYearPublication(std::vector<Book> &books)

{

std::sort(books.begin(), books.end(), sortByYearPublication);

}

void sortBooksByAuthor(std::vector<Book>& books)

{

std::sort(books.begin(), books.end(), sortByAuthor);

}

void sortBooksByQuantityPage(std::vector<Book>& books)

{

std::sort(books.begin(), books.end(), sortByQuantityPage);

}

bool sortByYearPublication(Book firstBook, Book secondBook)

{

return firstBook.yearPuplication < secondBook.yearPuplication;

}

bool sortByAuthor(Book firstBook, Book secondBook)

{

return firstBook.author < secondBook.author;

}

bool sortByQuantityPage(Book firstBook, Book secondBook)

{

return firstBook.quantityPage < secondBook.quantityPage;

}

void printBooks(const std::vector<Book>& books)

{

std::vector<size\_t> columnSize{ 24,20,20,20,20,20,30,20 }; //размер колонок

std::vector<std::string> columnNames{ "Регистрационный номер","Автор","Название","Год публикации","Издание","Число страниц","Последний пользователь","Книга на выдачи" };//заголовки колонок

drawTableRaw(columnSize, columnNames, true);//вывод заголовка

for (size\_t i{ 0 }; i < books.size(); ++i)//вывод книг

{

std::vector<std::string> userData{};

userData.push\_back(std::to\_string(books[i].registrationNumber)); //конвертация из int в string

userData.push\_back(books[i].author);

userData.push\_back(books[i].name);

userData.push\_back(std::to\_string(books[i].yearPuplication));//конвертация из int в string

userData.push\_back(books[i].publishingHouse);

userData.push\_back(std::to\_string(books[i].quantityPage));//конвертация из int в string

userData.push\_back(std::to\_string(books[i].lastReaderTicketNumber));//конвертация из int в string

userData.push\_back(std::to\_string(static\_cast<int>(books[i].status)));//конвертация из перечисления в int, затем в string

if (i < books.size() - 1)

drawTableRaw(columnSize, userData);

else

drawTableRaw(columnSize, userData, true);//если книга последняя нарисовать нижную границу

}

}

**userPassword.h**

#pragma once

#include <iostream>

#include <string>

#include <random>

#include <conio.h>

#include "checkFunction.h"

#include "editingUsers.h"

//функция авторизации

//принимает ссылку на вектор пользователей и ссылку на нового активного пользователя

//выходными данными является ссылка на нового активного пользователя, содржашая данные автороиированного пользователя

//если авторизация был успешной функция вернёт true иначе false

bool authorization(const std::vector<User>& users, User& newActivUser);

//функция создания «соли» к паролю для пользователя

//принимает логин пользователя

//возвращает строку с "солью" для пользователя

std::string createSaltForUser(std::string const userName);

//функция генерации «соли»

//принимает требуемы размер "соли"

//возвращает строку с "солью"

std::string generateSalt(size\_t const saltLen);

//функция перевода пароля с «солью» в хеш

//принимает пароль с "солью" и логин пользователя

//возвращает хеш пароля

std::string convertUserPasswordToHash(std::string const passwordWithSalt, std::string const userName);

//функция для чтения пароля с консоли, заменяет символы на «звёздочки» \*

//возвращает строку с паролем

std::string readPassword();

//функция хеширования

//принимает значение которое надо хешировать и номер сида для функции random

//возвращает хеш

std::string hashFunction(std::string const toHash, int const randomSeed);

**userPassword.cpp**

#include "userPassword.h"

bool authorization(const std::vector<User>& users, User& newActivUser)

{

system("cls");//очистка консоли

std::string const loginMessage{ "Логин:" };

std::cout << loginMessage << std::endl;

std::string loginNewActivUser{};

\_flushall();//очистке всех входных буферов

std::getline(std::cin, loginNewActivUser);//чтение строки с наличием " "

std::string const passwordMessage{ "Пароль:" };

std::cout << passwordMessage << std::endl;

std::string passwordNewActivUser{ readPassword() };//ввод паспорта

int indexNewActivUser{ findUserIndex(users, loginNewActivUser) };//поиск индекса пользователя по логину

if (indexNewActivUser < 0) //индекс -1 пользователь не найден

{

return 0;

}

else

{

std::string userSalt{ (users)[indexNewActivUser].salt };//считывание соли к паролю

std::string inputHashPassword{ convertUserPasswordToHash(passwordNewActivUser + userSalt, loginNewActivUser) }; //ковертация введённого пароля с добавлением "соли" в хеш

std::string userHashPassword{ (users)[indexNewActivUser].saltedHashPassword }; //верный хеш пароля

if (userHashPassword == inputHashPassword)//верный хеш совподает с хэшем полученым из введённого пароля

{

newActivUser = (users)[indexNewActivUser];//определение нового активного пользователя

return 1;

}

else //пароли не совпали

{

return 0;

}

}

}

std::string createSaltForUser(std::string const userName)

{

return generateSalt(userName.size());

}

std::string generateSalt(size\_t const saltLen)

{

std::random\_device rd{};

std::mt19937 gen{ rd() }; //генерация случайного сида

double minSimvolIndex{ 33.0 };//минимальный номер сиволя в ASCII(нижняя граница генерации)

double maxSimvolIndex{ 127.0 };//максимальный номер сиволя в ASCII(верхняя граница генерации)

std::uniform\_real\_distribution<double> simvolGenerate{ minSimvolIndex, maxSimvolIndex };//равномерная генерация чисел из диапазона

std::string salt{};//хранит результат генерирования "соли"

for (size\_t i{}; i < saltLen; ++i)

{

int newSimvol{ static\_cast<int>(round(simvolGenerate(gen))) };//генерация номера символа, round округляет до целой части, после конвертируется в int

salt.push\_back(newSimvol);//добавление нового символа

}

return salt;

}

std::string convertUserPasswordToHash(std::string const passwordWithSalt, std::string const userName)

{

size\_t randomSeed{ userName.size() + passwordWithSalt.size() };//установка сида для генерации случайных чисел, на основе суммы числа элементов в логине и пароли с "солью"

return hashFunction(passwordWithSalt, randomSeed);//результат хэширования

}

std::string readPassword()

{

std::string password{};//хранит введённый пароль

for (;;)

{

char simvol{ };

simvol = \_getch();// происход считывания клавиши без её вывода в кансоль и нажатия Enter

if (simvol == '\r')//если введён Enter

return password; //вернуть результат

else if (simvol == '\b')//если нажат backspace

{

std::cout << "\b"<<" "<< "\b";//вернутся назад ввести пробел и опять вернутся назад

if(password.size()>0) //пока в пароли есть символы

password.pop\_back();//удалять последний

}

else

{

password.push\_back(simvol);//добавить введённый символ в пароль

std::cout << "\*";//вывести символ \*

}

}

}

std::string hashFunction(std::string const toHash, int const randomSeed)

{

srand(randomSeed);//устанвка сида для генерации случайных чисел

std::string hash{};//переменная для хеш

int sumSimvols{ 0 };

for (size\_t i{}; i < toHash.size(); ++i)

{

sumSimvols += toHash[i];//сумма индексов символов пароля по таблице ASCII

}

for (size\_t i{}; i < toHash.size(); ++i)//для каждого из элементов пароля

{

double denominator{ static\_cast<double>(rand()) }; //генерация знаменателя

double firstPartHashElement{ sumSimvols / denominator };//первая часть элемента

double secondPartHashElement{ toHash[i] / denominator };//вторая часть элемента

hash += std::to\_string(firstPartHashElement + secondPartHashElement);//один элемент определяется как сумма двух частей, конвертируется в string и добовляется в hash

}

return hash;

}

**workingWithLibraryFile.h**

#pragma once

#include <fstream>

#include <string>

#include <vector>

#include "Book.h"

//проверка на наличие файла с базой данных библиотечных книг

//принимает имя файла с БД

//если файл отрывается корректно функция вернёт true иначе false

bool fileWithLibraryIsOpen(std::string const fileName);

//загрузка данных по библиотечным книга из файла в вектор

//принимает имя файла с БД и ссылку на вектор библиотечных книг

void downloadLibraryData(std::string const fileName, std::vector<Book>& booksInLibrary);

//чтение данных из файла базы данных о библиотечных книгах

//принимает файла с БД и ссылку на вектор библиотечных книг

void readLibraryDataFromFile(std::ifstream& fileWithData, std::vector<Book>& booksInLibrary);

//запись из вектора в файл данных о библиотечных книгах

//принимает имя файла с БД и указатель на вектор библиотечных книг

void writeLibraryDataToFile(std::string const fileName, std::vector<Book> const\* booksInLibrary);

**workingWithLibraryFile.cpp**

#include "workingWithLibraryFile.h"

bool fileWithLibraryIsOpen(std::string const fileName)

{

std::ifstream fileWithLibraryData(fileName);//открытие файла

if (fileWithLibraryData.is\_open())//файл открыт

return true;

else

return false;

}

void downloadLibraryData(std::string const fileName, std::vector<Book>& booksInLibrary)

{

std::ifstream fileWithLibraryData(fileName);

if (fileWithLibraryIsOpen(fileName))//если файл открыт

{

readLibraryDataFromFile(fileWithLibraryData, booksInLibrary);//чтение данных из файла

}

}

void readLibraryDataFromFile(std::ifstream& fileWithData, std::vector<Book>& booksInLibrary)

{

size\_t bookNumber{};//переменная хранения номера книги

size\_t stringNumber{};//переменная хранения номера строки

for (; !fileWithData.eof();) {//пока файл не закончился

std::string bufer;//буфер для хранения строки

std::getline(fileWithData, bufer);//чтение строки

switch (stringNumber) {

case 0: {

booksInLibrary.push\_back(Book{});//добавление книги

booksInLibrary[bookNumber].registrationNumber = atoi(bufer.c\_str());//установка регистрационного номера, конвертация из string в int

++stringNumber;//увеличенияе номера строки на 1

break;

}

case 1: {

booksInLibrary[bookNumber].author = bufer;//установка автора

++stringNumber;//увеличенияе номера строки на 1

break;

}

case 2: {

booksInLibrary[bookNumber].name = bufer;//установка названия книги

++stringNumber;//увеличенияе номера строки на 1

break;

}

case 3: {

booksInLibrary[bookNumber].yearPuplication = atoi(bufer.c\_str());//установка года издания, конвертация из string в int

++stringNumber;//увеличенияе номера строки на 1

break;

}

case 4: {

booksInLibrary[bookNumber].publishingHouse = bufer;//установка издательства

++stringNumber;//увеличенияе номера строки на 1

break;

}

case 5: {

booksInLibrary[bookNumber].quantityPage = atoi(bufer.c\_str());//установка число страниц, конвертация из string в int

++stringNumber;//увеличенияе номера строки на 1

break;

}

case 6: {

booksInLibrary[bookNumber].lastReaderTicketNumber = atoi(bufer.c\_str());//установкачитательского билета, конвертация из string в int

++stringNumber;//увеличенияе номера строки на 1

break;

}

case 7: {

booksInLibrary[bookNumber].status = bookStatus(atoi(bufer.c\_str()));//установка статуса книги, конвертация из string в int

stringNumber = 0;//обнуление номера строки

++bookNumber; //увеличение номера книги на 1

break;

}

}

}

}

void writeLibraryDataToFile(std::string const fileName, std::vector<Book> const\* booksInLibrary)

{

std::ofstream outFile(fileName, std::ofstream::out | std::ofstream::trunc);//открытие файла для записи и его очистка

for (size\_t i{ }; i < booksInLibrary->size(); ++i)

{

outFile << (\*booksInLibrary)[i].registrationNumber << std::endl;

outFile << (\*booksInLibrary)[i].author << std::endl;

outFile << (\*booksInLibrary)[i].name << std::endl;

outFile << (\*booksInLibrary)[i].yearPuplication << std::endl;

outFile << (\*booksInLibrary)[i].publishingHouse << std::endl;

outFile << (\*booksInLibrary)[i].quantityPage << std::endl;

outFile << (\*booksInLibrary)[i].lastReaderTicketNumber << std::endl;

outFile << static\_cast<int>((\*booksInLibrary)[i].status);

if (i < booksInLibrary->size() - 1)

outFile << std::endl;//если книга не последняя перейти на новую строку

}

}

**workingWithUserFile.h**

#pragma once

#include <fstream>

#include <string>

#include <vector>

#include"editingUsers.h"

//загрузка данных по пользователям из файла в вектор

//принимает имя файла с БД и ссылку на вектор пользователей

void downloadUsersData(std::string const fileName, std::vector<User>& users);

//чтение данных из файла базы данных о пользователях

//принимает файла с БД и ссылку на вектор пользователей

void readUsersDataFromFile(std::ifstream& fileWithData, std::vector<User>& users);

//запись из вектора в файл данных о пользователях

//принимает имя файла с БД и указатель на вектор пользователей

void writeUserDataToFile(std::string const fileName, std::vector<User> const\* users);

**workingWithUserFile.cpp**

#include "workingWithUserFile.h"

void downloadUsersData(std::string const fileName, std::vector<User>& users)

{

std::ifstream fileWithUsersData(fileName);//открытие файла

if (fileWithUsersData.is\_open())//если файл открыт

{

readUsersDataFromFile(fileWithUsersData, users);//чтение данных из файла

}

else

{

creatFirstUser(users);//создание первого пользователя

writeUserDataToFile(fileName, &users);//запись данных первого пользователя в файл

}

}

void readUsersDataFromFile(std::ifstream& fileWithData, std::vector<User>& users)

{

size\_t userNumber{};//хранение номера пользователя

size\_t stringNumber{};//хранение номера строки

for (; !fileWithData.eof();) {

std::string bufer;//буфер для хранения считанной строки

std::getline(fileWithData, bufer);//чтение строки

switch (stringNumber) {

case 0: {

users.push\_back(User{});//добавление новго пользователя

users[userNumber].login = bufer;//установка логина

++stringNumber;//увеличение номера строки на 1

break;

}

case 1: {

users[userNumber].saltedHashPassword = bufer;//запись хэша пароля

++stringNumber;//увеличение номера строки на 1

break;

}

case 2: {

users[userNumber].salt = bufer;//запись соли

++stringNumber;//увеличение номера строки на 1

break;

}

case 3: {

users[userNumber].role = userRole(atoi(bufer.c\_str()));//установка уровня доступа, конвертация из string в int

++stringNumber;//увеличение номера строки на 1

break;

}

case 4: {

users[userNumber].access = atoi(bufer.c\_str());//установка статуса активированности, конвертация из string в int

stringNumber = 0;//обнуление номера строки

++userNumber;//увеличение номера пользователя на 1

break;

}

}

}

}

void writeUserDataToFile(std::string const fileName, std::vector<User> const\* users)

{

std::ofstream outFile(fileName, std::ofstream::out | std::ofstream::trunc);//открытие файла для записи и его очистка

for (size\_t i{ }; i < users->size(); ++i)

{

outFile << (\*users)[i].login << std::endl;

outFile << (\*users)[i].saltedHashPassword << std::endl;

outFile << (\*users)[i].salt << std::endl;

outFile << static\_cast<int>((\*users)[i].role) << std::endl;

outFile << (\*users)[i].access;

if (i < users->size() - 1)

outFile << std::endl;//если пользователь не последний перейти на новую строку

}

}