МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

КУРСОВОЙ ПРОЕКТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

ассистент		Мурашова М. А.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

Тема курсового проекта:
«ИСПОЛЬЗОВАНИЕ ЗАДАННЫХ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ: РЕГИСТРАЦИЯ БОЛЬНЫХ В ПОЛИКЛИННИКЕ»

по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	М011		Н.Ю. Горлов
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Оглавление:

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

Первичная постановка задания.

Предметная область - «Обслуживание читателей в библиотеке».

Метод хеширования – «Закрытое хеширование с двойным хешированием с квадратичным опробованием».

Метод сортировки – «Распределением».

Вид списка – «Линейный двунаправленный».

Метод обхода дерева – «Симметричный».

Алгоритм поиска слова в тексте – «Боуера и Мура».

Цель и задачи программы.

Цель курсового проектирования: изучение структур данных и алгоритмов их обработки, а также получение практических навыков их использования при разработке программ. Задача курсового проекта: разработка информационной системы для заданной предметной области с использованием заданных структур данных и алгоритмов.

Информационная система для предметной области «Обслуживание читателей в библиотеке» должна осуществлять ввод, хранение, обработку и вывод данных о:

- читателях;
- книгах;
- выдаче и приеме книг от читателей.

Данные о каждом читателе должны содержать:

- Номер читательского билете – строка формата «ANNNN-YY», где A – буква, обозначающая права доступа читателя (А – только абонемент; Ч – только читальный зал, В – читальный зал и абонемент); NNNN – порядковый номер регистрации (цифры); YY – последние две цифры номера года рождения;
- ФИО – строка;
- Год рождения – целое;
- Адрес – строка;

- Место работы (учебы) – строка.

Данные о читателях должны быть организованны в виде хеш-таблицы, первичным ключом которой является «Номер читательского билета».
Данные о каждой книге должны содержать:

- Шифр – строка формата «NNN.MMM», где NNN – номер тематического раздела (цифры); MMM – порядковый номер книги в разделе (цифры);
- Автор(ы) – строка;
- Название – строка;
- Издательство – строка;
- Год издания – целое;
- Количество экземпляров всего – целое;
- Количество экземпляров в наличии – целое.

Данные о книгах должны быть организованны в виде АВЛ-дерева поиска, упорядоченного по «Шифру».
Данные о выдаче или возврате направлений к врачу должны содержать:

- Номер читательского билета – строка, формат которой соответствует аналогичной строке в данных о читателях;
- Шифр – строка, формат которой соответствует аналогичной строке в данных о книгах;
- Дату выдачи – строка;
- Дату возврата – строка.

Примечания:

1. Наличие в этих данных записи, содержащей в поле «Номер читательского билета» значение X и в поле «Шифр» значение Y, означает выдачу читателю с номером читательского билета X экземпляра книги с шифром Y. Отсутствие такой записи означает, что читателю с номером читательского билета X не выдавался ни один экземпляр книги с шифром Y.

2. Одному читателю может быть выдано несколько книг, и экземпляры одной книги могут быть выданы нескольким читателям. Таким образом, могут быть данные, имеющие повторяющиеся значения в своих полях. Данные о выдаче или возврате направлений к врачу должны быть организованны в виде списка, который упорядочен по первичному ключу – «ФИО врача».

Данные о выдаче или приеме книг от читателей должны быть организованны в виде списка, который упорядочен по первичному ключу – «Шифр». Вид списка и метод сортировки определяются вариантом задания.

Информационная система «Обслуживание читателей в библиотеке» должна осуществлять следующие операции:

- регистрацию нового читателя;
- снятие с обслуживания читателя;
- просмотр всех зарегистрированных читателей;
- очистку данных о читателях;
- поиск читателя по номеру читательского билета. Результаты поиска – все сведения о найденном читателе и шифры книг, которые ему выданы;
- поиск читателя по ФИО. Результаты поиска – список найденных читателей с указанием номера читательского билета и ФИО;
- добавление новой книги;
- удаление сведений о книге;
- просмотр всех имеющихся книг;
- очистку данных о книгах;
- поиск книги по шифру. Результаты поиска – все сведения о найденной книге, а также номера читательских билетов и ФИО читателей, которым выданы экземпляры этой книги;
- поиск книги по фрагментам ФИО автора(ов) или названия. Результаты поиска – список найденных книг с указанием шифра, автора(ов),

названия, издательства, года издания и количества экземпляров в наличии;

- регистрацию выдачи экземпляра книги читателю;
- регистрацию приема экземпляра книги от читателя.

Поиск книги по фрагментам ФИО автора(ов) или названия должен осуществляться путем систематического обхода АВЛ дерева поиска. Метод обхода определяется вариантом задания. При поиске книги по фрагментам ФИО автора(ов) или названия могут быть заданы как полное ФИО автора(ов) или названия, так и их части (например, ФИО одного из нескольких авторов, одно слово или часть слова из названия). Для обнаружения заданного фрагмента в полном ФИО автора(ов) или названии должен применяться алгоритм поиска слова в тексте, указанный в варианте задания.

Регистрация выдачи экземпляра книги читателю должна осуществляться только при наличии свободных экземпляров выдаваемой книги (значение поля «Количество экземпляров в наличии» для соответствующей книги больше нуля).

При регистрации выдачи экземпляра книги или приема экземпляра книги от читателя должно корректироваться значение поля «Количество экземпляров в наличии» для соответствующей книги.

При снятии с обслуживания читателя должны быть учтены и обработаны ситуации, когда у читателя имеются выданные книги. Аналогичным образом следует поступать и с удалением сведений о книгах.

Сценарии использования

Предусловие.

Создается 4 структуры данных:

1. Reader, содержащая следующие поля:
   - Номер читательского билета – строка

- ФИО – строка;
- Год рождения – целое;
- Адрес – строка;
- Место работы (учебы) – строка.

2. Book, содержащая следующие поля:
   - Шифр – строка;
   - Автор(ы) – строка;
   - Название – целое;
   - Издательство – строка.
   - Год издательства – целое число.
   - Количество копий всего – целое число.
   - Количество копий в наличии.
3. Taken, содержащая следующие поля:
   - Номер читательского билета – строка;
   - Шифр – строка;
   - Дату выдачи – строка;
   - Дату приема – строка.
4. List, содержащий следующие поля:
   - Выдача и прием книги – тип Taken
   - Указатель на следующий элемент – тип List*

Примечание:

4я структура нужна для сортировки распределением

Сценарий: Запуск приложения.

1. Пользователь запускает приложение.
2. На экране отображается пронумерованный список пунктов меню и предложение ввести номер одного из пунктов.
3. Пользователь вводит номер пункта меню. Дальнейшее поведение приложения описано в последующих сценариях.

Сценарии работы консольного приложения для структуры Reader:

Сценарий: Добавление в БД нового читателя библиотеки.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается текст с предложением ввести количество новых читателей, которые будут добавлены в БД библиотеки. С новой строки предлагается ввести данное.
3. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
4. Пользователь вводит ФИО читателя и нажимает Enter.
5. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
6. Пользователь вводит год рождения читателя и нажимает Enter.
7. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
8. Пользователь вводит адрес проживания читателя и нажимает Enter.
9. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
10. Пользователь вводит место работы(учебы) читателя и нажимает Enter.
11. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
12. Пользователь вводит права доступа читателя и нажимает Enter.
13. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
14. Происходит автоматическое заполнение читательского билета.

15. Выводится сообщение о том, что данные о читателях успешно добавлены.
16. Предлагается нажать любую кнопку для перехода обратно в меню.

Сценарий демонстрации БД читатели в консоли.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД читатели. При нажатии любой кнопки, пользователю отображается главное меню.

Сценарий поиска пациента по ФИО.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД.
3. Пользователь вводит ФИО читателя и нажимает Enter.
4. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
5. В консоли выводится пронумерованный список найденных читателей, если никого найти не удалось, то выводится сообщение о не нахождении пациента.
6. Предлагается нажать любую кнопку для перехода обратно в меню.

Сценарий поиска читателя по номеру читательского билета.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД читатели.
3. Пользователь вводит номер читательского билета читателя и нажимает Enter.
4. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных. Если ввод правильный, но читатель с

введённым номером читательского билета отсутствует, то приложение выводит об этом сообщение.

5. На экран выводится вся информация о читателе и книгах, которые выдавались читателю. Если у читателя нет книг, то таблица с информацией о книгах не заполнятся.
6. Предлагается нажать любую кнопку для перехода обратно в меню.

Сценарий удаление читателя по ФИО.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД.
3. Пользователь вводит ФИО читателя и нажимает Enter.
4. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
5. Если читатель имеет в наличии хотя бы одну книгу, то удаление невозможно, об этом выводится сообщение в консоли. Далее предлагается выйти в главное меню.
6. Если в БД хранится только один схожий читатель с введенным ранее, то приложение выводит его в консоль, далее удаляет. Потом пользователю предлагают выйти в главное меню.
7. Если в БД хранится несколько читателей с похожим ФИО, то приложение выводит в консоль список предполагаемых для удаления читателей и предлагает удалить одного из них выбрав порядковый номер.
8. Пользователь вводит порядковый номер и нажимает Enter.
9. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
10. Приложение удаляет читателя напротив выбранного порядкового номера и предлагает выйти в главное меню.

11. Если в БД нет введенного читателя, то приложение выводит об этом сообщение и предлагает выйти в главное меню.

Сценарий очистки БД читатели.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. Приложение выводит сообщение о полной очистке БД читатели.
3. Если хотя бы один читатель имеет в наличии книгу, то удаление невозможно, об этом выводится сообщение в консоли. Далее предлагается выйти в главное меню.

Сценарий: Сохранения БД читатели в файл формата txt.

1. После завершения работы программы осуществляется проверка формата БД читатели.
2. БД читатели сохраняется в файл.
3. Если БД читатели повреждена, то об этом выводиться сообщение.

Сценарий: Загрузка БД читатели из файла формата txt.

1. Запускается программа и автоматически осуществляется проверка формата БД читатели.
2. БД читатели загружается из файла.
3. Если БД читатели повреждена, то об этом выводиться сообщение.

Сценарии работы консольного приложения для структуры Book:

Сценарий: Добавление в БД библиотеки новой книги.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается текст с предложением ввести количество новых книг, которые будут добавлены в БД библиотеки. С новой строки предлагается ввести данное.
3. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
4. Пользователь вводит ФИО автора и нажимает Enter.

5. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
6. Пользователь вводит название книги и нажимает Enter.
7. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
8. Пользователь вводит издательство книги и нажимает Enter.
9. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
10. Пользователь вводит год публикации книги и нажимает Enter.
11. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
12. Пользователь вводит количество добавляемых экземпляров
13. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
14. Пользователь вводит жанр книги.
15. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
16. Выводится сообщение о том, что данные о книгах добавлены.
17. Предлагается нажать любую кнопку для перехода обратно в меню.

Сценарий демонстрации БД книг в консоли.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.

2. На экране отображается список содержимого БД. При нажатии любой кнопки, пользователю отображается главное меню.

Сценарий поиска книги по шифру.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД.
3. Пользователь вводит шифр книги и нажимает Enter.
4. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
5. Если в БД не удалось найти книгу, то поиск отменяется и приложение предлагает выйти в главное меню.
6. Если удалось найти книгу и есть выданные экземпляры данной книги, то в консоли выводится заполненные таблицы с информацией о книге и читателях, которым выданы экземпляры данной книги.
7. Если в БД удалось найти книгу, но ни один экземпляр данной книги не выдан, то выводится таблица с информацией только о книге, далее предлагается выйти в главное меню.

Сценарий поиска книги по фрагментам ФИО автора или названия.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД.
3. Пользователь вводит фрагмент ФИО автора или названия и нажимает Enter.
4. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
5. Если удалось найти книгу по фрагменту ФИО автора или названия, то в консоли выводится заполненная таблица с информацией о найденной книге.
6. Если не удалось найти книгу по фрагменту ФИО автора или названия, то в консоли выводится пустая таблица, далее предлагается выйти в главное меню

Сценарий удаление книги по шифру.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД.

3. Пользователь вводит шифр книги и нажимает Enter.
4. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
5. Если в БД удается найти книгу с введенным ранее шифром и все экземпляры данной книги находятся в библиотеке, то приложение выводит сообщение о ее удалении. Потом пользователю предлагают выйти на начальное меню.
6. Если в БД удается найти книгу с введенным ранее шифром, но не все экземпляры данной книги находятся в библиотеке, то приложение выводит сообщение о невозможности удаления. Потом пользователю предлагают выйти на начальное меню.
7. Если в БД нет введенной книги, то приложение выводит об этом сообщение и предлагает выйти на начальное меню.

Сценарий очистки БД книги.

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. Если есть хотя бы один не возвращенный в библиотеку экземпляр книги, то очистка БД отменяется об этом выводится сообщение, иначе происходит полная отчистка БД библиотеки.

Сценарий: Сохранения БД книги в файл формата txt.

1. После завершения работы программы осуществляется проверка формата БД книги.
2. БД книги сохраняется в файл.
3. Если БД книги повреждена, то об этом выводиться сообщение.

Сценарий: Загрузка БД книги из файла формата txt.

1. Запускается программа и автоматически осуществляется проверка формата БД книги.
2. БД книги загружается из файла.

3. Если БД книги повреждена, то об этом выводиться сообщение.

Сценарии работы консольного приложения для структуры Taken:

Сценарий: Выдача книги

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД читатели.
3. На экране отображается список содержимого БД книги.
4. Пользователю предлагается ввести номер читательского билета.
5. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
6. Если веденный номер читательского билета отсутствует, то выдача направления отменяется и приложение предлагает выйти в главное меню.
7. Если введенный номер читательского билета есть в БД, то приложение предлагает ввести шифр книги.
8. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
9. Если введенный шифр книги отсутствует, то выдача книги отменяется и приложение предлагает выйти в главное меню.
10. Если введенный шифр книги, но нет ее экземпляров в БД, то приложение выводит сообщение об отсутствии книги с данным шифром и предлагает вернуться в главное меню.
11. Если введенный шифр книги и ее экземпляры есть в БД, то приложение предлагает ввести дату выдачи.

12. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
13. Если книга с таким шифром уже есть у читателя, то выдача книги отменяется и приложение предлагает выйти в главное меню.
14. Выводится сообщение о выданном направлении, далее предлагается выйти в главное меню.

Сценарий: Просмотр истории приема и выдачи книг

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД. При нажатии любой кнопки, пользователю отображается главное меню.

Сценарий: Возврат книги

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню.
2. На экране отображается список содержимого БД читатели.
3. На экране отображается список содержимого БД книги.
4. Пользователю предлагается ввести номер читательского билета.
5. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
6. Если веденный номер читательского билета отсутствует, то выдача книги отменяется и приложение предлагает выйти в главное меню.
7. Если введенный номер читательского билета есть в БД, то приложение предлагает ввести шифр книги.
8. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.

9. Если введенный шифр книги в БД отсутствует, то возврат отменяется и приложение предлагает выйти в главное меню.
10. Если введенный шифр книги есть в БД, то приложение предлагает ввести дату приема.
11. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
12. Выводится сообщение об успешном приёме книги, далее предлагается выйти в главное меню.

Сценарий: Выход из приложения

1. Пользователь запускает приложение и вводит соответствующий номер пункта меню. Приложение заканчивает работу.

ОПИСАНИЕ СТРУКТУР ДАННЫХ

1. Описание структуры данных для хранения информации о Reader.

```
string library_card_number{};
string FIO{};
string adress{};
string workplace{};
int Year_of_birth{};
static int id;
```

Структура Reader содержит следующие поля:

- library_card_number – строка формата «ANNNN-YY», где A – буква, обозначающая права доступа читателя (А – только абонемент; Ч – только читальный зал, В – читальный зал и абонемент); NNNN – порядковый номер регистрации (цифры); YY – последние две цифры номера года рождения;
- FIO – ФИО читателя, строка;
- Year_of_birth - Год рождения, целое;
- Address - Адрес, строка;
- Place_of_Work_or_Study - Место работы (учебы), строка.
- id – id идентификатор необходимый для составления номера читательского билета, статическая целая перменная.

База данных Reader хранится на диске в текстовом файле формата txt.

2. Описание структуры данных для хранения информации о Book.

```
string cipher{};
string aouthor{};
string title{};
string publishing{};
int year_of_publishing{};
int amount_of_copies{};
int copies_available{};
static int id_detective;
static int id_fantasy;
static int id_drama;
```

Структура Doctor содержит следующие поля:

- cipher – шифр книги, строка;
- aouthor – фио автора, строка;
- title – название книги, строка;

- publishing - издателество, строка;
- year_of_publishing – год издательства, целое;
- amount_of_copies – количество экземпляров книги в библиотеке всего, целое;
- copies_available – количество экземпляров книги в наличии в библиотеке, целое;
- id_detective – id идентификатор для формирования шифра книги, целое;
- id_fantasy – id идентификатор для формирования шифра книги, целое;
- id_drama – id идентификатор для формирования шифра книги, целое.

База данных Book хранится на диске в текстовом файле формата txt.

3. Описание структуры данных для хранения информации о Taken.

```
struct Taken
{
    string library_card{};
    string book_cipher{};
    string date_when_taken{};
    string date_when_returned{};
    Taken* next{};
    Taken* back{};
};
```

Структура Taken содержит следующие поля:

- Library_card – номер читательского билета, строка;
- book_cipher – шифр книги, строка;
- date_when_taken – дата выдачи книги, строка;
- date_when_returned – дата приема книги, строка.

База данных Taken хранится на диске в текстовом файле формата txt.

4. Описание данных, запрашиваемых у пользователя.

Разрабатываемая программа предполагает текстовый интерфейс взаимодействия с пользователем. Введенные пользователем данные проходят проверки в соответствии с таблицей 2.1.

Таблица 2.1- Описание данных, вводимых пользователем

Наименование переменной	Тип данных	Семантика	Описание проводимых проверок
Menu	int	Основное меню	Натуральное число (от 1 до 17)
Структура Reader			
Library_card	string	Регистрационный номер	строка формата «ANNNN-YY», где A – буква, обозначающая права доступа читателя (А – только абонемент; Ч – только читальный зал, В – читальный зал и абонемент); NNNN – порядковый номер регистрации (цифры); YY – последние две цифры номера года рождения;
FIO	string	ФИО	Буквы русского алфавита, допускается использование знаков(«.», «-»)
Adress	string	Адрес	Буквы русского алфавита(не больше 50 символов),цифры, допускается использование знаков(«.», «-», «/»)
workplace	string	Место работы(учебы)	Буквы русского алфавита(не больше 50 символов),цифры, допускается использование знаков(«.», «-», «/»)

Year_of_birth	int	Год рождения	Натуральные числа в диапазоне [1940;2015]
Структура Book			
cipher	string	Шифр	строка формата «NNN.MMM», где NNN – номер тематического раздела (цифры); MMM – порядковый номер книги в разделе (цифры);
aouthor	string	ФИО автора	Буквы русского алфавита, допускается использование знаков(«.», «-»)
title	string	Название книги	Буквы русского алфавита, допускается использование знаков(«.», «-»)
publishing	string	Издательство	Буквы русского алфавита, допускается использование знаков(«.», «-»)
year_of_publishing	int	Год публикации	Натуральные числа в диапазоне [1800;2022]
Amount_of_copies	int	Количество добавляемых экземпляров	Натуральные числа
Структура Taken			
Library_card	string	Номер читательского билета	строка формата «ANNNN-YY», где A – буква, обозначающая права доступа читателя (A – только абонемент;

			Ч – только читальный зал, В – читальный зал и абонемент); NNNN – порядковый номер регистрации (цифры); YY – последние две цифры номера года рождения;
cipher	string	Шифр	строка формата «NNN.MMM», где NNN – номер тематического раздела (цифры); MMM – порядковый номер книги в разделе (цифры);
date_when_taken	string	Дата выдачи	Дата от 01.01.2000 до 31.12.2022
date_when_returned	string	Дата приема	Дата от 01.01.2000 до 31.12.2022

ОПИСАНИЕ АЛГОРИТМА И ФУНКЦИЙ

В таблице 3.1 приведен список всех разработанных для реализации программы функций и процедур.

Таблица 3.1 – Перечень разработанных функций и процедур

Прототип	**Входные параметры**	**Выходные параметры**	**Описание функционала**
void Menu()	-	-	Вывод меню
Функции внутри структуры Reader			
Reader()	-	-	Установка начальных значений
void SetInfo(string li, string fio, string ad, string work, int year)	Данные находящиеся в структуре читатели	-	Присвоение начальным значениям новые значения
Reader& operator = (const Reader& other)	Константная ссылка на структуру Reader	Ссылку на структуру Reader	Перегрузка оператора «=»
void SetZeroes()	-	-	Установка нулевых значений
Функции внутри структуры Doctor			
Book()	-	-	Установка начальных значений
void SetInfo(string ciphe, string aoutho, string titl, string publishin, int year_of_publishin, int amount_of_copie)	Данные находящиеся в структуре книги	-	Присвоение начальным значениям новые значения
void SetInfo(Book& other)	Ссылка на структуру Book	-	Присвоение начальным значениям новые значения
Book& operator = (const Book& other)	Константная ссылка на структуру Book	Ссылку на структуру Book	Перегрузка оператора «=»
void SetZeroes()	-	-	Установка нулевых значений
Функции внутри структуры Taken			
Taken()	-	-	Установка начальных значений
Taken(string card, string cipher, string date_taken, string date_returned)	Данные находящиеся в структуре история приема и выдачи книг	-	Присвоение начальным значениям новые значения

Taken& operator = (const Taken& other)	Константная ссылка на структуру Taken	Ссылку на структуру Taken	Перегрузка оператора «=»
Внешние функции			
int ChekInt(string text)	text - текст запрашивающий у пользователя ввести данное	Введенное данное	Проверка ввода на: 1) целочисленный формат. 2) целочисленный формат и диапазон вводимого данного
void ChekName(string& name, string text)	Name – ссылка на строковое данное ФИО text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод ФИО (допуск символов: «.», «-», первая буква должна быть заглавной)
void ChekAdress(string& ad, string text)	ad – ссылка на строковое данное адреса text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод адреса читателя. (строка до 50 символов, допуск символов: «.», «/», «-»)
string ChekDate()	-	Дата в строковом формате	Формирование даты строковом формате.
void FormLibraryCard(string& card, int birth)	card - ссылка на строковое данное регистрационного номера	-	Заполнение регистрационного номера
void ChekCard(string& card, string text)	card - ссылка на строковое данное регистрационного номера text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод регистрационного номера
void ChekCipher(string& cipher, string text)	cipher - ссылка на строковое данное шифра книги text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод шифра книги
void InfoAboutReader(Reader& new_reader)	new_reader – ссылка на ячейку структуры с данными читателя	-	Заполнение информации о читателе

int HashFunction(string key)	key - Строковый регистрационный номер передающийся для хеширование	Целое число	Возвращает адрес в хеш-таблице
bool FillHashSpreadsheet(Reader* array_of_readers, Reader& new_reader, int hashed_key)	array_of_readers -Указатель на хеш-таблицу new_reader - Ссылка на ячейку с данными о пациенте hashed_key – хешированный ключ	true/false	Показывает удалось ли поместить читателя в хэш-таблицу или нет
void ShowSpreadsheet(Reader* spreadsheet_of_readers)	spreadsheet_of_readers – указатель на массив данных с пациентами	-	Демонстрация хеш-таблицы
void FindReaderByFio(Reader* spreadsheet_of_readers)	spreadsheet_of_readers – Указатель на массив данных с пациентами	-	Поиск читателя по ФИО
int FindReaderByCard(Reader* spreadsheet_of_readers, string card)	spreadsheet_of_readers – Указатель на хеш-таблицу card – строковый регистрационный номер	Адрес в хэш-таблицы в случае удачного поиска или размер хеш-таблицы в случае неудачного	Поиск читателя по регистрационному номеру
bool CanDeleteReader(Taken* start, string card)	Start – указатель на начало списка истории регистраций card – строковое данное регистрационного номера	Возвращает правду или ложь	Показывает можно ли удалить читателя из БД библиотеки
Void RemoveWholeInfoAboutReaders(Reader* spreadsheet_of_readers, Taken* start)	Spreadsheet_of_readers – указатель на хэш-таблицу Start – указатель на начало списка истории регистраций	-	Очищает полностью базу данных о читателях
void DeleteReader(Reader* spreadsheet_of_readers, Taken* start)	Spreadsheet_of_readers – указатель на хеш-таблицу Start – указатель на начало списка истории регистраций	-	Удаление читателя по ФИО
void WriteReadersInFile(Reader* spreadsheet_of_readers)	spreadsheet_of_readers – Указатель на массив данных с читателями	-	Запись БД читателей в файл

void ReadersFromFile(Reader* spreadsheet_of_readers)	spreadsheet_of_readers – указатель на массив данных с читателями	-	Загрузка БД читателей из файла
void InfoAboutBook(Book& new_book)	new_book - Ссылка на ячейку с данными о книге	-	Заполнение информации о книге
int height(Doctor* p)	P – указатель на узел дерева	Целое число	Высота поддерева находится
int bfactor(Doctor* p)	P – указатель на узел дерева	Целое число (от -2 до 2)	Нахождение разности высот левого и правого поддеревьев
void fixheight(Doctor* p)	P – указатель на узел дерева	-	Восстановление корректного поля высоты заданного узла
Doctor* rotate_right(Doctor* p)	P – указатель на узел дерева	указатель на узел дерева	Правый поворот дерева
Doctor* rotate_left(Doctor* q)	q – указатель на узел дерева	Указатель на узел дерева	Левый поворот дерева
Doctor* balance(Doctor* p)	P – указатель на узел дерева	Указатель на узел дерева	Балансировка дерева
Book* Insert(Book* p, Book& new_book)	P – указатель на узел дерева New_book – ссылка на ячейку памяти с данными о новой книге	Указатель на узел для добавления	Добавление книги в структуру дерева
Book* FindMinCipher(Book* p)	P – указатель на узел дерева	Указатель на узел с минимальным данным по которому упорядоченно дерево	Нахождение узла с минимальным ключом в дереве
Book* ClearMinCipher(Book* p)	P – указатель на узел дерева	Указатель на самый правый лист	Удаление узла с минимальным ключом в дереве
Book* RemoveBookFromTree(Book* p, string cipher, Taken* start)	P – указатель на узел дерева cipher – шифр книги, start – указатель на начала списка регистраций приема и выдачи книг	Указатель на узел или в никуда	Удаление книги по шифру

bool CanDeleteBook(Taken* start, string cipher)	start – указатель на начала списка регистраций приема и выдачи книг. Cipher – шифр книги	True/false	Показывает, можно ли удалить книгу с заданным шифром
Book* FindBookByCipher(Book* tmp, string cipher)	Tmp – указатель на узел дерева, cipher – шифр книги	-	Возвращает указатель на узел с книгой с заданным шифром
void ClearWholeTree(Book* p)	P – указатель на узел дерева	-	Очистка БД книг
void TreePrint(Book* p, int& number)	P – указатель на узел дерева Number – ссылка на int для вывода таблицы	-	Демонстрация БД книг в виде таблицы
void TreePrintBeatiful(Book* p)	P – указатель на узел дерева	-	Демонстрация БД книг в виде АВЛ дерева
void TreeInFile(Book* p, ofstream& fout)	P – указатель на узел дерева Fout – ссылка на ofstream для записи в файл	-	Запись БД книг в файл
void BooksToFile(Book* p)	P – указатель на узел дерева	-	Проверка открытия файла для записи в него БД книг
void BooksOutFile(Book*& p)	P – ссылка на указатель на узел дерева	-	Загрузка БД врачи из файла
string ChekDate()	-	дата	Проверка даты
void FindBookByAouthorOrTitle(Book* p, string substring, int chose)	P – ссылка на указатель на узел дерева Substrig – подстрока для поиска в тексте Choose – выбор как искать	-	Вывод информации о всех книгах в названии которых (в ФИО авторов) есть заданная подстрока.
void BookIssuance(Taken*& start, Taken*& end, Reader* spreadsheet_of_readers, Book* p)	Start – ссылка на начало списка End – ссылка на указатель на конец списка spreadsheet_of_readers – указатель массива читателей P – указатель на корень дерева	-	Выдача книги
void ShowRegistrationHistory(Taken* start)	Start – ссылка на начало списка	-	Вывод истории регистраций приема и выдачи книг

void FindBooksInRegistrationHistory(Taken * start, string card)	Start - указатель на начало списка Card – номер читательского билета	-	Вывод всех книг, находящихся у читателя на руках
void FindReaderInRegistrationHistory(Taken* start, Reader* spreadsheet_of_readers, string cipher)	Start - указатель на начало списка spreadsheet_of_readers – указатель массива читателей cipher – шифр книги	-	Вывод всех читателей, кому выдана данная книга
void BookReception(Taken* start, Reader* spreadsheet_of_readers, Book* p)	Start – ссылка на начало списка spreadsheet_of_readers – указатель на хеш-таблицу p – указатель на корень дерева	-	Прием книги
void RegistrationHistoryOutFile(Taken*& start, Taken*& end)	Start – ссылка на указатель на начало списка регистраций end – ссылка на указатель на конец списка регистраций	-	Загрузка истории регистраций приема и выдачи книг из файла
void RegistrationHistoryInFile(Taken* start)	Start – указатель на начало списка направлений	-	Сохранение истории регистраций приема и выдачи книг в файл
void ClearWholeRegistrationHistory(Taken *& start, Taken*& end)	Start – ссылка на указатель на начало списка регистраций end – ссылка на указатель на конец списка регистраций	-	Отчистка истории регистраций выдачи и приема книг
void AddToPocket(list** pocket, Taken& data, int digit)	Pocket – массив указателей на карманы по разрядам Data – ссылка на элемент из истории регистраций Digit - разряд	-	Добавление элемента в карма по разряду
void Sort(Taken* arr, int amount_of_array)	Arr – массив истории регистраций Amount_of_arrray – количество регистраций		Сортировка распределением

void DistributionSort(Taken*& start, Taken*& end)	Start – ссылка на указатель на начало списка регистраций end – ссылка на указатель на конец списка регистраций	-	Подготовительная функция для сортировки распределением

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Для проверки корректности работы приложения были использованы тестовые данные, приведенные в таблице 4.1.

Таблица 4.1 – Тестовые данные

№	Название этапа тестирования	Тестовые данные	Ожидаемый результат
1	**Главное меню**	-	Вывод на консоль окна меню
	Некорректный ввод пункта меню	4 f	Некорректный ввод, пожалуйста повторите
		4.5	Некорректный ввод, пожалуйста повторите
		Ав 4	Некорректный ввод, пожалуйста повторите
		18	Некорректный ввод, пожалуйста повторите
	Корректный ввод пункта меню	17	Завершение программы
2	**Регистрация читателя**	Пункт 1	
	Некорректный ввод количества больных для регистрации	4 а	Некорректный ввод, пожалуйста повторите
		4.6	Некорректный ввод, пожалуйста повторите
		в 3	Некорректный ввод, пожалуйста повторите
	Корректный ввод количества больных для регистрации	1	Переход к следующему шагу
	Некорректный ввод ФИО	чебаков	Некорректный ввод, пожалуйста повторите
		Чебаков виктор	Некорректный ввод, пожалуйста повторите
		Ч4баков	Некорректный ввод, пожалуйста повторите
		Dkgddf	Некорректный ввод, пожалуйста повторите
		-Чебаков	Некорректный ввод, пожалуйста повторите
	Некорректный ввод года рождения	1920	Некорректный ввод, пожалуйста повторите
		2023	Некорректный ввод, пожалуйста повторите

	Некорректный ввод адреса проживания (функция проверки адреса отличается от функции проверки ФИО только количеством допустимого ввода символов)	Более 50 символов	Некорректный ввод, пожалуйста повторите
	Некорректный ввод адреса проживания (функция проверки адреса отличается от функции проверки ФИО только количеством допустимого ввода символов)	Менее 2 символов	Некорректный ввод, пожалуйста повторите
	Некорректный ввод места работы (учебы) (функция проверки места работы ничем не отличается от функции проверки адреса)	-	Некорректный ввод, пожалуйста повторите
	Некорректный ввод прав доступа читателя	0	Некорректный ввод, пожалуйста повторите
		4 а	Некорректный ввод, пожалуйста повторите
	Корректная регистрация читателя	Чебаков Виктор Михайлович 2002 Санкт-петербург Мавритания Права доступа: 2	Данные о новых читателях успешно добавлены
3	**Просмотр списка зарегистрированных читателей**	Пункт 2	Вывод таблицы с информацией о зарегистрированных читателях
4	**Поиск читателя по ФИО**	Пункт 3	
	Ввод ФИО, которого нет в БД	Кузнецов Андрей Анатольевич	Список найденных читателей: Пуст
	Ввод ФИО, которое есть в БД	Чернышенко Юрий Анатольевич	Вывод информации о найденном читателе
5	**Поиск читателя по номеру читательского билета**	Пункт 4	Вывод таблицы с информацией о зарегистрированных читателей, а также книг, выданных им.
	Ввод номера читательского билета, которого нет в БД	Ч0011-87	Читатель с данным номером не найден
	Некорректный ввод регистрационного номера	А005-223	Некорректный ввод, пожалуйста повторите
		Г0001-56	Некорректный ввод, пожалуйста повторите
		А0001-56 авы	Некорректный ввод, пожалуйста повторите
	Ввод регистрационного номера, который есть в БД	А0001-56	Вывод информации о найденном читателе и книгах, выданных читателю
6	**Удаление читателя по фио**	Пункт 5	-

	Ввод ФИО, которое нет в БД	Кампотов Саид Мамедович	Читателя с таким ФИО нет в базе данных
	Ввод ФИО, которое есть в БД	Пивнов Артем владимирович	Происходит удаление Выводиться сообщение: Данные о читателе успешно удалены
	Ввод ФИО, которое повторяется в БД	Багов Гога Бумович	Выводятся найденные читатели Предлагается выбрать пункт для удаления читателя Выводиться сообщение: Данные о читателе успешно удалены
	Некорректный ввод пункта напротив читателя которого хотим удалить	3	Некорректный ввод, пожалуйста повторите
	Ввод ФИО читателя, которому выданы книги	Рябцева Алина Алексеевна	Невозможно удалить данные о читателе. У читателя есть невозвращенные книги. Удаление отменено.
7	**Очистка базы данных книг**	Пункт 6	База данных библотеки полностью очищена
	Если есть хотя бы одна выданная книга	Пункт 6	Базу данных книг невозможно отчистить, так как некоторые книги не возвращены в библиотеку
8	**Добавление новой книги**	Пункт 7	
	Некорректный ввод количества книг для регистрации (используется та же функция проверки, что и во втором пункте)	-	-
	Корректный ввод количества врачей для добавления	1	Переход к следующему шагу
	Некорректный ввод ФИО автора (используется та же функция проверки, что и во втором пункте)	-	-
	Некорректный ввод названия книги (функция проверки названия ничем не отличается от функции проверки адреса во втором пункте)	-	-
	Некорректный ввод издательства книги (функция проверки издательства ничем не отличается от функции проверки адреса во втором пункте)	-	-
	Некорректный ввод года издательства книги (функция проверки года издательства работает также как во втором пункте проверка возраста читателя, только в диапозоне от 1800 до 2022)	-	-

	Некорректный ввод количества новых экземпляров	-1	Некорректный ввод, пожалуйста повторите
		0	Некорректный ввод, пожалуйста повторите
		4 а	Некорректный ввод, пожалуйста повторите
	Некорректноый ввод жанра книги (функция ничем не отличается от той, что проверяет права доступа читателя во втором пункте)	-	-
	Корректное добавление книги	Некрасов Николай Павлович Стрельцы Графство 5 2	Данные о новых книгах успешно добавлены
9	**Просмотр списка книг**	Пункт 8	Вывод БД данных книг в виде таблицы и в виде АВЛ-дерева
10	**Поиска книги по шифру**	Пункт 9	Вывод БД книг
	Некорректный ввод шифра формата NNN.MMM, где NNN и MMM цифры	1122.333	Некорректный ввод, пожалуйста повторите
		111-444	Некорректный ввод, пожалуйста повторите
		-122.000	Некорректный ввод, пожалуйста повторите
		111.000 11	Некорректный ввод, пожалуйста повторите
	Ввод шифра книги, которой нет БД библиотеки	Агутин НН	Доктор с веденным ФИО отсутствует. Поиск по ФИО отменен.
	Корректный ввод шифра книги	333.001	Вывод информации о книге и всех читателях, кому данная книга выдана.
11	**Поиск книги по фрагментам фио автора или названия**	Пункт 10	Вывод информации о книгах
	Некорректный ввод фрагмента ФИО автора или названия (функция проверки фрагмента должности ничем не отличается функции проверки ФИО во 2 пункте)	-	-
	Ввод фрагмента которого нет в БД	эра	Вывод таблицы с найденными книгами (в данном случае пустой таблицы)
	Ввод фрагмента который есть в БД	евич	Вывод таблицы с найденными книгами
12	**Удаление сведений о книге**	Пункт 11	Ввод шифра формата NNN.MMM
	Проверка на ввод шифра ничем не отличается от проверки на ввод шифра в пункте 10	-	-

	Ввод шифра, которого нет БД	333.009	Книги с заданным шифром нет в базе данных библиотеке
	Корректный шифра	333.002	Данные о докторе успешно удалены.
	Ввод шифра книги, экземпялры которой находяться на руках у читателей	333.001	Невозможно удалить данные о книге. Есть экземпляры книги, не возвращенные в билиотеку. Удаление отменено
13	**Очистка базы данных книг**	Пункт 12	База данных бибилотеки полностью отчищена
	Если есть хотя бы одна выданная книга	Пункт 12	Базу данных книг невозможно отчистить, так как некоторые книги не возвращены в библиотеку
14	**Выдача книги читателю**	Пункт 13	Вывод БД книг и читателей
	Ввод номера читательского билета (осуществляются все те же проверки что и в пункте 5)	-	-
	Ввод шифра книги (осуществляются все те же проверки что и в пункте 6)	-	-
	Некорректный ввод дня выдачи	0	Некорректный ввод, пожалуйста повторите
		32	Некорректный ввод, пожалуйста повторите
	Некорректный ввод месяца выдачи	0	Некорректный ввод, пожалуйста повторите
		15	Некорректный ввод, пожалуйста повторите
	Некорректный ввод года выдачи	2025	Некорректный ввод, пожалуйста повторите
	Корректная выдача книги	A0003-90 333.003 15 6 2018	Книга выдана
15	**Прием книги от читателя**	Пункт 15	Вывод БД книг и читателей
	Ввод номера читательского билета (осуществляются все те же проверки что и в пункте 5)	A0003-90	Вывод книг, выданных читателю с данным номером читательского билета
	Ввод шифра книги (осуществляются все те же проверки что и в пункте 6) , которой нет у читателя	333.001	Вывод сообщения, что у читателя нет данной книги
	Ввод шифра книги (осуществляются все те же проверки что и в пункте 6) , которая есть у читателя	333.002	Ввод даты возвращения
	Ввод даты приема книги (осуществляются все те же проверки что и в пункте 14)	-	-
	Некорректный ввод, дата приема < дата выдачи	13 4 2015	Вывод сообщения, что нельзя вернуть книгу раньше чем ее получить

	Корректный возврат направления	05-000006 Казакова ПП 1 5 2022	Направление успешно удалено.
16	**Сортировка истории регистраций по шифру книг**	Пункт 15	Сортировка истории регистраций
17	**Просмотр истории регистрации приема и выдачи книг**	Пункт 16	Вывод истории регистраций выдачи и приема книг

Результаты тестирования приложения по данным таблицы 4.1 приведены ниже (красными линиями подчеркнуты корректные данные).

1. Главное меню:

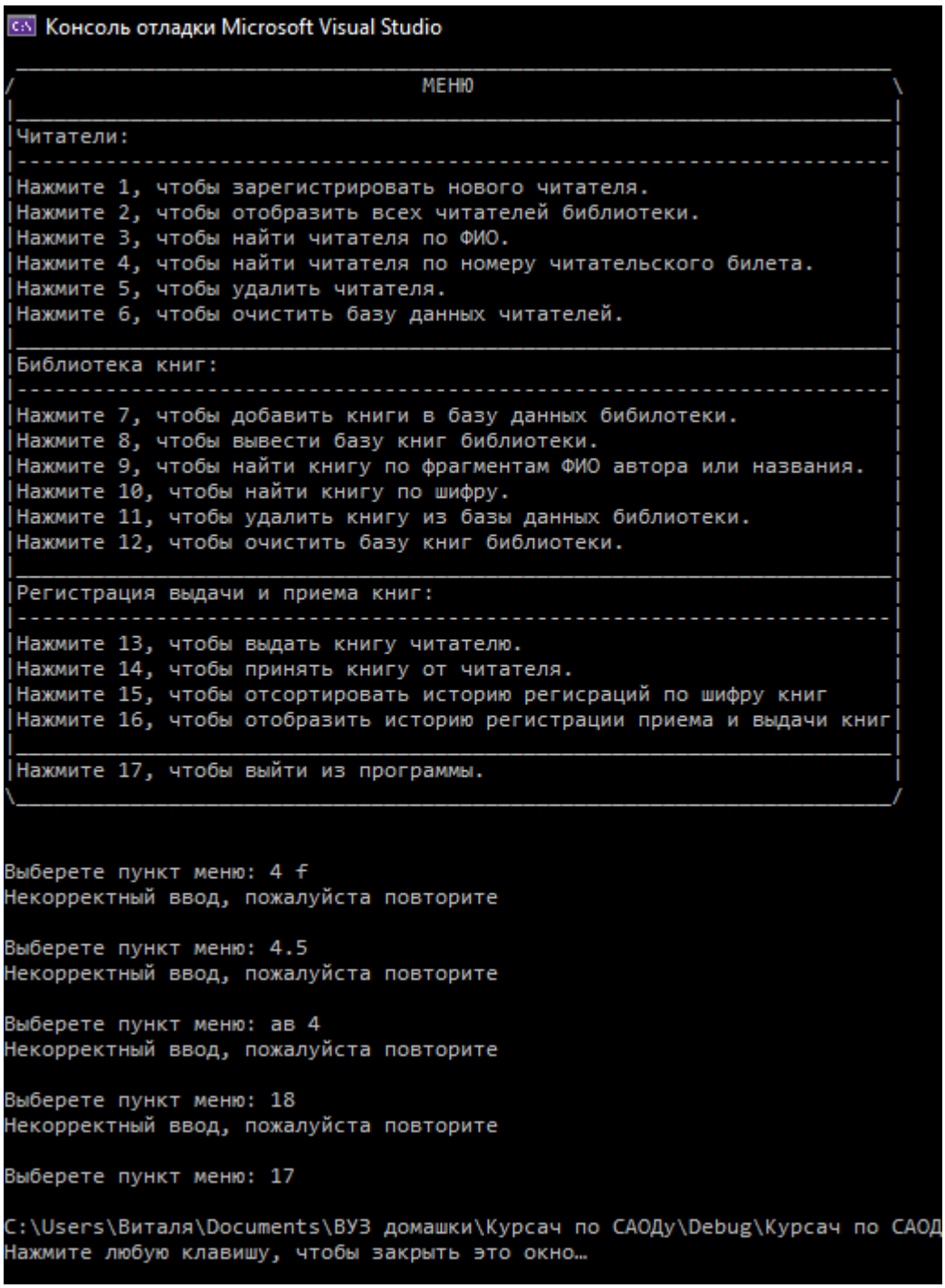


Рисунок 1 – Главное меню

2. Регистрация читателя:

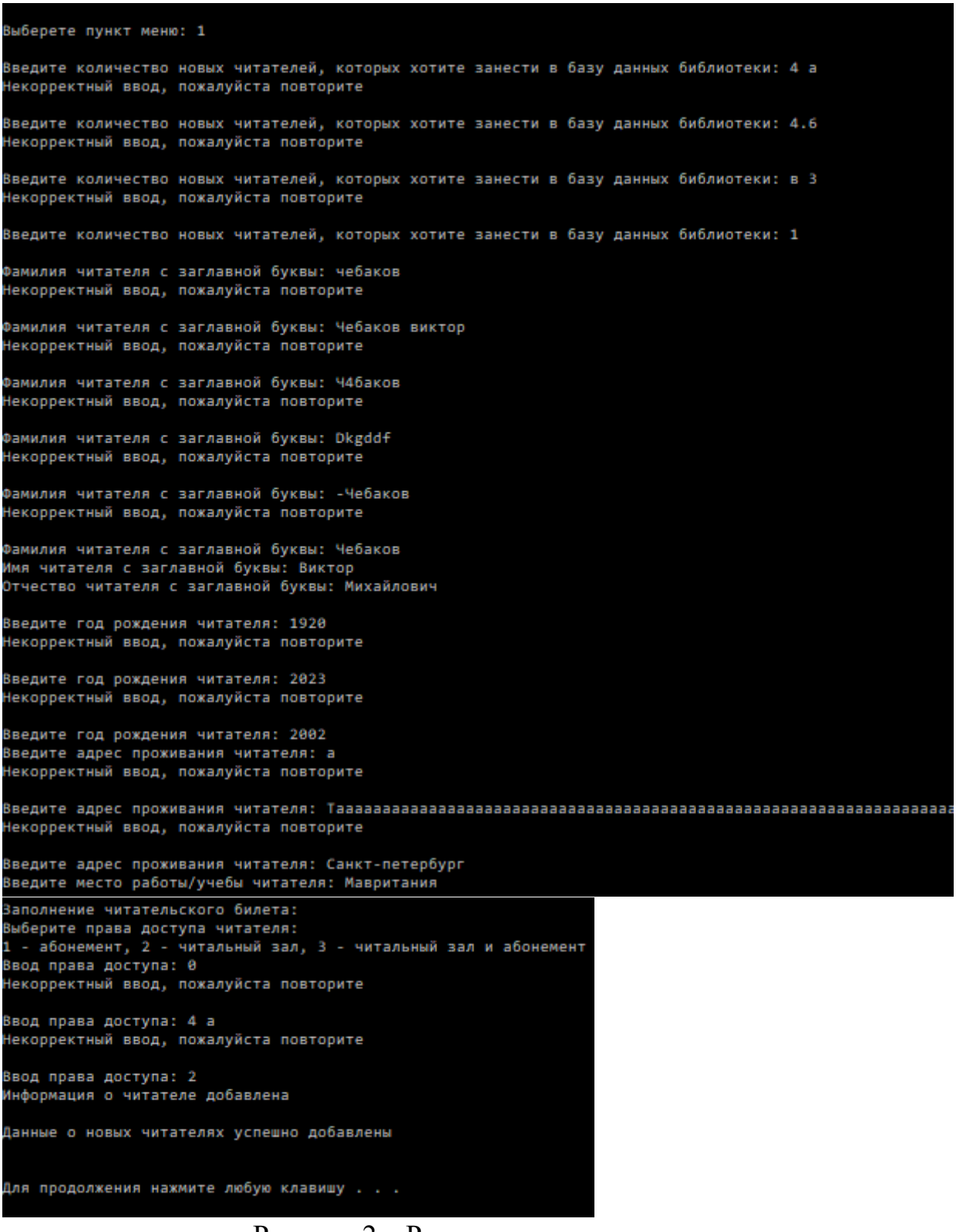


Рисунок 2 – Регистрация пациента

3. Просмотр списка зарегистрированных читателей

```
Выберете пункт меню: 2
Список читателей библиотеки:

N |Номер читательского билета  |ФИО
--|-----------------------------|------
1 |Ч0002-96                     |Рябцева Алина Алексеевна
2 |A0001-56                     |Чернышенко Юрий Анатольевич
3 |B0000-02                     |Пивнов Артем Владимирович


Для продолжения нажмите любую клавишу . . .
```

Рисунок 3 – Просмотр списка зарегистрированных читателей

4. Поиск читателя по ФИО

```
Выберете пункт меню: 3
Введите ФИО читателя, которого хотите найти

Фамилия читателя с заглавной буквы: Кузнецов
Имя читателя с заглавной буквы: Андрей
Отчество читателя с заглавной буквы: Анатольевич


Список найденных читателей: Пуст



Для продолжения нажмите любую клавишу . . .
```

```
Выберете пункт меню: 3
Введите ФИО читателя, которого хотите найти

Фамилия читателя с заглавной буквы: Чернышенко
Имя читателя с заглавной буквы: Юрий
Отчество читателя с заглавной буквы: Анатольевич


Список найденных читателей:

N |Номер читательского билета  |ФИО
--|-----------------------------|------
1 |A0001-56                     |Чернышенко Юрий Анатольевич


Для продолжения нажмите любую клавишу . . .
```

Рисунок 4 – Поиск читателя по ФИО

5. Поиск читателя по номеру читательского билета

```
Выберете пункт меню: 4
Список читателей библиотеки:

N |Номер читательского билета  |ФИО
--|-----------------------------|------
1 |Ч0002-96                     |Рябцева Алина Алексеевна
2 |A0001-56                     |Чернышенко Юрий Анатольевич
3 |B0000-02                     |Пивнов Артем Владимирович


Введите номер читательского билета в виде ANNNN-YY
A - права доступа (А - абонемент, Ч - читальный зал, В - читальный зал и абонемент
N - порядковый номер регистрации (цифры), YY - последние две цифры года рождения читателя

Ввод читательского билета: Ч0011-87

Читатель с данным номером не найден

Для продолжения нажмите любую клавишу . . .
```

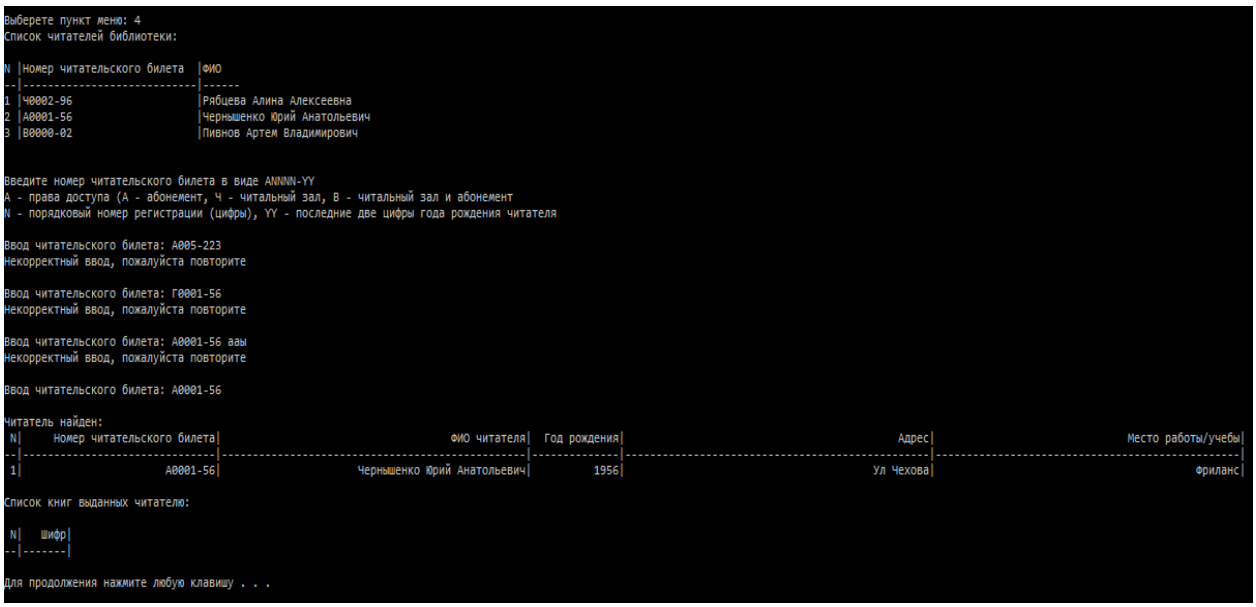
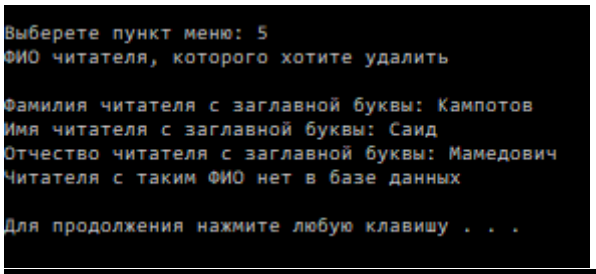
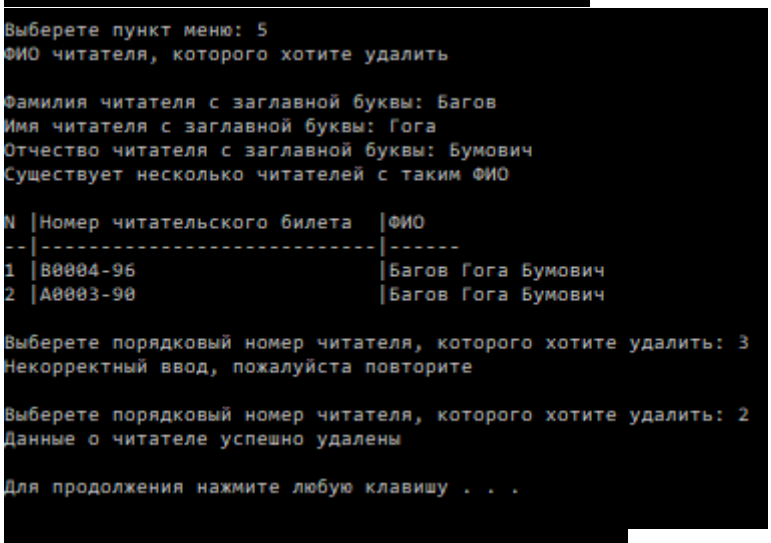
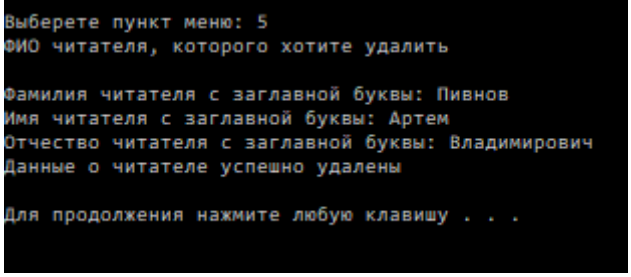


Рисунок 5 – Поиск читателя по номеру читательского билета

6. Удаление больного по ФИО







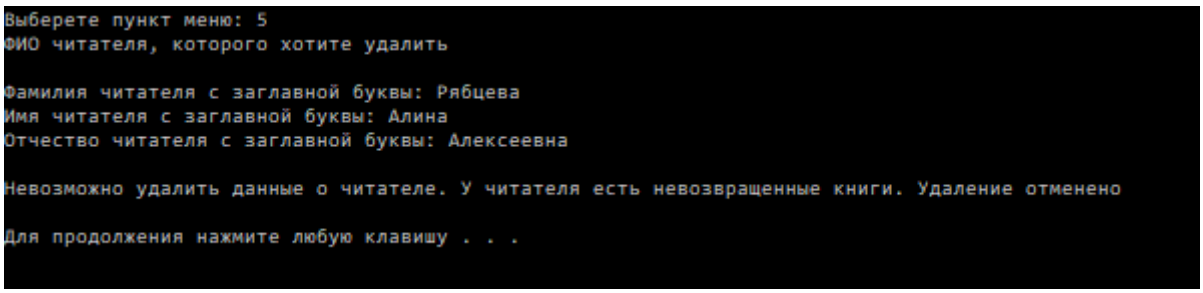


Рисунок 6 – Удаление читателя по ФИО

7. Очистка базы данных больных

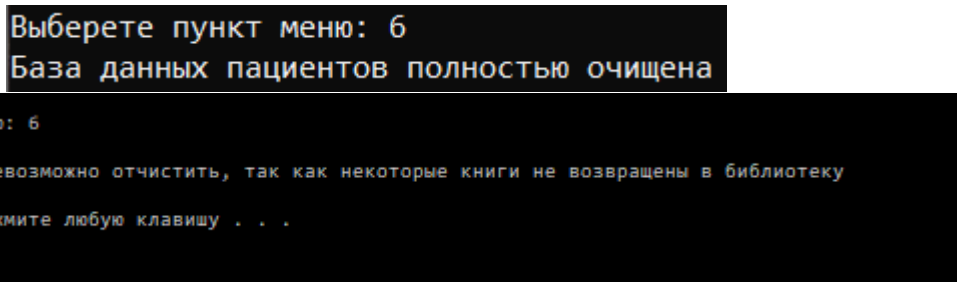


Рисунок 7 – Очистка базы данных читателей

8. Добавление новой книги

```
Выберете пункт меню: 7

Введите количество авторов, книги которых хотите занести в базу данных библиотеки: 1

Фамилия автора с заглавной буквы: Некрасов
Имя автора с заглавной буквы: Николай
Отчество автора с заглавной буквы: Павлович

Введите название книги: Стрельцы
Введите издательство книги: Графство
Введите год публикации книги: 1888
Введите количество экземпляров: -1
Некорректный ввод, пожалуйста повторите

Введите количество экземпляров: 0
Некорректный ввод, пожалуйста повторите

Введите количество экземпляров: 2 а
Некорректный ввод, пожалуйста повторите

Введите количество экземпляров: 5
Введите тип книги(1 - детектив, 2 - фентази, 3 - драма): 2

Данные о новых книгах успешно добавлены


Для продолжения нажмите любую клавишу . . .
```

Рисунок 8 – Добавление новой книги

9. Просмотр БД книг

```
Выберете пункт меню: 8

База книг библиотеки:

 N|   Шифр|              Название книги|                                    ФИО автора| кол-во копий всего|     осталось копий|
--|-------|----------------------------|----------------------------------------------|-------------------|-------------------|
 1|111.000|       Пираты карибского моря|                      Горлов Никита Юрьевич|                 10|                 10|
 2|111.001|        Генеологическое древо|                    Горлова Лидия Николаевна|                100|                100|
 3|222.000|               Политика везде|                    Рыбалева Дарья Андреевна|                 40|                 40|
 4|222.001|                    Стрельцы|                   Некрасов Николай Павлович|                  5|                  5|
 5|333.000|             Гармония музыки|                     Борисов Сергей Иванович|                 50|                 50|
 6|333.001|                    Экология|                      Морозов Иван Юрьевич|               2000|               1999|
 7|333.002|                        Юный|                   Романюк Максим Максимович|                400|                400|
 8|333.003|                      Бизнес|                   Горлов Виталий Сергеевич|                  1|                  1|



             333.003
         333.002
             333.001
     333.000
                 222.001
             222.000
         111.001
             111.000

Для продолжения нажмите любую клавишу . . .
```

Рисунок 9 - Просмотр БД книг

10.Поиска книги по шифру

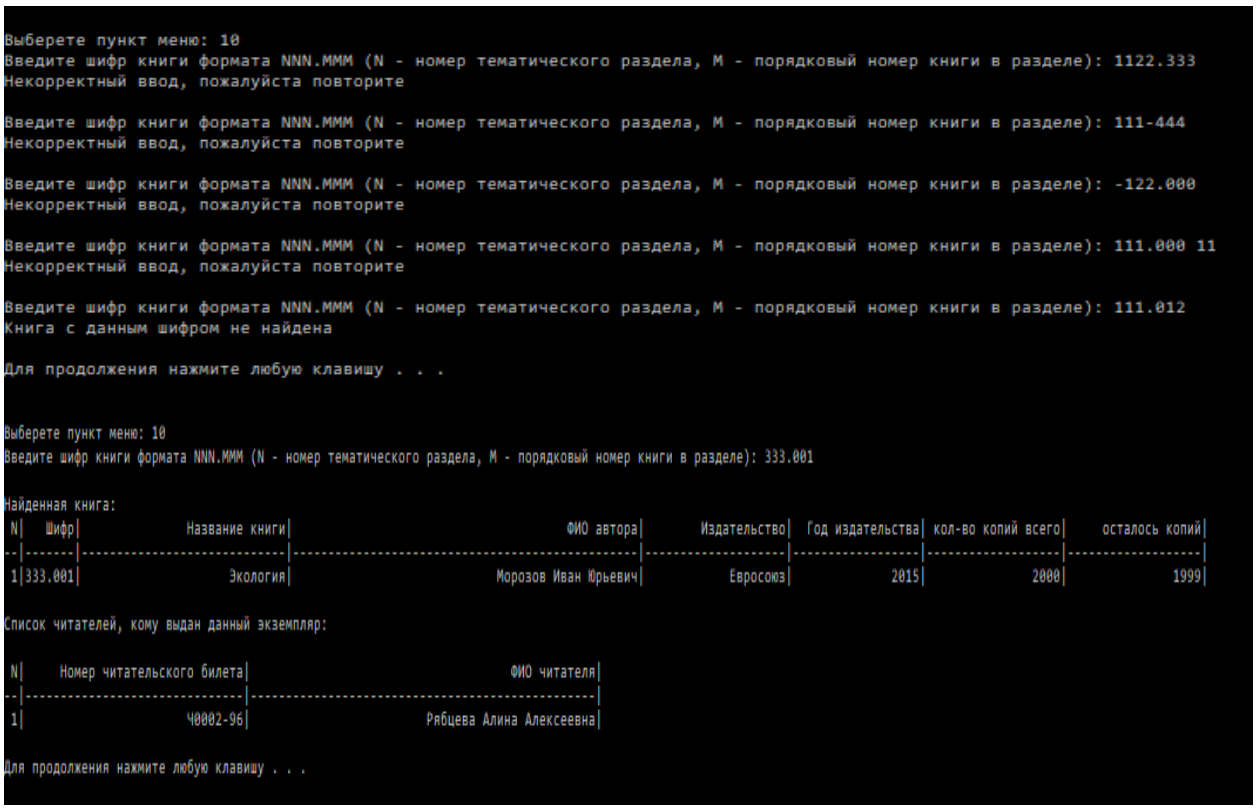


Рисунок 10 – Поиск книг по шифру

11.Поиск книги по фрагментам фио автора или названия

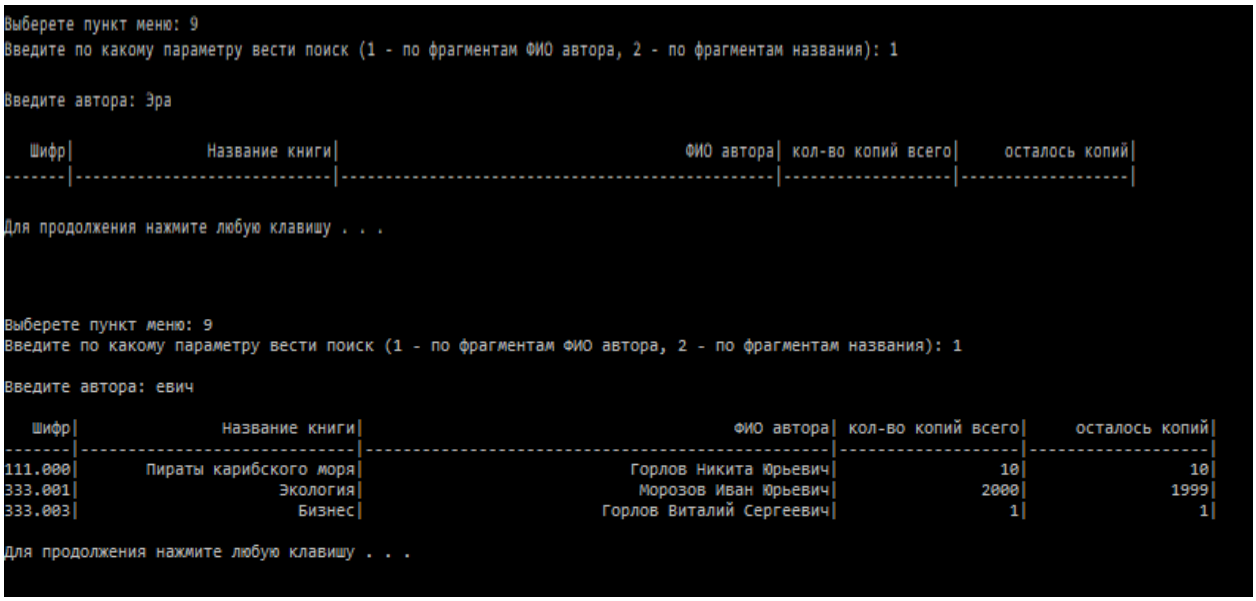


Рисунок 11 - Поиск книги по фрагментам фио автора или названия

12.Удаление сведений о книге

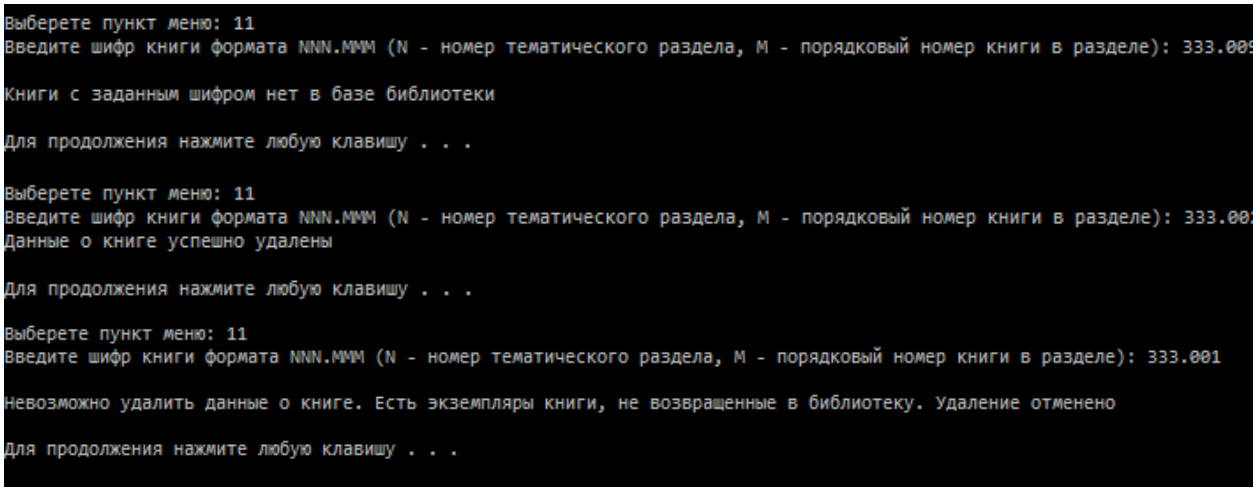


Рисунок 12 - Удаление сведений о книге

13.Очистка базы данных книг

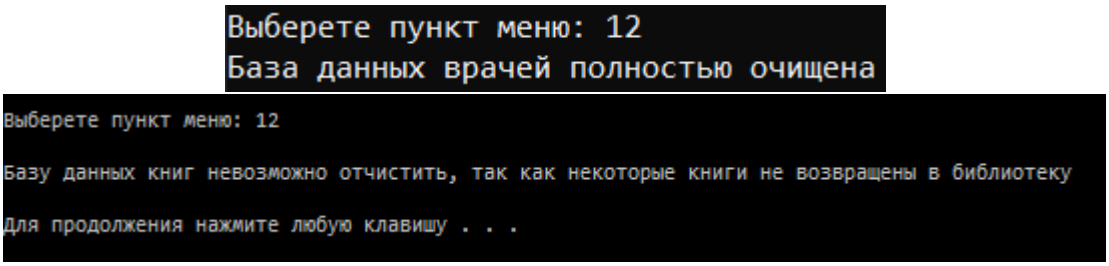


Рисунок 13 - Очистка базы данных книг

14.Выдача книги читателю

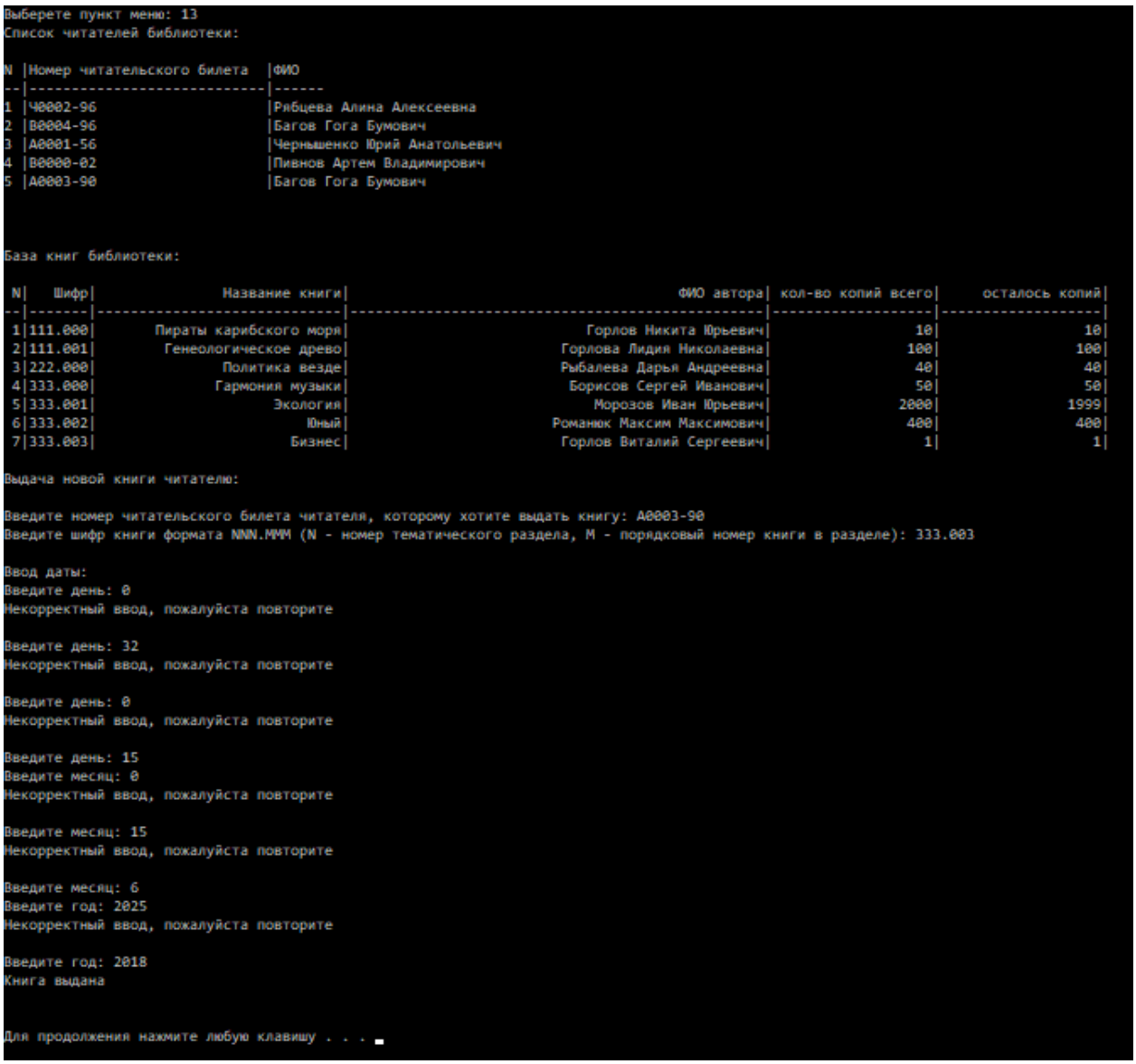


Рисунок 14 - Выдача книги читателю

15.Прием книги от читателя

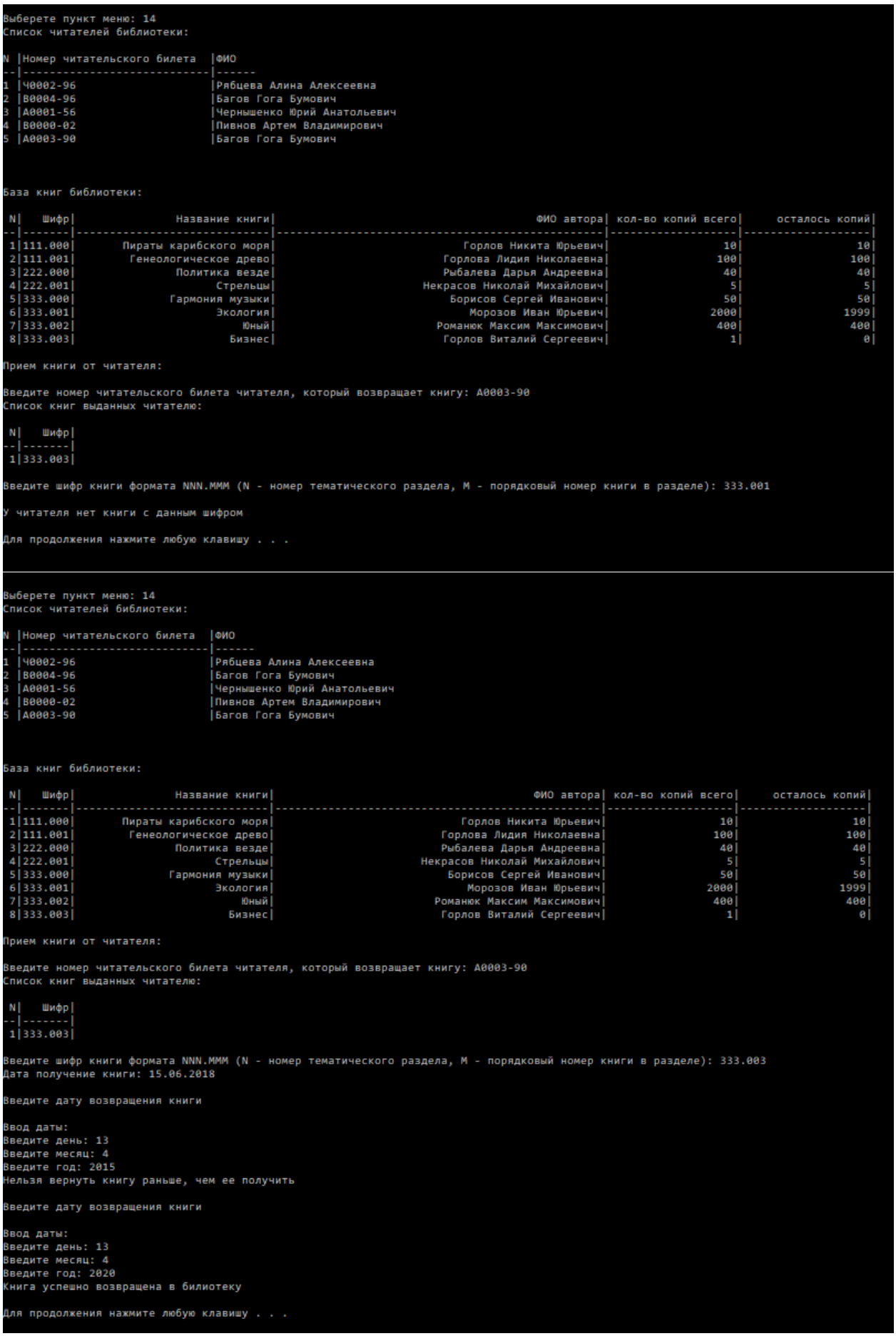


Рисунок 15 – Прием книги от читателя

16.Сортировка истории регистраций приема и выдачи книг

N	Шифр книги	Номер читательского билета	Дата выдачи	Дата приема
1	333.001	Ч0002-96	17.02.2020	У читателя
2	333.003	А0001-56	15.12.2009	15.12.2009
3	333.003	В0000-02	12.07.2015	13.07.2015
4	333.003	Ч0002-96	27.09.2016	27.09.2017
5	333.003	А0003-90	15.06.2018	13.04.2020
6	111.001	Ч0002-96	14.01.2020	У читателя
7	111.000	А0003-90	01.01.2020	У читателя
8	222.001	А0001-56	01.01.2020	У читателя

N	Шифр книги	Номер читательского билета	Дата выдачи	Дата приема
1	111.000	А0003-90	01.01.2020	У читателя
2	111.001	Ч0002-96	14.01.2020	У читателя
3	222.001	А0001-56	01.01.2020	У читателя
4	333.001	Ч0002-96	17.02.2020	У читателя
5	333.003	А0001-56	15.12.2009	15.12.2009
6	333.003	В0000-02	12.07.2015	13.07.2015
7	333.003	Ч0002-96	27.09.2016	27.09.2017
8	333.003	А0003-90	15.06.2018	13.04.2020

Рисунок 16 – История приема и выдачи книг

17. Вывод истории приема и выдачи книг

N	Шифр книги	Номер читательского билета	Дата выдачи	Дата приема
1	111.000	А0003-90	01.01.2020	У читателя
2	111.001	Ч0002-96	14.01.2020	У читателя
3	222.001	А0001-56	01.01.2020	У читателя
4	333.001	Ч0002-96	17.02.2020	У читателя
5	333.003	А0001-56	15.12.2009	15.12.2009
6	333.003	В0000-02	12.07.2015	13.07.2015
7	333.003	Ч0002-96	27.09.2016	27.09.2017
8	333.003	А0003-90	15.06.2018	13.04.2020

Рисунок 17 – Вывод истории приема и выдачи книг

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы, была разработана информационная система: обслуживание читателей в библиотеке. Данная система удовлетворяет заданным требованиям и решает все поставленные задачи. Во время выполнения работы были повторены используемые алгоритмы, а также реализации структур данных, необходимых для организации информационной системы.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / СПбГУАП. СПб., 2004.
2. Колдаев В.Д. Основы алгоритмизации и программирования: Учебное пособие / Колдаев В.Д; Под ред. проф.Л.Г. Гагариной - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2016. - 416 с.
3. Павловская Т. А. C/C++. Программирование на языке высокого уровня: учебник. СПб.: ПИТЕР, 2007. - 461 с

ПРИЛОЖЕНИЕ

```
#include <iostream>
#include <Windows.h>
#include <string>
#include <fstream>
#include <iomanip>

using namespace std;

const int size_of_spreadsheet = 100;
const int size_of_key = 8;
const int size_of_cipher = 7;
const int ascii_size = 256;
const int k = 7; //максимальное количество разрядов числа

string path_to_readers = "C:\\Users\\Виталя\\Documents\\ВУЗ домашки\\Курсач по САОДу\\Курсач по САОДу\\Читатели.txt";
string path_to_books = "C:\\Users\\Виталя\\Documents\\ВУЗ домашки\\Курсач по САОДу\\Курсач по САОДу\\Книги.txt";
string path_to_registration = "C:\\Users\\Виталя\\Documents\\ВУЗ домашки\\Курсач по САОДу\\Курсач по САОДу\\ИсторияРегистраций.txt";

struct Reader
{
	Reader()
	{
		library_card_number = "\0";
		FIO = "\0";
		adress = "\0";
		workplace = "\0";
		Year_of_birth = 0;
	}

	string library_card_number{};
	string FIO{};
	string adress{};
	string workplace{};
	int Year_of_birth{};
	static int id;

	void SetInfo(string li, string fio, string ad, string work, int year)
	{
		library_card_number = li;
		FIO = fio;
		adress = ad;
```

```
		workplace = work;
		Year_of_birth = year;
	}

	void SetZeroes()
	{
		library_card_number = "delete";
		FIO = "\0";
		adress = "\0";
		workplace = "\0";
		Year_of_birth = 0;
	}

	Reader& operator=(const Reader& other)
	{
		this->library_card_number = other.library_card_number;
		this->FIO = other.FIO;
		this->adress = other.adress;
		this->workplace = other.workplace;
		this->Year_of_birth = other.Year_of_birth;
		return *this;
	}

};

struct Book
{
	Book()
	{
		cipher = "\0";
		aouthor = "\0";
		title = "\0";
		publishing = "\0";
		year_of_publishing = 0;
		amount_of_copies = 0;
		copies_available = 0;
		height = 1;
		left = nullptr;
		right = nullptr;
	}

	string cipher{};
	string aouthor{};
	string title{};
	string publishing{};
	int year_of_publishing{};
```

```
        int amount_of_copies{};
        int copies_available{};
        int height{};
        Book* left{};
        Book* right{};

        enum
        {
                detective = 111,
                fantasy = 222,
                drama = 333
        };

        static int id_detective;
        static int id_fantasy;
        static int id_drama;

        void SetInfo(string ciphe, string aoutho, string titl, string publishin, int
year_of_publishin, int amount_of_copie)
        {
                cipher = ciphe;
                aouthor = aoutho;
                title = titl;
                publishing = publishin;
                year_of_publishing = year_of_publishin;
                amount_of_copies = amount_of_copie;
                copies_available = amount_of_copie;
        }

        void SetInfo(Book& other)
        {
                cipher = other.cipher;
                aouthor = other.aouthor;
                title = other.title;
                publishing = other.publishing;
                year_of_publishing = other.year_of_publishing;
                amount_of_copies = other.amount_of_copies;
                copies_available = other.copies_available;
        }

        bool Equel(Book& other)
        {
                return (cipher[0] == other.cipher[0] and aouthor == other.aouthor and title ==
other.title and publishing == other.publishing and year_of_publishing ==
other.year_of_publishing);
        }
```

```
        void SetZeroes()
        {
                cipher = "\0";
                aouthor = "\0";
                title = "\0";
                publishing = "\0";
                year_of_publishing = 0;
                amount_of_copies = 0;
                copies_available = 0;
                height = 1;
                left = nullptr;
                right = nullptr;
        }

};

struct Taken
{
        Taken()
        {
                library_card = "\0";
                book_cipher = "\0";
                date_when_taken = "\0";
                date_when_returned = "\0";
                int_cipher = 0;
                next = back = nullptr;
        }

        Taken(string card, string cipher, string date_taken, string date_returned)
        {
                library_card = card;
                book_cipher = cipher;
                date_when_taken = date_taken;
                date_when_returned = date_returned;
                int_cipher = 0;
                next = back = nullptr;
        }

        string library_card{};
        string book_cipher{};
        int int_cipher{};
        string date_when_taken{};
        string date_when_returned{};
        Taken* next{};
        Taken* back{};
```

```
	Taken& operator=(const Taken& other)
	{
		library_card = other.library_card;
		book_cipher = other.book_cipher;
		date_when_taken = other.date_when_taken;
		date_when_returned = other.date_when_returned;
		int_cipher = other.int_cipher;
		next = back = nullptr;
		return *this;
	}
};

struct list
{
	list(Taken& value, list* n)
	{
		data = value;
		next = n;
	}

	Taken data;
	list* next;
};

int Reader::id = 0;
int Book::id_detective = 0;
int Book::id_fantasy = 0;
int Book::id_drama = 0;

void Menu()
{
	cout << " ____________________________________________________________________________" <<
endl;
	cout << "/                                МЕНЮ                                        \\" <<
endl;
	cout << "|____________________________________________________________________________|" <<
endl;
	cout << "|Читатели:                                                                   |" <<
endl;
	cout << "|----------------------------------------------------------------------------|" <<
endl;
	cout << "|Нажмите 1, чтобы зарегистрировать нового читателя.                          |" <<
endl;
	cout << "|Нажмите 2, чтобы отобразить всех читателей библиотеки.                      |" <<
endl;
```

```
        cout << "|Нажмите 3, чтобы найти читателя по ФИО.                              |" <<
endl;
        cout << "|Нажмите 4, чтобы найти читателя по номеру читательского билета.      |" <<
endl;
        cout << "|Нажмите 5, чтобы удалить читателя.                                   |" <<
endl;
        cout << "|Нажмите 6, чтобы очистить базу данных читателей.                     |" <<
endl;
        cout << "|_____________________________________________________________________|" <<
endl;
        cout << "|Библиотека книг:                                                     |" <<
endl;
        cout << "|---------------------------------------------------------------------|" <<
endl;
        cout << "|Нажмите 7, чтобы добавить книги в базу данных бибилотеки.            |" <<
endl;
        cout << "|Нажмите 8, чтобы вывести базу книг библиотеки.                       |" <<
endl;
        cout << "|Нажмите 9, чтобы найти книгу по фрагментам ФИО автора или названия.  |" <<
endl;
        cout << "|Нажмите 10, чтобы найти книгу по шифру.                              |" <<
endl;
        cout << "|Нажмите 11, чтобы удалить книгу из базы данных библиотеки.           |" <<
endl;
        cout << "|Нажмите 12, чтобы очистить базу книг библиотеки.                     |" <<
endl;
        cout << "|_____________________________________________________________________|" <<
endl;
        cout << "|Регистрация выдачи и приема книг:                                    |" <<
endl;
        cout << "|---------------------------------------------------------------------|" <<
endl;
        cout << "|Нажмите 13, чтобы выдать книгу читателю.                             |" <<
endl;
        cout << "|Нажмите 14, чтобы принять книгу от читателя.                         |" <<
endl;
        cout << "|Нажмите 15, чтобы отсортировать историю регисраций по шифру книг     |" <<
endl;
        cout << "|Нажмите 16, чтобы отобразить историю регистрации приема и выдачи книг|" <<
endl;
        cout << "|_____________________________________________________________________|" <<
endl;
        cout << "|Нажмите 17, чтобы выйти из программы.                                |" <<
endl;
        cout << "\\_____________________________________________________________________/" <<
endl << endl;
```

```
}

int ChekInt(string text)
{
	while (true)
	{
		int digit;
		cout << text;
		cin >> digit;

		if (cin.fail() || cin.get() != '\n')
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			cin.clear();
			cin.ignore(32767, '\n');
			continue;
		}

		if (text == "Выберете пункт меню: ")
		{
			if (digit < 1 || digit > 17)
			{
				cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
				continue;
			}
			return digit;
		}

		if (text == "Введите количество новых читателей, которых хотите занести в базу
данных библиотеки: ")
		{
			if (digit < 0)
			{
				cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
				continue;
			}
			return digit;
		}

		if (text == "Введите год рождения читателя: ")
		{
			if (digit < 1930 or digit > 2014)
			{
				cout << "Некорректный ввод, пожалуйста повторите" << endl <<
```

```
endl;
                        continue;
                }
                return digit;
        }

        if (text == "Ввод права доступа: ")
        {
                if (digit < 1 or digit > 3)
                {
                        cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
                        continue;
                }
                return digit;
        }

        if (text == "Введите количество авторов, книги которых хотите занести в базу
данных библиотеки: ")
        {
                if (digit < 0)
                {
                        cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
                        continue;
                }
                return digit;
        }

        if (text == "Введите год публикации книги: ")
        {
                if (digit < 1800 or digit > 2022)
                {
                        cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
                        continue;
                }
                return digit;
        }

        if (text == "Введите количество экземпляров: ")
        {
                if (digit <= 0)
                {
                        cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
```

```
                        continue;
                }
                return digit;
        }

        if (text == "Введите тип книги(1 - детектив, 2 - фентази, 3 - драма): ")
        {
                if (digit < 1 or digit > 3)
                {
                        cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
                        continue;
                }
                return digit;
        }

        if (text == "Введите по какому параметру вести поиск (1 - по фрагментам ФИО
автора, 2 - по фрагментам названия): ")
        {
                if (digit < 1 or digit > 2)
                {
                        cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
                        continue;
                }
                return digit;
        }

        if (text == "Введите день: ")
        {
                if (digit < 1 or digit > 31)
                {
                        cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
                        continue;
                }
        }

        if (text == "Введите месяц: ")
        {
                if (digit < 1 or digit > 12)
                {
                        cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
                        continue;
                }
```

```
		}

		if (text == "Введите год: ")
		{
			if (digit < 2000 or digit > 2021)
			{
				cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
				continue;
			}
		}

		return digit;
	}
}

void ChekName(string& name, string text)
{
	while (true)
	{
		cout << text;
		cin >> name;
		bool incorrect = false;
		if (cin.get() != '\n')
		{
			cin.ignore(32767, '\n');
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
		}
		else
		{
			if (int(name[0]) > -33 || int(name[0]) < -64)
			{
				cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
				continue;
			}

			int i = 1;
			while (name[i] != '\0')
			{
				if (int(name[i]) > -1 || int(name[i]) < -32)
				{
					cout << "Некорректный ввод, пожалуйста повторите" << endl
<< endl;
					incorrect = true;
					break;
```

```
                        }
                        i++;
                }
                if (incorrect)
                {
                        continue;
                }
                else
                {
                        break;
                }
        }
    }
}

void ChekAdress(string& ad, string text)
{
    while (true)
    {
        cout << text;
        getline(cin, ad);
        int i = 0;

        if (ad.length() > 50)
        {
            cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
            continue;
        }

        if (ad.length() < 2)
        {
            cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
            continue;
        }

        while (ad[i] != '\0')
        {
            if (int(ad[i]) < -64 or (int(ad[i]) > -1 and ad[i] < ' ') or (ad[i] > ' '
and ad[i] < '-') or (ad[i] > '-' and ad[i] < '0') or ad[i] > '9')
            {
                cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
                continue;
            }
            i++;
        }
```

```
		if (ad[0] == '-')
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}

		break;
	}
}

void ChekCard(string& card, string text)
{
	bool incorrect;
	while (true)
	{
		cout << text;
		cin >> card;
		incorrect = false;
		if (cin.get() != '\n')
		{
			cin.ignore(32767, '\n');
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}

		if (card.length() != size_of_key)
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}

		if (card[0] != 'A' and card[0] != 'Ч' and card[0] != 'B')
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}

		if (card[5] != '-')
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}


		for (int i = 1; i < size_of_key; i++)
```

```
		{
			if (i == 5)
			{
				continue;
			}

			if (card[i] < '0' or card[i] > '9')
			{
				incorrect = true;
				break;
			}
		}

		if (incorrect)
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}

		break;
	}
}

string ChekDate()
{
	int day, month, year;
	string date = "\0";

	while (true)
	{
		cout << endl << "Ввод даты: " << endl;
		day = ChekInt("Введите день: ");
		month = ChekInt("Введите месяц: ");
		year = ChekInt("Введите год: ");

		if (day > 28 and month == 2 and year % 4 != 0)
		{
			cout << "-----------------------------" << endl;
			cout << "| Заданная дата некорректна |" << endl;
			cout << "-----------------------------" << endl;
			continue;
		}

		else if (day > 28 and year % 4 == 0 and year % 100 == 0 and year % 400 != 0)
		{
			cout << "-----------------------------" << endl;
```

```
                        cout << "| Заданная дата некорректна |" << endl;
                        cout << "-----------------------------" << endl;
                        continue;
                }

                else if (day > 31 and ((month % 2 == 1 and month <= 7) or (month % 2 == 0 and
month > 7)))
                {
                        cout << "-----------------------------" << endl;
                        cout << "| Заданная дата некорректна |" << endl;
                        cout << "-----------------------------" << endl;
                        continue;
                }

                else if (day > 30 and ((month % 2 == 0 and month <= 7) or (month % 2 == 1 and
month > 7)))
                {
                        cout << "-----------------------------" << endl;
                        cout << "| Заданная дата некорректна |" << endl;
                        cout << "-----------------------------" << endl;
                        continue;
                }

                break;
        }

        if (day < 10)
        {
                date += "0";
        }

        date += to_string(day);
        date += ".";

        if (month < 10)
        {
                date += "0";
        }
        date += to_string(month);
        date += ".";
        date += to_string(year);

        return date;
}

void FormLibraryCard(string& card, int birth)
```

```
{
	int rights;
	cout  << endl << "Заполнение читательского билета:" << endl;
	cout << "Выберите права доступа читателя:" << endl;
	cout << "1 - абонемент, 2 - читальный зал, 3 - читальный зал и абонемент" << endl;
	rights = ChekInt("Ввод права доступа: ");

	int number;
	number = Reader::id;
	Reader::id++;
	string str_number = to_string(number);

	birth %= 100;
	string str_birth = to_string(birth);

	if (rights == 1)
	{
		card += "A";
	}

	else if (rights == 2)
	{
		card += "Ч";
	}

	else
	{
		card += "B";
	}

	for (int i = 0; i < 4 - str_number.length(); i++)
	{
		card += "0";
	}

	card += str_number;
	card += "-";
	if (birth < 10)
	{
		card += "0";
	}
	card += str_birth;
}

void ChekCipher(string& cipher, string text)
{
```

```
	while (true)
	{
		cout << text;
		cin >> cipher;
		bool incorrect = false;

		if (cin.get() != '\n')
		{
			cin.ignore(32767, '\n');
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}

		if (cipher[3] != '.')
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}

		for (int i = 0; i < size_of_cipher; i++)
		{
			if (i == 3)
			{
				continue;
			}

			if (cipher[i] < '0' or cipher[i] > '9')
			{
				incorrect = true;
				break;
			}
		}

		if (incorrect)
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}
		else
		{
			break;
		}
	}
}

void ChekSubstring(string& substring, string text)
```

```
{
	while (true)
	{
		cout << text;
		getline(cin, substring);
		bool incorrect = false;

		int i = 0;
		while (substring[i] != '\0')
		{
			if (int(substring[i]) < -64 or (int(substring[i]) > -1 and
int(substring[i]) < 44) or int(substring[i]) > 57)
			{
				cout << "Некорректный ввод, пожалуйста повторите" << endl <<
endl;
				incorrect = true;
				break;
			}
			i++;
		}

		if (incorrect)
		{
			continue;
		}

		else
		{
			break;
		}
	}
}

void InfoAboutReader(Reader& new_reader)
{
	string F, I, O, FIO, work_place, adress, card;
	int birtday;

	ChekName(F, "Фамилия читателя с заглавной буквы: ");
	ChekName(I, "Имя читателя с заглавной буквы: ");
	ChekName(O, "Отчество читателя с заглавной буквы: ");
	FIO = F + ' ' + I + ' ' + O;
	cout << endl;

	birtday = ChekInt("Введите год рождения читателя: ");
```

```
	ChekAdress(adress, "Введите адрес проживания читателя: ");
	ChekAdress(work_place, "Введите место работы/учебы читателя: ");

	FormLibraryCard(card, birtday);

	new_reader.SetInfo(card, FIO, adress, work_place, birtday);
}

int HashFunction(string key)
{
	int hash = 0;
	for (int i = 0; i < size_of_key; i++)
	{
		hash += pow(int(key[i]), 3);
	}
	hash %= size_of_spreadsheet;
	return hash;
}

bool FillHashSpreadsheet(Reader* array_of_readers, Reader& new_reader, int hashed_key)
{
	bool key_setted = false;

	if (array_of_readers[hashed_key].library_card_number == "\0" or
array_of_readers[hashed_key].library_card_number == "delete")
	{
		array_of_readers[hashed_key] = new_reader;
		key_setted = true;
		cout << "Информация о читателе добавлена" << endl << endl;
		return key_setted;
	}

	else
	{
		int const const_1 = 3;
		int const const_2 = 5;
		int adress;

		for (int i = 0; i < 75; i++)
		{
			adress = (hashed_key + const_1 * i + const_2 * (i * i)) %
size_of_spreadsheet;
			if (array_of_readers[adress].library_card_number == "\0" or
array_of_readers[adress].library_card_number == "delete")
			{
				array_of_readers[adress] = new_reader;
```

```
                        key_setted = true;
                        cout << "Информация о читателе добавлена" << endl << endl;
                        return key_setted;
                }
        }

        hashed_key++;

        for (int i = 0; i < 75; i++)
        {
                adress = (hashed_key + const_1 * i + const_2 * (i * i)) %
size_of_spreadsheet;
                if (array_of_readers[adress].library_card_number == "\0" or
array_of_readers[adress].library_card_number == "delete")
                {
                        array_of_readers[adress] = new_reader;
                        key_setted = true;
                        cout << "Информация о читателе добавлена" << endl << endl;
                        return key_setted;
                }
        }

    }
    cout << "Информация о читателе не добавлена, таблица переполнена" << endl << endl;
    return key_setted;
}

void ShowSpreadsheet(Reader* spreadsheet_of_readers)
{
    cout << "Список читателей библиотеки:" << endl << endl;
    cout << "N |Номер читательского билета  |ФИО " << endl ;
    cout << "--|---------------------------|------" << endl;
    int N = 1;
    for (int i = 0; i < size_of_spreadsheet; i++)
    {
        if (spreadsheet_of_readers[i].FIO != "\0")
        {
            cout << N;
            if (N < 10)
            {
                cout << " ";
            }
            cout << "|" << spreadsheet_of_readers[i].library_card_number;
            cout << "                    |" << spreadsheet_of_readers[i].FIO << endl;
            N++;
        }
```

```
	}
	cout << endl;
}

void FindReaderByFio(Reader* spreadsheet_of_readers)
{
	string F, I, O, FIO;
	cout << "Введите ФИО читателя, которого хотите найти" << endl << endl;

	ChekName(F, "Фамилия читателя с заглавной буквы: ");
	ChekName(I, "Имя читателя с заглавной буквы: ");
	ChekName(O, "Отчество читателя с заглавной буквы: ");
	FIO = F + ' ' + I + ' ' + O;
	cout << endl << endl;
	int amount = 1;
	int found = 0;
	cout << "Список найденных читателей: ";

	for (int i = 0; i < size_of_spreadsheet; i++)
	{
		if (spreadsheet_of_readers[i].FIO == FIO)
		{
			found++;
			if (found == 1)
			{
				cout << endl << endl;
				cout << "N |Номер читательского билета  |ФИО " << endl;
				cout << "--|----------------------------|------" << endl;
				found++;
			}

			cout << amount;
			if (amount < 10)
			{
				cout << " ";
			}
			cout << "|" << spreadsheet_of_readers[i].library_card_number;
			cout << "                    |" << spreadsheet_of_readers[i].FIO << endl;
			amount++;
		}
	}

	if (found == 0)
	{
		cout << "Пуст" << endl << endl;
	}
```

```
	cout << endl;
}

int FindReaderByCard(Reader* spreadsheet_of_readers, string card)
{
	int hashed_key = HashFunction(card);
	int adress = hashed_key;

	if (spreadsheet_of_readers[adress].library_card_number == card)
	{
		return adress;
	}

	if (spreadsheet_of_readers[adress].library_card_number == "\0")
	{
		return size_of_spreadsheet;
	}

	int const const_1 = 3;
	int const const_2 = 5;

	for (int i = 0; i < 75; i++)
	{
		adress = (hashed_key + const_1 * i + const_2 * (i * i)) % size_of_spreadsheet;
		if (spreadsheet_of_readers[adress].library_card_number == card)
		{
			return adress;
		}

		if (spreadsheet_of_readers[adress].library_card_number == "\0")
		{
			return size_of_spreadsheet;
		}
	}

	hashed_key++;
	for (int i = 0; i < 75; i++)
	{
		adress = (hashed_key + const_1 * i + const_2 * (i * i)) % size_of_spreadsheet;
		if (spreadsheet_of_readers[adress].library_card_number == card)
		{
			return adress;
		}

		if (spreadsheet_of_readers[adress].library_card_number == "\0")
```

```
		{
			return size_of_spreadsheet;
		}
	}

	return size_of_spreadsheet;
}

bool CanDeleteReader(Taken* start, string card)
{
	while (start != nullptr)
	{
		if (start->library_card == card and start->date_when_returned == "У читателя")
		{
			return false;
		}
		start = start->next;
	}
	return true;
}

void RemoveWholeInfoAboutReaders(Reader* spreadsheet_of_readers, Taken* start)
{
	while (start != nullptr)
	{
		if (start->date_when_returned == "У читателя")
		{
			cout << endl << "Базу данных книг невозможно отчистить, так как некоторые
книги не возвращены в библиотеку" << endl;
			return;
		}
		start = start->next;
	}

	for (int i = 0; i < size_of_spreadsheet; i++)
	{
		spreadsheet_of_readers[i].SetZeroes();
	}

	cout << "База данных книг отчищена" << endl;
}

void DeleteReader(Reader* spreadsheet_of_readers, Taken* start)
{
	int candidates_to_delete = 0;
	string FIO, F, I, O;
```

```
	cout << "ФИО читателя, которого хотите удалить" << endl << endl;
	ChekName(F, "Фамилия читателя с заглавной буквы: ");
	ChekName(I, "Имя читателя с заглавной буквы: ");
	ChekName(O, "Отчество читателя с заглавной буквы: ");
	FIO = F + ' ' + I + ' ' + O;

	for (int i = 0; i < size_of_spreadsheet; i++)
	{
		if (spreadsheet_of_readers[i].FIO == FIO)
		{
			candidates_to_delete++;
		}
	}

	if (candidates_to_delete == 0)
	{
		cout << "Читателя с таким ФИО нет в базе данных" << endl;
		return;
	}

	if (candidates_to_delete == 1)
	{
		for (int i = 0; i < size_of_spreadsheet; i++)
		{
			if (spreadsheet_of_readers[i].FIO == FIO)
			{
				if (CanDeleteReader(start,
spreadsheet_of_readers[i].library_card_number))
				{
					spreadsheet_of_readers[i].SetZeroes();
					cout << "Данные о читателе успешно удалены" << endl;
					return;
				}

				else
				{
					cout << endl << "Невозможно удалить данные о читателе. У
читателя есть невозвращенные книги. Удаление отменено" << endl;
					return;
				}
			}
		}
	}

	int number_to_delete = 1;
	cout << "Существует несколько читателей с таким ФИО" << endl << endl;
```

```
	cout << "N |Номер читательского билета  |ФИО " << endl;
	cout << "--|---------------------------|------" << endl;
	for (int i = 0; i < size_of_spreadsheet; i++)
	{
		if (spreadsheet_of_readers[i].FIO == FIO)
		{
			cout << number_to_delete;
			if (number_to_delete < 10)
			{
				cout << " ";
			}
			cout << "|" << spreadsheet_of_readers[i].library_card_number;
			cout << "                    |" << spreadsheet_of_readers[i].FIO << endl;
			number_to_delete++;
		}
	}

	cout << endl;
	while (true)
	{
		number_to_delete = ChekInt("Выберете порядковый номер читателя, которого хотите
удалить: ");
		if (number_to_delete > candidates_to_delete or number_to_delete <= 0)
		{
			cout << "Некорректный ввод, пожалуйста повторите" << endl << endl;
			continue;
		}
		break;
	}

	int compare_to_delete = 1;
	for (int i = 0; i < size_of_spreadsheet; i++)
	{
		if (spreadsheet_of_readers[i].FIO == FIO)
		{
			if (compare_to_delete == number_to_delete)
			{
				if (CanDeleteReader(start,
spreadsheet_of_readers[i].library_card_number))
				{
					spreadsheet_of_readers[i].SetZeroes();
					cout << "Данные о читателе успешно удалены" << endl;
					return;
				}

				else
```

```
                        {
                                cout << endl << "Невозможно удалить данные о читателе. У
читателя есть невозвращенные книги. Удаление отменено" << endl;
                                return;
                        }
                }
                compare_to_delete++;
        }
    }

}

void WriteReadersInFile(Reader* spreadsheet_of_readers)
{
    ofstream fout;
    fout.open(path_to_readers);

    if (!fout.is_open())
    {
        cout << "Ошибка открытия файла" << endl;
        return;
    }

    for (int i = 0; i < size_of_spreadsheet; i++)
    {
        if (spreadsheet_of_readers[i].Year_of_birth != 0)
        {
            fout << spreadsheet_of_readers[i].library_card_number << "\n";
            fout << spreadsheet_of_readers[i].FIO << "\n";
            fout << spreadsheet_of_readers[i].adress << "\n";
            fout << spreadsheet_of_readers[i].workplace << "\n";
            fout << spreadsheet_of_readers[i].Year_of_birth << '\n';
        }
    }

    fout.close();
}

void ReadersFromFile(Reader* spreadsheet_of_readers)
{
    ifstream fin;
    fin.open(path_to_readers);

    if (!fin.is_open())
    {
        cout << "Ошибка открытия файла" << endl;
```

```
		return;
	}

	Reader tmp;
	char ex;
	int max_id = -1;
	int tmp_id;
	int hashed_card;

	while (!fin.eof())
	{
		tmp.SetZeroes();
		fin >> tmp.library_card_number;
		fin.get(ex);
		getline(fin, tmp.FIO);

		if (tmp.FIO == "\0")
		{
			break;
		}

		getline(fin, tmp.adress);
		getline(fin, tmp.workplace);
		fin >> tmp.Year_of_birth;
		tmp_id = (int(tmp.library_card_number[1]) - 48) * 1000 +
(int(tmp.library_card_number[2]) - 48) * 100 + (int(tmp.library_card_number[3]) - 48) * 10 +
(int(tmp.library_card_number[4]) - 48) * 1;

		if (tmp_id > max_id)
		{
			max_id = tmp_id;
		}

		hashed_card = HashFunction(tmp.library_card_number);
		FillHashSpreadsheet(spreadsheet_of_readers, tmp, hashed_card);
	}

	fin.close();
	Reader::id = max_id + 1;
	system("cls");
	return;
}

void InfoAboutBook(Book& new_book)
{
	string F, I, O, FIO, title, publiching, cipher;
```

```
	int year_of_publish, amount_of_copies;

	ChekName(F, "Фамилия автора с заглавной буквы: ");
	ChekName(I, "Имя автора с заглавной буквы: ");
	ChekName(O, "Отчество автора с заглавной буквы: ");
	FIO = F + ' ' + I + ' ' + O;
	cout << endl;

	ChekAdress(title, "Введите название книги: ");
	ChekAdress(publiching, "Введите издательство книги: ");
	year_of_publish = ChekInt("Введите год публикации книги: ");
	amount_of_copies = ChekInt("Введите количество экземпляров: ");

	int type_of_book = ChekInt("Введите тип книги(1 - детектив, 2 - фентази, 3 - драма):
");
	string id_of_book;

	if (type_of_book == 1)
	{
		cipher = to_string(Book::detective);
		id_of_book = to_string(Book::id_detective);
		Book::id_detective++;
	}

	else if (type_of_book == 2)
	{
		cipher = to_string(Book::fantasy);
		id_of_book = to_string(Book::id_fantasy);
		Book::id_fantasy++;
	}

	else
	{
		cipher = to_string(Book::drama);
		id_of_book = to_string(Book::id_drama);
		Book::id_drama++;
	}

	cipher += ".";
	for (int i = 0; i < 3 - id_of_book.length(); i++)
	{
		cipher += "0";
	}
	cipher += id_of_book;

	new_book.SetInfo(cipher, FIO, title, publiching, year_of_publish, amount_of_copies);
```

```
}

int Height(Book* p)
{
	return p ? p->height : 0;
}

int Bfactor(Book* p)
{
	return Height(p->right) - Height(p->left);
}

void FixHeight(Book* p)
{
	unsigned char hl = Height(p->left);
	unsigned char hr = Height(p->right);
	p->height = (hl > hr ? hl : hr) + 1;
}

Book* RotateRight(Book* p) // правый поворот вокруг p
{
	Book* q = p->left;
	p->left = q->right;
	q->right = p;
	FixHeight(p);
	FixHeight(q);
	return q;
}

Book* RotateLeft(Book* q) // левый поворот вокруг q
{
	Book* p = q->right;
	q->right = p->left;
	p->left = q;
	FixHeight(q);
	FixHeight(p);
	return p;
}

Book* balance(Book* p) // балансировка узла p
{
	FixHeight(p);
	if (Bfactor(p) == 2)
	{
		if (Bfactor(p->right) < 0)
			p->right = RotateRight(p->right);
```

```
		return RotateLeft(p);
	}
	if (Bfactor(p) == -2)
	{
		if (Bfactor(p->left) > 0)
			p->left = RotateLeft(p->left);
		return RotateRight(p);
	}
	return p; // балансировка не нужна
}

Book* Insert(Book* p, Book& new_book) // вставка элемента в дерево с корнем p
{
	if (!p)
	{
		Book* new_elem_for_insert = new Book;
		new_elem_for_insert->SetInfo(new_book);

		return new_elem_for_insert;
	}

	if (p->Equel(new_book))
	{
		p->amount_of_copies += new_book.amount_of_copies;
		p->copies_available += new_book.amount_of_copies;
		if (new_book.cipher[0] == '1')
		{
			Book::id_detective--;
		}

		else if (new_book.cipher[0] == '2')
		{
			Book::id_fantasy--;
		}

		else
		{
			Book::id_drama--;
		}

		return p;
	}

	else
	{
		if (new_book.cipher < p->cipher)
```

```
		{
			p->left = Insert(p->left, new_book);
		}

		else
		{
			p->right = Insert(p->right, new_book);
		}
	}

	return balance(p);
}

void TreePrint(Book* p, int& number)
{
	if (p != nullptr)
	{
		TreePrint(p->left, number);
		number++;
		cout << setw(2) << number << "|" << setw(7) << p->cipher << "|" << setw(29) <<
p->title << "|" << setw(49) << p->aouthor << "|" << setw(19) << p->amount_of_copies << "|" <<
setw(19) << p->copies_available << "|" << endl;
		TreePrint(p->right, number);
	}
}

int tabs = 0;
void TreePrintBeatiful(Book* p)
{
	if (p != NULL)
	{
		tabs += 5;
		TreePrintBeatiful(p->right);
		for (int i = 0; i < tabs; i++) cout << " ";

		cout << p->cipher << endl;

		TreePrintBeatiful(p->left);
		tabs -= 5;
	}
}

void TreeInFile(Book* p, ofstream& fout)
{
	if (p != nullptr)
	{
```

```
		TreeInFile(p->left, fout);
		fout << p->cipher << "\n";
		fout << p->aouthor << "\n";
		fout << p->title << "\n";
		fout << p->publishing << "\n";
		fout << p->year_of_publishing << "\n";
		fout << p->amount_of_copies << "\n";
		fout << p->copies_available << "\n";
		TreeInFile(p->right, fout);
	}
}

Book* FindBookByCipher(Book* tmp, string cipher)
{
	while (tmp != nullptr)
	{
		if (tmp->cipher == cipher)
		{
			return tmp;
		}

		if (tmp->cipher > cipher)
		{
			tmp = tmp->left;
		}

		else
		{
			tmp = tmp->right;
		}
	}

	return nullptr;
}

void BooksToFile(Book* p)
{
	ofstream fout;
	fout.open(path_to_books);

	if (!fout.is_open())
	{
		cout << "Ошибка открытия файла" << endl;
		return;
	}
```

```
	TreeInFile(p, fout);
	fout.close();
}

void BooksOutFile(Book*& p)
{
	ifstream fin;
	fin.open(path_to_books);

	if (!fin.is_open())
	{
		cout << "Ошибка открытия файла" << endl;
		return;
	}

	Book new_book;
	int max_id_detective = -1;
	int max_id_fantasy = -1;
	int max_id_drama = -1;
	int tmp_id_detective = -2;
	int tmp_id_fantasy = -2;
	int tmp_id_drama = -2;

	while (!fin.eof())
	{
		new_book.SetZeroes();
		fin >> new_book.cipher;
		fin.get();
		getline(fin, new_book.aouthor);

		if (new_book.aouthor == "\0")
		{
			break;
		}

		getline(fin, new_book.title);
		getline(fin, new_book.publishing);
		fin >> new_book.year_of_publishing;
		fin >> new_book.amount_of_copies;
		fin >> new_book.copies_available;

		if (new_book.cipher[0] == '1')
		{
			tmp_id_detective = (int(new_book.cipher[4]) - 48) * 100 +
(int(new_book.cipher[5]) - 48) * 10 + (int(new_book.cipher[6]) - 48) * 1;
```

```
                        if (tmp_id_detective > max_id_detective)
                        {
                                max_id_detective = tmp_id_detective;
                        }
                }

                if (new_book.cipher[0] == '2')
                {
                        tmp_id_fantasy = (int(new_book.cipher[4]) - 48) * 100 +
(int(new_book.cipher[5]) - 48) * 10 + (int(new_book.cipher[6]) - 48) * 1;

                        if (tmp_id_fantasy > max_id_fantasy)
                        {
                                max_id_fantasy = tmp_id_fantasy;
                        }
                }

                if (new_book.cipher[0] == '3')
                {
                        tmp_id_drama = (int(new_book.cipher[4]) - 48) * 100 +
(int(new_book.cipher[5]) - 48) * 10 + (int(new_book.cipher[6]) - 48) * 1;

                        if (tmp_id_drama > max_id_drama)
                        {
                                max_id_drama = tmp_id_drama;
                        }
                }

                p = Insert(p, new_book);
        }
        fin.close();

        Book::id_detective = max_id_detective + 1;
        Book::id_fantasy = max_id_fantasy + 1;
        Book::id_drama = max_id_drama + 1;

        system("cls");
        return;
}

void ClearWholeTree(Book* p)
{
        if (p != nullptr)
        {
                ClearWholeTree(p->left);
                ClearWholeTree(p->right);
```

```
		delete p;
	}
}

Book* FindMinCipher(Book* p) // поиск узла с минимальным ключом в дереве p
{
	return p->left ? FindMinCipher(p->left) : p;
}

Book* ClearMinCipher(Book* p) // удаление узла с минимальным ключом из дерева p
{
	if (p->left == 0)
	{
		return p->right;
	}
	p->left = ClearMinCipher(p->left);
	return balance(p);
}

bool CanDeleteBook(Taken* start, string cipher)
{
	while (start != nullptr)
	{
		if (start->book_cipher == cipher and start->date_when_returned == "У читателя")
		{
			return false;
		}
		start = start->next;
	}
	return true;
}

Book* RemoveBookFromTree(Book* p, string cipher, Taken* start)//удаление cipher из дерева p
{
	if (!p)
	{
		cout << endl << "Книги с заданным шифром нет в базе библиотеки" << endl;
		return nullptr;
	}

	if (cipher < p->cipher)
	{
		p->left = RemoveBookFromTree(p->left, cipher, start);
	}

	else if (cipher > p->cipher)
```

```
	{
		p->right = RemoveBookFromTree(p->right, cipher, start);
	}

	else
	{
		if (CanDeleteBook(start, cipher))
		{
			Book* q = p->left;
			Book* r = p->right;
			delete p;

			cout << "Данные о книге успешно удалены" << endl;

			if (!r) return q;

			Book* min = FindMinCipher(r);
			min->right = ClearMinCipher(r);
			min->left = q;

			return balance(min);
		}

		else
		{
			cout << endl << "Невозможно удалить данные о книге. Есть экземпляры
книги, не возвращенные в библиотеку. Удаление отменено" << endl;
		}
	}
	return balance(p);
}

void BoyerMoor(Book* p, string txt, string substring, int chose)
{
	int length_of_substring = substring.length();
	int length_of_txt = txt.length();

	if (length_of_substring > length_of_txt)
	{
		return;
	}

	int* pattern = new int[length_of_substring];
	for (int i = 0; i < length_of_substring; i++) //для сравнение сдвигов
	{
		pattern[i] = 100;
```

```
	}
	pattern[length_of_substring - 1] = length_of_substring;

	for (int i = length_of_substring - 2; i >= 0; i--) //определение сдвигов всех символов подстроки кроме последнего
	{
		int shift = length_of_substring - i - 1;

		if (shift < pattern[i])
		{
			pattern[i] = shift;
		}

		for (int j = i - 1; j >= 0; j--)
		{
			if (substring[j] == substring[i])
			{
				pattern[j] = pattern[i];
			}
		}
	}

	for (int i = 0; i < length_of_substring - 1; i++) //определение сдвига последнего символа
	{
		if (substring[i] == substring[length_of_substring - 1])
		{
			pattern[length_of_substring - 1] = pattern[i];
		}
	}

	int spreadsheet_ascii[ascii_size];
	for (int i = 0; i < ascii_size; i++) //заполнение таблицы аски сдвигом = длине подстроки
	{
		spreadsheet_ascii[i] = length_of_substring;
	}

	for (int i = 0; i < length_of_substring; i++) //заполнение аски таблицы сдвигами из образа
	{
		int code_sym = substring[i];
		if (code_sym < 0)
		{
			code_sym += 256;
		}
```

```
        spreadsheet_ascii[code_sym] = pattern[i];
}

int k = length_of_substring - 1;
bool last_sym_equal = false;
bool word_found;
while (k <= length_of_txt - 1)
{
        int for_compare = k;
        last_sym_equal = false;
        word_found = true;

        for (int j = length_of_substring - 1; j >= 0; j--)
        {
                if (txt[for_compare] == substring[j])
                {
                        last_sym_equal = true;
                        for_compare--;
                }

                else
                {
                        word_found = false;
                        if (last_sym_equal == false)
                        {
                                int cod = int(txt[k]);
                                if (cod < 0)
                                {
                                        cod += 256;
                                }
                                k += spreadsheet_ascii[cod];
                                break;
                        }

                        else
                        {
                                int cod = int(substring[length_of_substring -1]);
                                if (cod < 0)
                                {
                                        cod += 256;
                                }

                                k += spreadsheet_ascii[cod];
                                break;
                        }
                }
```

```
			}

			if (word_found)
			{
				cout << setw(7) << p->cipher << "|" << setw(29) << p->title << "|" << 
setw(49) << p->aouthor << "|" << setw(19) << p->amount_of_copies << "|" << setw(19) << p-
>copies_available << "|" << endl;
				delete[] pattern;
				return;
			}
		}
		delete[] pattern;
		return;
}

void FindBookByAouthorOrTitle(Book* p, string substring, int chose)
{
		if (p != nullptr)
		{
			FindBookByAouthorOrTitle(p->left, substring, chose);
			if (chose == 1)
			{
				BoyerMoor(p, p->aouthor, substring, chose);
			}
			else
			{
				BoyerMoor(p, p->title, substring, chose);
			}
			FindBookByAouthorOrTitle(p->right, substring, chose);
		}
}

void BookIssuance(Taken*& start, Taken*& end, Reader* spreadsheet_of_readers, Book* p)
{
		// Нахождение читателя и книги. проверка наличия книги в библиотеке
		cout << endl << "Выдача новой книги читателю: " << endl << endl;

		string card;
		ChekCard(card, "Введите номер читательского билета читателя, которому хотите выдать 
книгу: ");
		int adress = FindReaderByCard(spreadsheet_of_readers, card);
		if (adress == size_of_spreadsheet)
		{
			cout << endl << "Читатель с данным номером билета не найден. Выдача книги 
отменена" << endl;
			return;
```

```
	}

	string cipher;
	ChekCipher(cipher, "Введите шифр книги формата NNN.MMM (N - номер тематического раздела, M - порядковый номер книги в разделе): ");
	Book* book = FindBookByCipher(p, cipher);
	if (book == nullptr)
	{
		cout << endl << "Книги с данным шифром нет в базе данных библиотеки. Выдача книги отменена" << endl;
		return;
	}

	if (book->copies_available == 0)
	{
		cout << endl << "Книги с данным шифром в настоящий момент отсутствуют в библиотеке, приходите позже. (выдача книги отменена)" << endl;
		return;
	}

	////////////////////
	// Регистрация выдачи
	string date_taken = ChekDate();
	if (start == nullptr)
	{
		start = new Taken(card, cipher, date_taken, "У читателя");
		end = start;
		cout << endl << "Книга выдана" << endl;
		book->copies_available--;
		return;
	}

	else
	{
		Taken* tmp = start;
		while (tmp != nullptr)
		{
			if (tmp->library_card == card and tmp->book_cipher == cipher and tmp->date_when_returned == "У читателя")
			{
				cout << "Данная книга уже есть у читателя. В выдаче такой же книги читателю отказано" << endl;
				return;
			}

			else
```

```
                        {
                                tmp = tmp->next;
                        }
                }

                end->next = new Taken(card, cipher, date_taken, "У читателя");
                end = end->next;
                cout << "Книга выдана" << endl << endl;
                book->copies_available--;
                return;
        }
}

void ShowRegistrationHistory(Taken* start)
{
        int N = 0;
        cout << endl << setw(3) << "N|" << setw(13) << "Шифр книги|" << setw(29) << "Номер
читательского билета|" << setw(14) << "Дата выдачи|" << setw(14) << "Дата приема|" << endl;
        cout << "--|------------|---------------------------|-------------|-------------|" <<
endl;
        while (start != nullptr)
        {
                N++;
                cout << setw(2) << N << "|" << setw(12) << start->book_cipher << "|" << setw(28)
<< start->library_card << "|" << setw(13) << start->date_when_taken << "|" << setw(13) <<
start->date_when_returned << "|" << endl;
                start = start->next;
        }
}

void FindBooksInRegistrationHistory(Taken* start, string card)
{
        cout << setw(3) << "N|" << setw(8) << "Шифр|" << endl;
        cout << "--|-------|" << endl;
        int N = 0;
        while (start != nullptr)
        {
                if (start->library_card == card and start->date_when_returned == "У читателя")
                {
                        N++;
                        cout << setw(2) << N << "|" << setw(7) << start->book_cipher << "|" <<
endl;
                }
                start = start->next;
        }
}
```

```
void FindReaderInRegistrationHistory(Taken* start, Reader* spreadsheet_of_readers, string
cipher)
{
	cout << setw(3) << "N|" << setw(32) << "Номер читательского билета|" << setw(50) <<
"ФИО читателя|" << endl;
	cout << "--|------------------------------|-----------------------------------------
------|" << endl;
	int N = 0;
	int adress;
	while (start != nullptr)
	{
		if (start->book_cipher == cipher and start->date_when_returned == "У читателя")
		{
			adress = FindReaderByCard(spreadsheet_of_readers, start->library_card);
			N++;
			cout << setw(2) << N << "|" << setw(31) <<
spreadsheet_of_readers[adress].library_card_number << "|" << setw(49) <<
spreadsheet_of_readers[adress].FIO << "|" << endl;
		}
		start = start->next;
	}
}

void BookReception(Taken* start, Reader* spreadsheet_of_readers, Book* p)

{
	cout << endl << "Прием книги от читателя: " << endl << endl;

	string card;
	Taken* tmp = start;
	ChekCard(card, "Введите номер читательского билета читателя, который возвращает книгу:
");
	int adress = FindReaderByCard(spreadsheet_of_readers, card);

	if (adress == size_of_spreadsheet)
	{
		cout << "Читатель с данным номером билета не зарегистрирован в библиотеке" <<
endl;
		return;
	}

	cout << "Список книг выданных читателю: " << endl << endl;
	FindBooksInRegistrationHistory(start,
spreadsheet_of_readers[adress].library_card_number);
```

```
	string cipher;
	cout << endl;
	ChekCipher(cipher, "Введите шифр книги формата NNN.MMM (N - номер тематического раздела, M - порядковый номер книги в разделе): ");
	Book* book = FindBookByCipher(p, cipher);

	if (book == nullptr)
	{
		cout << endl << "Книги с данным шифром нет в базе данных библиотеки. Прием книги отменен" << endl;
		return;
	}

	while (tmp != nullptr)
	{
		if (tmp->library_card == card and tmp->book_cipher == cipher and tmp->date_when_returned == "У читателя")
		{
			break;
		}
		tmp = tmp->next;
	}

	if (tmp == nullptr)
	{
		cout << endl << "У читателя нет книги с данным шифром" << endl;
		return;
	}


	cout << "Дата получение книги: " << tmp->date_when_taken << endl;
	string date_returned;
	int return_less;
	while (true)
	{

		return_less = 1;
		cout << endl << "Введите дату возвращения книги" << endl;
		date_returned = ChekDate();
		for (int i = 6; i != 10; i++)
		{
			if (tmp->date_when_taken[i] > date_returned[i])
			{
				return_less = 0;
				break;
			}
```

```
        if (tmp->date_when_taken[i] < date_returned[i])
        {
                return_less = 2;
                break;
        }

}

if (return_less == 0)
{
        cout << "Нельзя вернуть книгу раньше, чем ее получить" << endl;
        continue;
}

if (return_less == 2)
{
        break;
}

for (int i = 3; i != 5; i++)
{
        if (tmp->date_when_taken[i] > date_returned[i])
        {
                return_less = 0;
                break;
        }

        if (tmp->date_when_taken[i] < date_returned[i])
        {
                return_less = 2;
                break;
        }

}

if (return_less == 0)
{
        cout << "Нельзя вернуть книгу раньше, чем ее получить" << endl;
        continue;
}

if (return_less == 2)
{
        break;
}
```

```
		for (int i = 0; i != 2; i++)
		{
			if (tmp->date_when_taken[i] > date_returned[i])
			{
				return_less = 0;
				break;
			}

			if (tmp->date_when_taken[i] < date_returned[i])
			{
				break;
			}

		}

		if (return_less == 0)
		{
			cout << "Нельзя вернуть книгу раньше, чем ее получить" << endl;
			continue;
		}

		break;
	}

	tmp->date_when_returned = date_returned;
	book->copies_available++;
	cout << "Книга успешно возвращена в билиотеку" << endl;
}

void RegistrationHistoryInFile(Taken* start)
{
	ofstream fout;
	fout.open(path_to_registration);

	if (!fout.is_open())
	{
		cout << "Ошибка открытия файла" << endl;
		return;
	}

	while (start != nullptr)
	{
		fout << start->library_card << "\n";
		fout << start->book_cipher << "\n";
		fout << start->date_when_taken << "\n";
```

```
			fout << start->date_when_returned << "\n";
			start = start->next;
		}

		fout.close();
}

void RegistrationHistoryOutFile(Taken*& start, Taken*& end)
{
		ifstream fin;
		fin.open(path_to_registration);

		if (!fin.is_open())
		{
			cout << "Ошибка открытия файла" << endl;
			return;
		}

		string card, cipher, date_taken, date_return;
		while (!fin.eof())
		{
			card = cipher = date_taken = date_return = "\0";
			fin >> card;
			fin >> cipher;

			if (cipher == "\0")
			{
				break;
			}

			fin >> date_taken;
			fin.get();
			getline(fin, date_return);

			if (start == nullptr)
			{
				start = new Taken(card, cipher, date_taken, date_return);
				end = start;
			}

			else
			{
				end->next = new Taken(card, cipher, date_taken, date_return);
				end = end->next;
			}
		}
```

```
	fin.close();
}

void ClearWholeRegistrationHistory(Taken*& start, Taken*& end)
{
	Taken* tmp = start;
	while (start != nullptr)
	{
		start = start->next;
		delete tmp;
		tmp = start;
	}
	start = end = nullptr;
}

void AddToPocket(list** pocket, Taken& data, int digit)
{
	list* node = new list(data, nullptr);

	if (pocket[digit] == nullptr)
	{
		pocket[digit] = node;
	}

	else
	{
		list* tmp = pocket[digit];
		while (tmp->next != nullptr)
		{
			tmp = tmp->next;
		}
		tmp->next = node;
	}
}

void Sort(Taken* arr, int amount_of_array)
{
	list* pocket[10]{ nullptr };
	int digit;

	for (int i = 1; i <= k; i++)
	{
		int degree = pow(10, i - 1);
		for (int j = 0; j < amount_of_array; j++)
		{
```

```
                digit = (arr[j].int_cipher / degree) % 10;
                AddToPocket(pocket, arr[j], digit);
        }

        int number_in_array = 0;
        for (int j = 0; j < 10; j++)
        {
                while (pocket[j] != nullptr)
                {
                        arr[number_in_array] = pocket[j]->data;
                        list* tmp = pocket[j];
                        pocket[j] = pocket[j]->next;
                        number_in_array++;
                        delete tmp;
                }
        }
    }
}

void DistributionSort(Taken*& start, Taken*& end)
{
    Taken* tmp = start;
    int amount_in_list = 0;
    while (tmp != nullptr)
    {
        amount_in_list++;
        tmp = tmp->next;
    }

    if (amount_in_list < 1)
    {
        cout << endl << "Сортировать нечего" << endl;
        return;
    }

    Taken* registration_arr = new Taken[amount_in_list];
    tmp = start;

    for (int i = 0; i < amount_in_list; i++)
    {
        registration_arr[i].library_card = tmp->library_card;
        registration_arr[i].book_cipher = tmp->book_cipher;
        registration_arr[i].date_when_taken = tmp->date_when_taken;
        registration_arr[i].date_when_returned = tmp->date_when_returned;
        registration_arr[i].int_cipher = (int(tmp->book_cipher[0]) - 48) * pow(10, 6) +
(long long(tmp->book_cipher[1]) - 48) * pow(10, 5) + (long long(tmp->book_cipher[2]) - 48) *
```

```
pow(10, 4) + 0 * pow(10, 3) + (long long(tmp->book_cipher[4]) - 48) * pow(10, 2) + (long
long(tmp->book_cipher[5]) - 48) * pow(10, 1) + (long long(tmp->book_cipher[6]) - 48) * pow(10,
0);

		tmp = tmp->next;
	}

	ClearWholeRegistrationHistory(start, end);
	Sort(registration_arr, amount_in_list);

	for (int i = 0; i < amount_in_list; i++)
	{
		if (start == nullptr)
		{
			start = new Taken(registration_arr[i].library_card,
registration_arr[i].book_cipher, registration_arr[i].date_when_taken,
registration_arr[i].date_when_returned);
			end = start;
		}

		else
		{
			end->next = new Taken(registration_arr[i].library_card,
registration_arr[i].book_cipher, registration_arr[i].date_when_taken,
registration_arr[i].date_when_returned);
			end = end->next;
		}
	}
	cout << endl << "История регистраций отсортирована" << endl;
}

int main()
{
	setlocale(LC_ALL, "ru");
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	int variant = 0;
	Reader* spreadsheet_of_readers = new Reader[size_of_spreadsheet];
	ReadersFromFile(spreadsheet_of_readers);

	Book* root = nullptr;
	BooksOutFile(root);

	Taken* start = nullptr;
	Taken* end = nullptr;
```

```
	RegistrationHistoryOutFile(start, end);

	while (variant != 17)
	{
		Menu();
		cout << endl;
		variant = ChekInt("Выберете пункт меню: ");

		switch (variant)
		{

		case 1:
		{
			int amount_of_readers;
			cout << endl;
			amount_of_readers = ChekInt("Введите количество новых читателей, которых
хотите занести в базу данных библиотеки: ");
			cout << endl;
			for (int i = 0; i < amount_of_readers; i++)
			{
				Reader new_reader;
				InfoAboutReader(new_reader);
				int hashed_card = HashFunction(new_reader.library_card_number);
				FillHashSpreadsheet(spreadsheet_of_readers, new_reader,
hashed_card);
			}

			if (amount_of_readers != 0)
			{
				cout << "Данные о новых читателях успешно добавлены" << endl <<
endl;
			}
			break;
		}

		case 2:
		{
			ShowSpreadsheet(spreadsheet_of_readers);
			break;
		}

		case 3:
		{
			FindReaderByFio(spreadsheet_of_readers);
			break;
		}
```

```
		case 4:
		{
			ShowSpreadsheet(spreadsheet_of_readers);
			string card;
			cout << endl << "Введите номер читательского билета в виде ANNNN-YY" <<
endl;
			cout << "A - права доступа (A - абонемент, Ч - читальный зал, B -
читальный зал и абонемент" << endl;
			cout << "N - порядковый номер регистрации (цифры), YY - последние две
цифры года рождения читателя" << endl << endl;
			ChekCard(card, "Ввод читательского билета: ");
			int adress = FindReaderByCard(spreadsheet_of_readers, card);

			if (adress != size_of_spreadsheet)
			{
				cout << endl << "Читатель найден: " << endl;
				cout << setw(3) << "N|" << setw(32) << "Номер читательского
билета|" << setw(50) << "ФИО читателя|" << setw(15) << "Год рождения|" << setw(50) << "Адрес|"
<< setw(50) << "Место работы/учебы|" << endl;
				cout << "--|------------------------------|--------------------
----------------------------|--------------|-------------------------------------------------
|-------------------------------------------------|" << endl;
				cout << setw(2) << 1 << "|" << setw(31) <<
spreadsheet_of_readers[adress].library_card_number << "|" << setw(49) <<
spreadsheet_of_readers[adress].FIO << "|" << setw(14) <<
spreadsheet_of_readers[adress].Year_of_birth << "|" << setw(49) <<
spreadsheet_of_readers[adress].adress << "|" << setw(49) <<
spreadsheet_of_readers[adress].workplace << "|" << endl << endl;

				cout << "Список книг выданных читателю: " << endl << endl;
				FindBooksInRegistrationHistory(start,
spreadsheet_of_readers[adress].library_card_number);
			}

			else
			{
				cout << endl << "Читатель с данным номером не найден" << endl;
			}

			break;
		}

		case 5:
		{
			DeleteReader(spreadsheet_of_readers, start);
```

```
			break;
		}

		case 6:
		{
			RemoveWholeInfoAboutReaders(spreadsheet_of_readers, start);
			break;
		}

		case 7:
		{
			int amount_of_books;
			cout << endl;
			amount_of_books = ChekInt("Введите количество авторов, книги которых
хотите занести в базу данных библиотеки: ");
			cout << endl;
			for (int i = 0; i < amount_of_books; i++)
			{
				Book new_book;
				InfoAboutBook(new_book);
				root = Insert(root, new_book);
				cout << endl;
			}

			if (amount_of_books != 0)
			{
				cout << "Данные о новых книгах успешно добавлены" << endl <<
endl;
			}

			break;
		}

		case 8:
		{
			int number = 0;
			cout << endl << "База книг библиотеки: " << endl << endl;

			cout << setw(3) << "N|" << setw(8) << "Шифр|" << setw(30) << "Название
книги|" << setw(50) << "ФИО автора|" << setw(20) << "кол-во копий всего|" << setw(20) <<
"осталось копий|" << endl;
			cout << "--|-------|----------------------------|----------------------
--------------------------|------------------|------------------|" << endl;
			TreePrint(root, number);

			cout << endl << endl << endl;
```

```
                TreePrintBeatiful(root);
                break;
        }

        case 9:
        {
                int chose_for_find = ChekInt("Введите по какому параметру вести поиск (1 - по фрагментам ФИО автора, 2 - по фрагментам названия): ");
                string substring;
                cout << endl;

                if (chose_for_find == 1)
                {
                        ChekSubstring(substring, "Введите автора: ");
                }

                else
                {
                        ChekSubstring(substring, "Введите название книги: ");
                }

                cout << endl << setw(8) << "Шифр|" << setw(30) << "Название книги|" << setw(50) << "ФИО автора|" << setw(20) << "кол-во копий всего|" << setw(20) << "осталось копий|" << endl;
                cout << "-------|------------------------------|--------------------------------------------------|--------------------|--------------------|" << endl;
                FindBookByAouthorOrTitle(root, substring, chose_for_find);
                break;
        }

        case 10:
        {
                string cipher;
                ChekCipher(cipher, "Введите шифр книги формата NNN.MMM (N - номер тематического раздела, M - порядковый номер книги в разделе): ");
                Book* tmp = FindBookByCipher(root, cipher);
                if (tmp != nullptr)
                {
                        cout << endl << "Найденная книга:" << endl;
                        cout << setw(3) << "N|" << setw(8) << "Шифр|" << setw(30) << "Название книги|" << setw(50) << "ФИО автора|" << setw(21) << "Издательство|" << setw(19) << "Год издательства|" << setw(20) << "кол-во копий всего|" << setw(20) << "осталось копий|" << endl;
                        cout << "--|-------|------------------------------|--------------------------------------------------|---------------------|-------------------|--------------------|" << endl;
```

```
                        cout << setw(2) << 1 << "|" << setw(7) << tmp->cipher << "|" <<
setw(29) << tmp->title << "|" << setw(49) << tmp->aouthor << "|" << setw(20) << tmp-
>publishing << "|" << setw(18) << tmp->year_of_publishing << "|" << setw(19) << tmp-
>amount_of_copies << "|" << setw(19) << tmp->copies_available << "|" << endl << endl;

                        cout << "Список читателей, кому выдан данный экземпляр:" << endl
<< endl;
                        FindReaderInRegistrationHistory(start, spreadsheet_of_readers,
tmp->cipher);
                }

                else
                {
                        cout << "Книга с данным шифром не найдена" << endl;
                }

                break;
        }

        case 11:
        {
                string cipher;
                ChekCipher(cipher, "Введите шифр книги формата NNN.MMM (N - номер
тематического раздела, M - порядковый номер книги в разделе): ");
                root = RemoveBookFromTree(root, cipher,start);

                break;
        }

        case 12:
        {
                Taken* tmp = start;
                bool can_clear = true;

                while (tmp != nullptr)
                {
                        if (tmp->date_when_returned == "У читателя")
                        {
                                can_clear = false;
                                break;
                        }
                        tmp = tmp->next;
                }

                if (can_clear)
                {
```

```
                        ClearWholeTree(root);
                        root = nullptr;
                        cout << endl << "База данных книг библиотеки полностью отчищена"
<< endl;
                }

                else
                {
                        cout << endl << "Базу данных книг невозможно отчистить, так как
некоторые книги не возвращены в библиотеку" << endl;
                }

                break;
        }

        case 13:
        {
                ShowSpreadsheet(spreadsheet_of_readers);
                cout << endl;

                cout << endl << "База книг библиотеки: " << endl << endl;
                int number = 0;
                cout << setw(3) << "N|" << setw(8) << "Шифр|" << setw(30) << "Название
книги|" << setw(50) << "ФИО автора|" << setw(20) << "кол-во копий всего|" << setw(20) <<
"осталось копий|" << endl;
                cout << "--|-------|----------------------------|----------------------
--------------------------|------------------|------------------|" << endl;
                TreePrint(root, number);

                BookIssuance(start, end, spreadsheet_of_readers, root);

                break;
        }

        case 14:
        {
                ShowSpreadsheet(spreadsheet_of_readers);
                cout << endl;

                cout << endl << "База книг библиотеки: " << endl << endl;
                int number = 0;
                cout << setw(3) << "N|" << setw(8) << "Шифр|" << setw(30) << "Название
книги|" << setw(50) << "ФИО автора|" << setw(20) << "кол-во копий всего|" << setw(20) <<
"осталось копий|" << endl;
                cout << "--|-------|----------------------------|----------------------
--------------------------|------------------|------------------|" << endl;
```

```
                TreePrint(root, number);

                BookReception(start, spreadsheet_of_readers, root);
                break;
        }

        case 15:
        {
                DistributionSort(start, end);
                break;
        }

        case 16:
        {
                ShowRegistrationHistory(start);
                break;
        }

        }

        if (variant != 17)
        {
                cout << endl;
                system("pause");
                system("cls");
        }
    }

    WriteReadersInFile(spreadsheet_of_readers);
    delete[] spreadsheet_of_readers;

    BooksToFile(root);
    ClearWholeTree(root);

    RegistrationHistoryInFile(start);

    return 0;
}
```