

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 ЗАДАНЫХ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ПРИ
РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ:
РЕГИСТРАЦИЯ БОЛЬНЫХ В ПОЛИКЛИНИКЕ»

по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	M011		Борисов С.И.
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Содержание:

1. ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ	3 стр
2. ОПИСАНИЕ СТРУКТУР ДАННЫХ	17 стр
3. ОПИСАНИЕ ПРОГРАММЫ	21 стр
4. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ	28 стр
5. ЗАКЛЮЧЕНИЕ	42 стр
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	43 стр
ПРИЛОЖЕНИЕ	44 стр

1. ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

Первичная постановка задания.

Предметная область - «Регистрация больных в поликлинике».

Метод хеширования – «Закрытое хеширование с двойным хешированием».

Метод сортировки – «Шейкерная».

Вид списка – «Слоеный».

Метод обхода дерева – «Симметричный».

Алгоритм поиска слова в тексте – «Прямой».

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 «Регистрация больных в поликлинике» должна осуществлять ввод, хранение, обработку и вывод данных о:

- больных;
- врачах;
- выдаче и возврате направлений к врачу.

Данные о каждом больном должны содержать:

- Регистрационный номер – строка формата «ММ-NNNNNN», где ММ – номер участка (цифры); NNNNNN – порядковый номер (цифры);
- ФИО – строка;
- Год рождения – целое;
- Адрес – строка;
- Место работы (учебы) – строка.

Данные о больных должны быть организованы в виде хеш-таблицы, первичным ключом которой является «Регистрационный номер».

Данные о каждом враче должны содержать:

- ФИО врача – строка длиной до 25 символов, содержащая фамилию врача и его инициалы;
- Должность – строка;
- Номер кабинета – целое;
- График приема – строка.

Данные о врачах должны быть организованы в виде AVL-дерева поиска, упорядоченного по «ФИО врача».

Данные о выдаче или возврате направлений к врачу должны содержать:

- Регистрационный номер – строка, формат которой соответствует аналогичной строке в данных о больных; –
- ФИО врача – строка, формат которой соответствует аналогичной строке в данных о врачах;
- Дату направления – строка;
- Время направления – строка.

Примечания:

1. Наличие в этих данных записи, содержащей в поле «Регистрационный номер» значение X и в поле «ФИО врача» значение Y, означает выдачу направления больному с регистрационным номером X к врачу с ФИО Y. Отсутствие такой записи означает, что больной с регистрационным номером X не имеет направления к врачу с ФИО Y.
2. К одному врачу могут направляться несколько больных в течение одного дня, но в разное время. Таким образом, могут быть данные, имеющие повторяющиеся значения в некоторых своих полях.

Данные о выдаче или возврате направлений к врачу должны быть организованны в виде списка, который упорядочен по первичному ключу – «ФИО врача».

Информационная система «Регистрация больных в поликлинике» должна осуществлять следующие операции:

- регистрацию нового больного;
- удаление данных о больном;
- просмотр всех зарегистрированных больных;
- очистку данных о больных;
- поиск больного по регистрационному номеру. Результаты поиска – все сведения о найденном больном и ФИО врача, к которому он имеет направление;
- поиск больного по его ФИО. Результаты поиска – список найденных больных с указанием регистрационного номера и ФИО;
- добавление нового врача;
- удаление сведений о враче;
- просмотр всех имеющихся врачей;
- очистку данных о врачах;
- поиск врача по «ФИО врача». Результаты поиска – все сведения о найденном враче, а также ФИО и регистрационные номера больных, которые имеют направление к этому врачу;
- поиск врача по фрагментам «Должность». Результаты поиска – список найденных врачей с указанием ФИО врача, должности, номера кабинета, графика приема;
- регистрацию выдачи больному направления к врачу;
- регистрацию возврата врачом или больным направления к врачу.

Поиск должности по фрагментам «Должности» должен осуществляться путем систематического обхода АВЛ-дерева поиска. При поиске врача по фрагментам «Должности» могут быть заданы как полное наименование

должности врача, так и его часть. Для обнаружения заданного фрагмента в должности врача должен применяться алгоритм поиска слова в тексте, указанный в варианте задания.

Регистрация выдачи направления к врачу на определенную дату и время должна осуществляться только при отсутствии уже выданного направления к этому же врачу на те же дату и время.

При удалении сведений о враче, должны быть учтены и обработаны ситуации, когда к врачу уже есть записанные на прием больные. Аналогичным образом следует поступать и с удалением сведений больных.

Сценарии использования

Предусловие.

Создается 3 структуры данных:

1. **Patient**, содержащая следующие поля:

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

2. **Doctor**, содержащая следующие поля:

- ФИО врача – строка;
- Должность – строка;
- Номер кабинета – целое;
- График приема – строка.

3. **Referral_to_Doctor**, содержащая следующие поля:

- Регистрационный номер пациента – строка;
- ФИО врача – строка;
- Дату направления – строка;
- Время направления – строка.

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 и нажимает Enter. Дальнейшее поведение приложения описано в последующих сценариях.

Сценарии работы консольного приложения для структуры Patient:

Сценарий: Добавление в БД нового пациента поликлиники.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. На экране отображается текст с предложением ввести количество новых пациентов, которые будут добавлены в БД поликлиники. Пользователь вводит данное и нажимает Enter.
3. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
4. Пользователь вводит ФИО пациента 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 пациента 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 пациента 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) пациента 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 номер участка 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 регистрационного номера.
15. Выводится сообщение о том, что данные о больных успешно добавлены.
16. Предлагается нажать любую кнопку для перехода в главное меню.

Сценарий демонстрации БД пациенты в консоли.

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

Сценарий поиска пациента по ФИО.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. На экране отображается список содержимого БД.
3. Пользователь вводит ФИО пациента 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 больных, если никого найти не удалось, то выводится сообщение о не нахождении пациента.
6. Предлагается нажать любую кнопку для перехода в главное меню.

Сценарий поиска пациента по регистрационному номеру.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.

2. На экране отображается список содержимого БД пациенты.
3. Пользователь вводит регистрационный номер пациента 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 пациент с введенным регистрационным номером отсутствует, то приложение выводит об этом сообщение.
5. На экран выводится вся информация о пациенте и враче, к которому он получил направление. Если у пациента нет направлений, то таблица с информацией о враче не заполняется.
6. Предлагается нажать любую кнопку для перехода в главное меню.

Сценарий удаление пациента по ФИО.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. На экране отображается список содержимого БД.
3. Пользователь вводит ФИО пациента 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 пациент имеет запись к врачу, то удаление невозможно, об этом выводится сообщение в консоли. Далее при нажатии любой кнопки пользователь может выйти в главное меню.
6. Если в БД хранится только один схожий пациент с введенным ранее, то приложение выводит его в консоли, далее удаляет. Далее при нажатии любой кнопки пользователь может выйти в главное меню.
7. Если в БД хранится несколько пациентов с похожим ФИО, то приложение выводит в консоль список предполагаемых для удаления пациентов и предлагает удалить одного из них выбрав пункт меню.
8. Пользователь вводит пункт меню 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 пациента напротив выбранного пункта меню. Далее при нажатии любой кнопки пользователь может выйти в главное меню.
11. Если в БД нет введенного пациента, то приложение выводит об этом сообщение. Далее при нажатии любой кнопки пользователь может выйти в главное меню.

Сценарий очистки БД пациенты.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. Приложение выводит сообщение о полной очистке БД пациенты.
3. Если хотя бы один пациент имеет запись к врачу, то удаление невозможно, об этом выводится сообщение в консоли. Далее предлагается выйти в главное меню.

Сценарий: Сохранения БД пациенты в файл формата txt.

1. После завершения работы программы осуществляется проверка формата БД пациенты.
2. БД пациенты сохраняется в файл.

Сценарий: Загрузка БД пациенты из файла формата txt.

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

Сценарии работы консольного приложения для структуры **Doctor:**

Сценарий: Добавление в БД нового врача поликлиники.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.

2. На экране отображается текст с предложением ввести количество новых врачей, которые будут добавлены в БД поликлиники. Пользователь вводит данное и нажимает Enter.
3. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
4. Пользователь вводит ФИО врача 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 врача 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 номер кабинета врача 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 график работы врача и нажимает Enter.
11. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий шаг либо просят повторить ввод данных.
12. Выводится сообщение о том, что данные о врачах добавлены.
13. Предлагается нажать любую кнопку для перехода обратно в меню.

Сценарий демонстрации БД врачей в консоли.

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

Сценарий поиска врача по ФИО.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. На экране отображается список содержимого БД.
3. Пользователь вводит ФИО врача 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 врача по ФИО, то поиск отменяется и приложение предлагает выйти в главное меню.
6. Если удалось найти врача по ФИО и к нему записан пациент, то в консоли выводится заполненные таблицы с информацией о враче и пациенте.
7. Если в БД удалось найти врача по ФИО, но к нему не записан пациент, то выводится таблица с информацией только о враче, далее предлагается выйти в главное меню.

Сценарий поиска пациента по фрагменту должность.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. На экране отображается список содержимого БД.
3. Пользователь вводит фрагмент должности врача 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 врача по фрагменту должность, то в консоли выводится заполненная таблица с информацией о найденном враче.
6. Если не удалось найти врача по фрагменту должность, то в консоли выводится пустая таблица, далее предлагается выйти в главное меню

Сценарий удаление врача по ФИО.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. На экране отображается список содержимого БД.
3. Пользователь вводит ФИО врача 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 удастся найти врача с введенным ранее ФИО и к нему не записан пациент, то приложение выводит сообщение о его удалении. Потом пользователю предлагают выйти в главное меню.
6. Если в БД нет введенного врача, то приложение выводит об этом сообщение и предлагает выйти в главное меню.

Сценарий очистки БД врачи.

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. Если к хотя бы одному врачу имеется направление, то очистка БД отменяется об этом выводится сообщение, иначе очистка отменяется.

Сценарий: Сохранения БД врачи в файл формата txt.

1. После завершения работы программы осуществляется проверка формата БД врачи.
2. БД врачи сохраняется в файл.

Сценарий: Загрузка БД врачи из файла формата txt.

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

Сценарии работы консольного приложения для структуры Referral_to_Doctor:

Сценарий: Выдача направления

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

3. На экране отображается список содержимого БД врачи.
4. Пользователю предлагается ввести регистрационный номер.
5. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
6. Если введенный регистрационный номер отсутствует, то выдача направления отменяется и приложение предлагает выйти в главное меню.
7. Если введенный регистрационный номер есть в БД, то приложение предлагает ввести ФИО врача.
8. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
9. Если введенное ФИО врача отсутствует, то выдача направления отменяется и приложение предлагает выйти в главное меню.
10. Если введенное ФИО врача есть в БД, то приложение предлагает ввести время приема.
11. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
12. Если врач не работает в данные часы, то выдача направления отменяется и приложение предлагает выйти в главное меню.
13. Если врач работает в данные часы, то предлагается ввести дату приема.
14. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
15. Выводится сообщение о выданном направлении, далее предлагается выйти в главное меню.

Сценарий: Просмотр выданных направлений

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

Сценарий: Возврат направления

1. Пользователь запускает приложение, вводит соответствующий номер пункта меню и нажимает Enter.
2. На экране отображается список содержимого БД пациенты.
3. На экране отображается список содержимого БД врачи.
4. Пользователю предлагается ввести регистрационный номер.
5. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
6. Если введенный регистрационный номер отсутствует, то выдача направления отменяется и приложение предлагает выйти в главное меню.
7. Если введенный регистрационный номер есть в БД, то приложение предлагает ввести ФИО врача.
8. Приложение проверяет введенные данные. В зависимости от результата проверки пользователя отправляют на следующий пункт либо просят повторить ввод данных.
9. Если введенное ФИО врача отсутствует, то выдача направления отменяется и приложение предлагает выйти в главное меню.
10. Если введенное ФИО врача 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 удалении направления, далее предлагается выйти в главное меню.

Сценарий: Выход из приложения

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

2 ОПИСАНИЕ СТРУКТУР ДАННЫХ

1. Описание структуры данных для хранения информации о **Patient**.

```
struct Patient
{
    string RegistrationNumber;    // Регистрационный номер пациента
    string FIO;                  // ФИО пациента
    string Address;              // Адрес пациента
    string Place_of_Work_or_Study; // Место работы (учебы) пациента
    int Year_of_birth;            // Год рождения пациента
};
```

Структура **Patient** содержит следующие поля:

- RegistrationNumber - регистрационный номер, строка формата «ММ-NNNNNN», где ММ - номер участка (цифры); NNNNNN - порядковый номер (цифры);
- FIO - ФИО, строка;
- Year_of_birth - Год рождения, целое;
- Address - Адрес, строка;
- Place_of_Work_or_Study - Место работы (учебы), строка.

База данных **Patient** хранится на диске в текстовом файле формата txt.

2. Описание структуры данных для хранения информации о **Doctor**.

```
struct Doctor
{
    string FIO;                  // ФИО врача
    string Position;            // Должность врача
    string Admission_Schedule;  // График приема врача
    int Cabinet_Number;         // № кабинета врача

    unsigned char height;       // высота подтерева
    Doctor* left;               // указатель на левое поддерево
    Doctor* right;              // указатель на правое поддерево
};
```

Структура **Doctor** содержит следующие поля:

- FIO - ФИО врача, строка;
- Position - Должность, строка;
- Cabinet_Number - Номер кабинета, целое;
- Admission_Schedule - График приема, строка.

База данных **Doctor** хранится на диске в текстовом файле формата txt.

3. Описание структуры данных для хранения информации о **Referral_to_Doctor**.

```
struct Referral_to_Doctor
{
    string RegistrationNumber_Patient; // Регистрационный номер пациента
    string FIO_Doctor;                // ФИО врача
    string Referral_time;              // Время направления
    string Referral_date;              // Дата направления

    Referral_to_Doctor* Next;          // Указатель на след. ячейку
    Referral_to_Doctor* Next_block;    // Указатель на след. Блок ячеек
};
```

Структура **Referral_to_Doctor** содержит следующие поля:

- RegistrationNumber_Patient - Регистрационный номер пациента, строка;
- FIO_Doctor - ФИО врача, строка;
- Referral_time - Дату направления, строка;
- Referral_date - Время направления, строка.

База данных **Referral_to_Doctor** 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 0 до 15)
Структура Patient			
RegistrationNumber	string	Регистрационный номер	строка формата «ММ-NNNNNN», где ММ – номер участка (цифры);

			NNNNNN – порядковый номер (цифры)
FIO	string	ФИО	Буквы русского алфавита(до 25 символов), допускается использование знаков(«.», «-»)
Address	string	Адрес	Буквы русского алфавита(не больше 50 символов),цифры, допускается использование знаков(«.», «-», «/»)
Place_of_Work_or_Study	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 [1900;2022]
Структура Doctor			
FIO	string	ФИО	Буквы русского алфавита(до 25 символов), допускается использование знаков(«.», «-»)
Position	string	Должность	Буквы русского алфавита(до 25 символов), допускается использование знаков(«.», «-»)

Admission_Schedule	string	График приема	Целые числа в диапазоне от 8 до 17
Cabinet_Number	int	Номер кабинета	Целые числа в диапазоне от 1 до 24
Структура Referral_to_Doctor			
RegistrationNumber_Patient	string	Регистрационный номер	строка формата «ММ-NNNNNN», где ММ – номер участка (цифры); NNNNNN – порядковый номер (цифры)
FIO_Doctor	string	ФИО врача	Буквы русского алфавита(до 25 символов), допускается использование знаков(«.»), «-»)
Referral_time;	string	Время направления	Числа не выходящие за пределы графика приема врача
Referral_date	string	Дата направления	Год – натуральное число равное 2022 Месяц – натуральное число в диапазоне [1;12] День – натуральное число. Диапазон вводимого числа зависит от введенных ранее параметром Года и Месяца.

3 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
<code>void Menu()</code>	-	-	Вывод меню
Функции внутри структуры Patient			
<code>Patient()</code>	-	-	Установка начальных значений
<code>void SetInfo(string reg, string fio, string ad, string pl, int year)</code>	Данные находящиеся в структуре пациенты	-	Присвоение начальным значениям новые значения
<code>Patient& operator = (const Patient& other)</code>	Константная ссылка на структуру Patient	Ссылку на структуру Patient	Перегрузка оператора «=»
<code>void SetZeroes()</code>	-	-	Установка нулевых значений
Функции внутри структуры Doctor			
<code>Doctor()</code>	-	-	Установка начальных значений
<code>void SetInfo(string fio, string position, string admission_schedule, int cabinet_number)</code>	Данные находящиеся в структуре врачи	-	Присвоение начальным значениям новые значения
<code>void SetInfo(Doctor& other)</code>	Ссылка на структуру Doctor	-	Присвоение начальным значениям новые значения
<code>Doctor& operator = (const Doctor& other)</code>	Константная ссылка на структуру Doctor	Ссылку на структуру Doctor	Перегрузка оператора «=»
<code>void SetZeroes()</code>	-	-	Установка нулевых значений
Функции внутри структуры Referral_to_Doctor			
<code>Referral_to_Doctor()</code>	-	-	Установка начальных значений
<code>Referral_to_Doctor(string RegistrationNumber_Patient, string FIO_Doctor, string Referral_time, string Referral_date)</code>	Данные находящиеся в структуре врачи	-	Присвоение начальным значениям новые значения

<code>Referral_to_Doctor& operator = (const Referral_to_Doctor& other)</code>	Константная ссылка на структуру Referral_to_Doctor	Ссылку на структуру Referral_to_Doctor or	Перегрузка оператора «=»
Внешние функции			
<code>int Int_check(string text)</code>	text - текст запрашивающий у пользователя ввести данное	Введенное данное	Проверка ввода на: 1) целочисленный формат. 2) целочисленный формат и диапазон вводимого данного
<code>void Name_check(string& name, string text)</code>	Name – ссылка на строковое данное ФИО text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод ФИО (строка до 25 символов, допуск символов: «.», «-», первая буква должна быть заглавной)
<code>void Adress_check(string& ad, string text)</code>	ad – ссылка на строковое данное адреса проживания text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод адреса пациента. (строка до 50 символов, допуск символов: «.», «/», «- », первая буква должна быть заглавной)
<code>void Place_check(string& pl, string text)</code>	pl – ссылка на строковое данное места работы(учебы) text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод места работы(учебы) пациента (строка до 50 символов, допуск символов: «.», «/», «- », первая буква должна быть заглавной)
<code>void Filling_Reg_Number(string& RegNumber)</code>	RegNumber - ссылка на строковое данное регистрационного номера	-	Заполнение регистрационного номера
<code>void Reg_Number_check(string& RegNumber, string text)</code>	RegNumber - ссылка на строковое данное регистрационного номера	-	Проверка на ввод регистрационного номера

	text - текст запрашивающий у пользователя ввести данное		
void infoAboutPatient(Patient& new_patient)	new_patient – ссылка на ячейку структуры с данными пациента	-	Заполнение информации о пациенте
int HashFunction(string key)	key - Строковый регистрационный номер передающийся для хеширование	Целое число	Возвращает адрес в хеш-таблице
int HashFunction2(string key)	key - Строковое регистрационный номер передающийся для для хеширование	Целое число	Возвращает адрес в хеш-таблице
bool FillHashSpreadsheet(Patient* array_of_patient, Patient& new_patient, int hashed_key)	array_of_patient -Указатель на хеш-таблицы new_patient - Ссылка на ячейку с данными о пациенте hashed_key – данное для хешированное	Возвращает правду или ложь	Показывает присвоен ключ или нет
void SwowSpreadsheet(Patient* spreadsheet_of_patient)	spreadsheet_of_patient – указатель на массив данных с пациентами	-	Демонстрация хеш-таблицы
void Find_Patient_FIO(Patient* spreadsheet_of_patient)	spreadsheet_of_patient – Указатель на массив данных с пациентами	-	Поиск пациента по ФИО
int Find_Patient_RegNumber(Patient* spreadsheet_of_patient, string Regnumber)	spreadsheet_of_patient – Указатель на хеш-таблицу Regnumber – строковый регистрационный номер	Адрес на регистрационны й номер или размер хеш-таблицы	Поиск пациента по регистрационному номеру
bool Can_delete_Patient(Referral_to_Doctor* start, string Reg_Number)	Start – указатель на начало списка направлений Reg_Number – строковое данное регистрационного номера	Возвращает правду или ложь	Просматривает, есть ли пациент с веденным регистрационным номером в БД направлений
bool Can_delete_Doctor(Referral_to_Doctor* start, string FIO)	Start – указатель на начало списка направлений	Возвращает правду или ложь	Просматривает, есть ли врач с веденным

	ФИО – строковое данное ФИО врача		ФИО в БД направлений
<code>void Clear_Patient_FIO(Patient* spreadsheet_of_patient)</code>	spreadsheet_of_patient – указатель на массив данных с пациентами	-	Удаление пациента по ФИО
<code>void ClearDataBase_Patients(Patient* spreadsheet_of_patient)</code>	spreadsheet_of_patient – указатель на массив данных с пациентами	-	Очистка базы данных пациенты
<code>void Patient_to_File(Patient* spreadsheet_of_patient)</code>	spreadsheet_of_patient – Указатель на массив данных с пациентами	-	Запись БД пациенты в файл
<code>void Patient_from_File(Patient* spreadsheet_of_patient)</code>	spreadsheet_of_patient – указатель на массив данных с пациентами	-	Загрузка БД пациенты из файла
<code>void Position_check(string& position, string text)</code>	Position – ссылка на должность Text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод должности врача(строка до 25 символов, допуск символов: «.», «-», первая буква должна быть заглавной)
<code>void Admission_Schedule_check(string& admission_schedule, string text)</code>	admission_schedule - ссылка на график приема Text - текст запрашивающий у пользователя ввести данное	-	Проверка на ввод графика приема врача (натуральное число от 8 до 17)
<code>void InfoAboutDoctor(Doctor& new_doctor)</code>	new_doctor - Ссылка на ячейку с данными о пациенте	-	Заполнение информации о враче
<code>void Find_Doctor_Position(Doctor* p, string Position)</code>	P – указатель на узел дерева Position – должность	-	Поиск врача по фрагменту должность
<code>Doctor* Find_Doctor_FIO(Doctor* tmp, string FIO)</code>	Tmp - указатель на узел дерева FIO - ФИО	Найденный доктор или nullptr	Поиск врача по ФИО
<code>int height(Doctor* p)</code>	P – указатель на узел дерева	Целое число	Нахождение высоты поддерева
<code>int bfactor(Doctor* p)</code>	P – указатель на узел дерева	Целое число (от -2 до 2)	Нахождение разности высот левого и правого поддеревьев

<code>void fixheight(Doctor* p)</code>	P – указатель на дерево	-	Восстановление корректного поля высоты заданного узла
<code>Doctor* rotate_right(Doctor* p)</code>	P – указатель на узел дерева	указатель на узел дерева	Правый поворот дерева
<code>Doctor* rotate_left(Doctor* q)</code>	q – указатель на узел дерева	Указатель на узел дерева	Левый поворот дерева
<code>Doctor* balance(Doctor* p)</code>	P – указатель на узел дерева	Указатель на узел дерева	Балансировка дерева
<code>Doctor* Insert_Doctor(Doctor* p, Doctor& new_doctor)</code>	P – указатель на узел дерева	На узел для добавления	Добавление врача в структуру дерева
<code>Doctor* find_min(Doctor* p)</code>	P – указатель на узел дерева	Указатель на узел с минимальным данным по которому упорядоченно дерево	Нахождение узла с минимальным ключом в дереве
<code>Doctor* Clear_Doctor_FIO_min(Doctor* p)</code>	P – указатель на узел дерева	Указатель на самый правый лист	Удаление узла с минимальным ключом в дереве
<code>Doctor* Clear_Doctor_FIO(Doctor* p, string FIO)</code>	P – указатель на узел дерева FIO – ФИО врача Removed -	Указатель на узел	Удаление врача по ФИО
<code>void ClearDataBase_Doctor(Doctor* p)</code>	P – указатель на узел дерева	-	Очистка БД врачи
<code>void Show_Doctors_Tree(Doctor* p, int &number)</code>	P – указатель на узел дерева Number – целое число	-	Демонстрация БД врачи в виде таблицы
<code>void tree_print_beautiful(Doctor* p)</code>	P – указатель на узел дерева	-	Демонстрация БД врачи в виде AVL дерева
<code>void Tree_in_File(Doctor* p, ofstream& fout)</code>	P – указатель на узел дерева	-	Запись БД врачи из файла
<code>void Doctors_to_File(Doctor* p)</code>	P – указатель на узел дерева	-	Проверка открытия файла для записи в него БД врачи
<code>void Doctors_from_File(Doctor*& p)</code>	P – указатель на узел дерева	-	Загрузка БД врачи из файла
<code>string ChekDate()</code>	-	дата	Проверка даты

void Clear_List(Referral_to_Doctor*& start)	Start – ссылка на начало списка	-	Очистка списка
void Layer_List(Referral_to_Doctor*& start, Referral_to_Doctor* writes, int amount_of_writes)	Start – ссылка на начало списка Writes – указатель ячейку массива amount_of_writes – кол-во направлений	-	Создание слоеного списка
void Swap(Referral_to_Doctor& first, Referral_to_Doctor& second)	First – ссылка на ячейку с направлением Second – ссылка на ячейку с направлением	-	Перестановка местами направлений
void ShakerSort(Referral_to_Doctor*& start)	Start - ссылка на начало списка	-	Сортировка направлений
void Issuing_Referrals_to_the_Patient(Referral_to_Doctor*& Next, Referral_to_Doctor*& Next_block, Patient* spreadsheet_of_patient, Doctor* p)	Next – ссылка на первую ячейку списка Next_block – ссылка на первый блок ячеек spreadsheet_of_patient – указатель на первый элемент хеш-таблицы p – указатель на узел дерева	-	Выдача направлений
void Return_Referrals_to_the_Patient(Referral_to_Doctor*& start, Patient* spreadsheet_of_patient, Doctor* p)	Start – ссылка на начало списка spreadsheet_of_patient – указатель на первый элемент хеш-таблицы p – указатель на узел дерева	-	Возврат направлений
void Show_Referrals(Referral_to_Doctor*& start)	Start – указатель на начало списка	-	Вывод таблицы с выданными направлениями
void Show_Referrals_Patient(Referral_to_Doctor* start, string Reg_Number)	Start – указатель на начало списка Reg_Number – регистрационный номер	-	Вывод врачей к которым пациент получил направления
void Show_Referrals_Doctors(Referral_to_Doctor* start, string FIO, Patient* spreadsheet_of_patient)	Start – указатель на начало списка FIO – ФИО врача	-	Вывод пациентов направленных к введенному врачу

	spreadsheet_of_patient – указатель на первый элемент хеш-таблицы		
void Referral_to_Doctor_from_File(Referral_to_Doctor*& start)	Start – ссылка на начало списка направлений	-	Загрузка списка направлений из файла
void Referral_to_Doctor_to_File(Referral_to_Doctor* start)	Start – указатель на начало списка направлений	-	Сохранение списка направлений в файл

4 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	Sdf	Ошибка! Повторите ввод!
		Ыва	Ошибка! Повторите ввод!
		4.5	Ошибка! Повторите ввод!
		16	Ошибка! Повторите ввод!
	Корректный ввод пункта меню	0	Завершение программы
2	Регистрация больного	Пункт 1	
	Некорректный ввод количества больных для регистрации	Sdf	Ошибка! Повторите ввод!
		Ыва	Ошибка! Повторите ввод!
		4.5	Ошибка! Повторите ввод!
	Корректный ввод количества больных для регистрации	1	Переход к следующему шагу
	Некорректный ввод ФИО	Sdf	S - Недопустимый символ!
		4.5	4 - Недопустимый символ!
		борисов ИН	[б] - Недопустимый символ!
		Более 25 символов	Ошибка! Повторите ввод
		.борисов ИН	[.] - Недопустимый символ!
		-борисов ИН	[-] - Недопустимый символ!
	Некорректный ввод года рождения	1800	Ошибка! Повторите ввод!
		2222	Ошибка! Повторите ввод!
	Некорректный ввод адреса проживания (функция проверки адреса отличается от функции проверки ФИО только количеством допустимого ввода символов)	Более 50 символов	Ошибка! Повторите ввод
	Некорректный ввод места работы (учебы) (функция проверки места работы ничем не отличается от функции проверки адреса)	-	-
	Некорректный ввод номера участка	22	Ошибка! Повторите ввод!
		Sdf	Ошибка! Повторите ввод!
	Корректная регистрация пациента	Борисов ИН 1966 Самара ул. Челлюскинцев 15 Самарский районный суд 5	Регистрационный номер добавлен. Данные о новых больных добавлены

3	Просмотр списка зарегистрированных больных	Пункт 2	Вывод таблицы с информацией о зарегистрированных больных
4	Поиск больного по ФИО	Пункт 3	
	Ввод ФИО, которого нет в БД	Калашников ИИ	Список найденных больных: Данного пациента в списке нет.
	Ввод ФИО, которое есть в БД	Борисов ИН	Вывод информации о найденном больном
5	Поиск больного по регистрационному номеру	Пункт 4	Вывод таблицы с информацией о зарегистрированных больных
	Ввод регистрационного номера, которого нет в БД	04-000009	Пациент с веденным регистрационным номером отсутствует. Поиск по регистрационному номеру отменен.
	Некорректный ввод регистрационного номера	14-000009	Ошибка! Повторите ввод
		04-0000099	Ошибка! Повторите ввод
		0-4000003	Ошибка! Повторите ввод
	Ввод регистрационного номера, который есть в БД	05-000006	Вывод информации о найденном больном и докторе(если есть направление)
6	Удаление больного по ФИО	Пункт 5	Вывод таблицы с информацией о зарегистрированных больных
	Ввод ФИО, которое нет в БД	Калашников ИИ	Данного пациента в списке нет.
	Ввод ФИО, которое есть в БД	Борисов ИН	Выводятся найденный больной Происходит удаление Выводятся сообщение: Данные о пациенте успешно удалены
	Ввод ФИО, которое повторяется в БД	Жвакин АО	Выводятся найденные больные Предлагается выбрать пункт для удаления больного Выводятся сообщение: Данные о пациенте успешно удалены
	Некорректный ввод пункта напротив больного которого хотим удалить	3	Ошибка ввода! Повторите ввод!
	Ввод ФИО, у которого есть направление к врачу	Борисов ИН	Данному пациенту выдано направление! Удаление невозможно!
7	Очистка базы данных больных	Пункт 6	База данных пациентов полностью очищена
	Если есть хотя бы одно выданное направление	Пункт 6	Базу данных очистить нельзя! Имеются выданные направления!
8	Добавление нового врача	Пункт 7	
	Некорректный ввод количества больных 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 (используется та же функция проверки, что и во втором пункте)	-	-

	Некорректный ввод должности врача (функция проверки должности ничем не отличается функции проверки ФИО во втором пункте)	-	-
	Некорректный ввод номера кабинета врача	0	Ошибка! Повторите ввод!
		29	Ошибка! Повторите ввод!
	Некорректный ввод графика работы врача	7	7 - Недопустимый символ!
		20	2 - Недопустимый символ!
	Корректное добавление врача	Обломов ОГ Патологоанатом 20 8 17	Данные о новых врачах добавлены
9	Просмотр списка врачей	Пункт 8	Вывод таблицы с информацией о добавленных врачах
10	Поиска врача по ФИО	Пункт 9	Вывод БД врачей
	Некорректный ввод ФИО (используется та же функция проверки, что и во втором пункте)	-	-
	ввод ФИО врача, которого нет в БД	Агутин НН	Доктор с веденным ФИО отсутствует. Поиск по ФИО отменен.
	Корректный ввод ФИО врача	Обломов ОГ	Вывод информации о враче и пациенте (если у пациента есть направление к данному врачу)
11	Поиск врача по фрагменту должность	Пункт 10	Вывод БД врачей
	Некорректный ввод фрагмента должность (функция проверки фрагмента должности ничем не отличается функции проверки ФИО во 2 пункте)	-	-
	Ввод фрагмента которого нет в БД	чь	Вывод таблицы с найденными врачами (в данном случае пустой таблицы)
	Ввод фрагмента которого есть в БД	олог	Вывод таблицы с найденными врачами
12	Удаление сведений о враче	Пункт 11	Вывод БД врачей
	Некорректный ввод ФИО (используется та же функция проверки, что и во втором пункте)	-	-
	Ввод ФИО, которого нет в БД	Агутин НН	Доктор не найден.
	Корректный ввод ФИО	Обломов ОГ	Данные о докторе успешно удалены.
	Ввод ФИО врача, к которому есть направление	Казакова ПП	Удаление невозможно! К доктору имеется направление!
13	Очистка базы данных врачей	Пункт 12	База данных врачей полностью очищена
	Если есть хотя бы один врач с выданным на него направлением	Пункт 12	Базу данных очистить нельзя! Имеются выданные направления!
14	Выдача направлений пациенту	Пункт 13	Вывод БД пациентов и врачей

	Ввод регистрационного номера (осуществляются все те же проверки что и в пункте 5)	-	-
	Ввод ФИО врача (осуществляются все те же проверки что и в пункте 6)	-	-
	Ввод времени приема врача (осуществляются все те же проверки что и в пункте 8)	-	-
	Некорректный ввод дня приема у врача	0	Ошибка! Повторите ввод!
		32	Ошибка! Повторите ввод!
	Некорректный ввод месяца приема у врача	0	Ошибка! Повторите ввод!
		15	Ошибка! Повторите ввод!
	Некорректный ввод года приема у врача	2025	Ошибка! Повторите ввод!
	Корректная выдача направления	05-000006 Казакова ПП 10 1 5 2022	Предложение нажать любую клавишу для перехода в главное меню
15	Просмотр выданных направлений	Пункт 14	Вывод информации о выданных направлениях
16	Возврат направления у пациента	Пункт 15	Вывод БД направлений
	Ввод регистрационного номера (осуществляются все те же проверки что и в пункте 5)	-	-
	Ввод ФИО врача (осуществляются все те же проверки что и в пункте 6)	-	-
	Ввод даты приема врача (осуществляются все те же проверки что и в пункте 14)	-	-
	Корректный возврат направления	05-000006 Казакова ПП 1 5 2022	Направление успешно удалено.

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

1. Главное меню:

```

      Меню
-----
Пациенты поликлиники:
Нажмите 1, чтобы зарегистрировать больного.
Нажмите 2, чтобы просмотреть список зарегистрированных больных.
Нажмите 3, чтобы найти больного по ФИО.
Нажмите 4, чтобы найти больного по регистрационному номеру.
Нажмите 5, чтобы удалить больного.
Нажмите 6, чтобы очистить базу данных больных.
-----
Врачи поликлиники:
Нажмите 7, чтобы добавить нового врача.
Нажмите 8, чтобы просмотреть список врачей.
Нажмите 9, чтобы найти врача по ФИО.
Нажмите 10, чтобы найти врача по фрагменту Должность.
Нажмите 11, чтобы удалить сведения о враче.
Нажмите 12, чтобы очистить базу данных врачей.
-----
Нажмите 13, чтобы выдать пациенту направление:
Нажмите 14, чтобы просмотреть выданные направления:
-----
Нажмите 15, чтобы возвратить направление у пациента:
-----
Нажмите 0, чтобы выйти из программы.
-----

Выберете пункт меню: Sdf
Ошибка! Повторите ввод!
Выберете пункт меню: bva
Ошибка! Повторите ввод!
Выберете пункт меню: 4.5
Ошибка! Повторите ввод!
Выберете пункт меню: 16
Ошибка! Повторите ввод!
Выберете пункт меню: 0
Для продолжения нажмите любую клавишу . . .

```

Рисунок 1 – Главное меню

[illegible]

3. Просмотр списка зарегистрированных больных

'Список зарегистрированных пациентов больницы'		
N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

Для продолжения нажмите любую клавишу . . .

Рисунок 3 – Просмотр списка зарегистрированных больных

4. Поиск больного по ФИО

'Список зарегистрированных пациентов больницы'		
N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

'Поиск пациента по ФИО'

Введите ФИО больного: Калашников ИИ

Список найденных больных:

Данного пациента в списке нет.

Для продолжения нажмите любую клавишу . . .

'Список зарегистрированных пациентов больницы'		
N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

'Поиск пациента по ФИО'

Введите ФИО больного: Борисов ИН

Список найденных больных:

N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН

Для продолжения нажмите любую клавишу . . .

Рисунок 4 – Поиск больного по ФИО

5. Поиск больного по регистрационному номеру

'Список зарегистрированных пациентов больницы'		
N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

Введите регистрационный номер пациента: 04-000009

Пациент с введенным регистрационным номером отсутствует. Поиск по регистрационному номеру отменен.

'Список зарегистрированных пациентов больницы'

N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

Введите регистрационный номер пациента: 14-000009

Ошибка! Повторите ввод

Введите регистрационный номер пациента: 04-000009

Ошибка! Повторите ввод

Введите регистрационный номер пациента: 0-4000003

Ошибка! Повторите ввод

Введите регистрационный номер пациента: 05-000006

N	Регистрационный номер	ФИО пациента	Адрес	Место работы(учебы)	Год рождения
1	05-000006	Борисов ИН	Самара ул. Челюскинцев 15	Самарский районный суд	1966

Доктора к которому пациент получил направления:

N	ФИО доктора
---	-------------

Рисунок 5 – Поиск больного по регистрационному номеру

6. Удаление больного по ФИО

'Список зарегистрированных пациентов больницы'

N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

'Удаление пациента по ФИО'

Введите ФИО больного: Калашников ИИ

Данного пациента в списке нет.

Для продолжения нажмите любую клавишу . . .

'Список зарегистрированных пациентов больницы'

N	Регистрационный номер	ФИО пациента
1	04-000007	Жвакин АО
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

'Удаление пациента по ФИО'

Введите ФИО больного: Жвакин АО

В базе хранится несколько пациентов с таким ФИО

N	Регистрационный номер	ФИО пациента
1	04-000007	Жвакин АО
2	06-000004	Жвакин АО

Выберите порядковый номер пациента, которого хотите удалить: 2

Данные о пациенте успешно удалены

'Список зарегистрированных пациентов больницы'

N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

'Удаление пациента по ФИО'

Введите ФИО больного: Борисов ИН

Список найденных больных:

N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН

Данные о пациенте успешно удалены

Для продолжения нажмите любую клавишу . . .

'Список зарегистрированных пациентов больницы'

N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА

'Удаление пациента по ФИО'

Введите ФИО больного: Борисов ИН

Список найденных больных:

N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН

Данному пациенту выдано направление! Удаление невозможно!

Рисунок 6 – Удаление больного по ФИО

7. Очистка базы данных больных

Выберете пункт меню: 6

База данных пациентов полностью очищена

Выберете пункт меню: 6

Базу данных очистить нельзя! Имеются выданные направления!

Рисунок 7 – Очистка базы данных больных

8. Добавление нового врача

```
Введите количество новых врачей, которых хотите занести в базу данных поликлиники: 1
Введите ФИО доктора(до 25 символов): Обломов ОГ
Введите должность доктора(до 25 символов): Патологоанатом
Введите номер кабинета доктора(1-24): 0
Ошибка! Повторите ввод!
Введите номер кабинета доктора(1-24): 29
Ошибка! Повторите ввод!
Введите номер кабинета доктора(1-24): 20
Введите время начала приема (часы): 7
7 - Недопустимый символ!
Введите время начала приема (часы): 8
Введите время окончания приема (часы): 20
2 - Недопустимый символ!
Введите время окончания приема (часы): 17

Данные о новых врачах добавлены
```

Рисунок 8 - Просмотр списка врачей

9. Просмотр списка врачей

```
'Список зарегистрированных докторов больницы'
```

	ФИО доктора	Должность	Н каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	<u>Обломов ОГ</u>	Патологоанатом	20	08:00 - 17:00
7	Шиловский СС	Проктолог	6	08:00 - 17:00
8	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

```

      Анисимов ИИ
    Борисов СИ
    Вишнецкий РР
  Галицкий НН
    Казакова ПП
      Обломов ОГ
    Шиловский СС
      Шишкин ПП

Для продолжения нажмите любую клавишу . . .
```

Рисунок 9 - Просмотр списка врачей

10.Поиска врача по ФИО

'Список зарегистрированных докторов больницы'

N	ФИО доктора	Должность	N каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	Обломов ОГ	Патологоанатом	20	08:00 - 17:00
7	Шиловский СС	Проктолог	6	08:00 - 17:00
8	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

Введите ФИО доктора(до 25 символов): Агутин НН

Доктор с введенным ФИО отсутствует. Поиск по ФИО отменен.
Для продолжения нажмите любую клавишу . . .

'Список зарегистрированных докторов больницы'

N	ФИО доктора	Должность	N каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	Обломов ОГ	Патологоанатом	20	08:00 - 17:00
7	Шиловский СС	Проктолог	6	08:00 - 17:00
8	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

Введите ФИО доктора(до 25 символов): Обломов ОГ

Найденный доктор:

N	ФИО доктора	Должность	N каб.	График приема
1	Обломов ОГ	Патологоанатом	20	08:00 - 17:00

Пациенты, направленные к данному доктору:

N	Регистрационный номер	ФИО пациента
---	-----	-----

Для продолжения нажмите любую клавишу . . .

Рисунок 10 - Поиска врача по ФИО

11.Поиск врача по фрагменту должность

'Список зарегистрированных докторов больницы'

	ФИО доктора	Должность	N каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	Обломов ОГ	Патологоанатом	20	08:00 - 17:00
7	Шиловский СС	Проктолог	6	08:00 - 17:00
8	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

'Поиск доктора по фрагменту Должность'

Введите должность доктора(до 25 символов): чьъ

Найденный доктор:

	ФИО доктора	Должность	N каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	Обломов ОГ	Патологоанатом	20	08:00 - 17:00
7	Шиловский СС	Проктолог	6	08:00 - 17:00
8	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

'Поиск доктора по фрагменту Должность'

Введите должность доктора(до 25 символов): олог

Найденный доктор:

	ФИО доктора	Должность	N каб.	График приема
	Борисов СИ	Стоматолог	2	08:00 - 17:00
	Галицкий НН	Невролог	4	08:00 - 17:00
	Обломов ОГ	Патологоанатом	20	08:00 - 17:00
	Шиловский СС	Проктолог	6	08:00 - 17:00
	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

Для продолжения нажмите любую клавишу . . .

Рисунок 11 - Поиск врача по фрагменту должность

12. Удаление сведений о врачах

'Список зарегистрированных докторов больницы'

	ФИО доктора	Должность	Н каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	Обломов ОГ	Патологоанатом	20	08:00 - 17:00
7	Шиловский СС	Проктолог	6	08:00 - 17:00
8	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

'Удаление доктора по ФИО'

Введите ФИО доктора(до 25 символов): Агутин НН

Доктор не найден.

'Список зарегистрированных докторов больницы'

	ФИО доктора	Должность	Н каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	Обломов ОГ	Патологоанатом	20	10:00 - 08:00
7	Шиловский СС	Проктолог	6	08:00 - 17:00
8	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

'Удаление доктора по ФИО'

Введите ФИО доктора(до 25 символов): Обломов ОГ

Данные о докторе успешно удалены.

'Список зарегистрированных докторов больницы'

	ФИО доктора	Должность	Н каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	Шиловский СС	Проктолог	6	08:00 - 17:00
7	Шишкин ПП	Офтальмолог	7	08:00 - 17:00

'Удаление доктора по ФИО'

Введите ФИО доктора(до 25 символов): Казакова ПП

Удаление невозможно! К доктору имеется направление!

Рисунок 12 - Удаление сведений о врачах

13. Очистка базы данных врачей

Выберете пункт меню: 12

База данных врачей полностью очищена

Выберете пункт меню: 12

Базу данных очистить нельзя! Имеются выданные направления!

Рисунок 13 - Очистка базы данных врачей

14.Выдача направлений пациенту

```
'Список зарегистрированных пациентов больницы'
```

N	Регистрационный номер	ФИО пациента
1	05-000006	Борисов ИН
2	06-000004	Жвакин АО
3	05-000002	Рябцева МО
4	07-000005	Уланский НД
5	01-000000	Бориосв СИ
6	04-000003	Бовыкина ПА


```
'Список зарегистрированных докторов больницы'
```

	ФИО доктора	Должность	N каб.	График приема
1	Анисимов ИИ	Хирург	1	08:00 - 17:00
2	Борисов СИ	Стоматолог	2	08:00 - 17:00
3	Вишнецкий РР	Дантист	3	08:00 - 17:00
4	Галицкий НН	Невролог	4	08:00 - 17:00
5	Казакова ПП	Окулист	8	08:00 - 17:00
6	Шиловский СС	Проктолог	6	08:00 - 17:00
7	Шишкин ПП	Офтальмолог	7	08:00 - 17:00


```
Выдача направления к доктору
```

Введите регистрационный номер пациента: 05-000006

Введите ФИО доктора(до 25 символов): Казакова ПП

Введите время приема: 10

Ввод даты:

Введите день: 0

Ошибка! Повторите ввод!

Введите день: 32

Ошибка! Повторите ввод!

Введите день: 1

Введите месяц: 0

Ошибка! Повторите ввод!

Введите месяц: 15

Ошибка! Повторите ввод!

Введите месяц: 5

Введите год: 2025

Ошибка! Повторите ввод!

Введите год: 2022

Для продолжения нажмите любую клавишу . . .

Рисунок 14 - Выдача направлений пациенту

15.Просмотр выданных направлений

```
Список выданных направлений:
```

N	Регистрационный номер	ФИО доктора	Время приема	Дата приема
1	05-000006	Казакова ПП	10:00	01.05.2022


```
Для продолжения нажмите любую клавишу . . .
```

Рисунок 15 - Просмотр выданных направлений

16. Возврат направления у пациента

Список выданных направлений:

N	Регистрационный номер	ФИО доктора	Время приема	Дата приема
1	05-000006	Казакова ПП	10:00	01.05.2022

Возврат направления к доктору

Введите регистрационный номер пациента: 05-000006

Введите ФИО доктора(до 25 символов): Казакова ПП

Ввод даты:

Введите день: 1

Введите месяц: 5

Введите год: 2022

Направление успешно удалено.

Рисунок 16 - Возврат направления у пациента

5 ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы, была разработана информационная система для регистрации больных в поликлинике. 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. С/С++. Программирование на языке высокого уровня: учебник. СПб.: ПИТЕР, 2007. - 461 с

ПРИЛОЖЕНИЕ

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

using namespace std;

const int size_of_key = 9;
const int size_of_spreadsheet = 100;

string path_to_patient = "D:\\Учеба\\4 семестр\\КП по САОД\\КП по САОД (прога)\\Пациенты.txt";
string path_to_doctors = "D:\\Учеба\\4 семестр\\КП по САОД\\КП по САОД (прога)\\Врачи.txt";
string path_to_referral = "D:\\Учеба\\4 семестр\\КП по САОД\\КП по САОД (прога)\\Направления.txt";

struct Patient
{
    Patient()
    {
        RegistrationNumber = "\\0";
        FIO = "\\0";
        Address = "\\0";
        Place_of_Work_or_Study = "\\0";
        Year_of_birth = 0;
    }

    string RegistrationNumber;    // Регистрационный номер пациента
    string FIO;                  // ФИО пациента
    string Address;              // Адрес пациента
    string Place_of_Work_or_Study; // Место работы (учебы) пациента
    int Year_of_birth;            // Год рождения пациента

    static int id;

    void SetInfo(string reg, string fio, string ad, string pl, int year)
    {
        RegistrationNumber = reg;
        FIO = fio;
        Address = ad;
        Place_of_Work_or_Study = pl;
        Year_of_birth = year;
    }

    void SetZeroes()
```

```

{
    RegistrationNumber = "\0";
    FIO = "\0";
    Address = "\0";
    Place_of_Work_or_Study = "\0";
    Year_of_birth = 0;
}

Patient& operator = (const Patient& other)
{
    this->RegistrationNumber = other.RegistrationNumber;
    this->FIO = other.FIO;
    this->Address = other.Address;
    this->Place_of_Work_or_Study = other.Place_of_Work_or_Study;
    this->Year_of_birth = other.Year_of_birth;
    return *this;
}
};

struct Doctor
{
    Doctor()
    {
        FIO = "\0"; // до 25 символов
        Position = "\0";
        Admission_Schedule = "\0";
        Cabinet_Number = 0;

        left = right = 0;
        height = 1;
    }

    string FIO; // до 25 символов
    string Position;
    string Admission_Schedule; // график приема
    int Cabinet_Number;

    // для дерева:
    unsigned char height;
    Doctor* left;
    Doctor* right;

    void SetInfo(string fio, string position, string admission_schedule, int cabinet_number)
    {
        FIO = fio;
        Position = position;
    }
}

```

```

        Admission_Schedule = admission_schedule;
        Cabinet_Number = cabinet_number;
    }
    void SetInfo(Doctor& other)
    {
        FIO = other.FIO;
        Position = other.Position;
        Admission_Schedule = other.Admission_Schedule;
        Cabinet_Number = other.Cabinet_Number;
    }
    void SetZeroes()
    {
        FIO = "\0";
        Position = "\0";
        Admission_Schedule = "\0";
        Cabinet_Number = 0;

        left = right = 0;
        height = 1;
    }

    Doctor& operator = (const Doctor& other)
    {
        this->FIO = other.FIO;
        this->Position = other.Position;
        this->Admission_Schedule = other.Admission_Schedule;
        this->Cabinet_Number = other.Cabinet_Number;
        return *this;
    }
};

struct Referral_to_Doctor
{
    Referral_to_Doctor()
    {
        RegistrationNumber_Patient = "\0";
        FIO_Doctor = "\0";
        Referral_time = "\0";
        Referral_date = "\0";

        Next = nullptr;
        Next_block = nullptr;
    }

    Referral_to_Doctor(string RegistrationNumber_Patient, string FIO_Doctor, string
Referral_time, string Referral_date)
    {

```

```

        this->RegistrationNumber_Patient = RegistrationNumber_Patient;
        this->FIO_Doctor = FIO_Doctor;
        this->Referral_time = Referral_time;
        this->Referral_date = Referral_date;

        Next = nullptr;
        Next_block = nullptr;
    }

    string RegistrationNumber_Patient;
    string FIO_Doctor;
    string Referral_time;
    string Referral_date;

    //для слоеного списка
    Referral_to_Doctor* Next;
    Referral_to_Doctor* Next_block;

    Referral_to_Doctor& operator = (const Referral_to_Doctor& other)
    {
        this->RegistrationNumber_Patient = other.RegistrationNumber_Patient;
        this->FIO_Doctor = other.FIO_Doctor;
        this->Referral_time = other.Referral_time;
        this->Referral_date = other.Referral_date;
        return *this;
    }
};

void Menu()
{
    system("cls");
    cout << " _____" << endl;
    cout << "                      Меню                      \\" << endl;
    cout << " |-----|" << endl;
    cout << " |Пациенты поликлиники:                |" << endl;
    cout << " |Нажмите 1, чтобы зарегистрировать больного.        |" << endl;
    cout << " |Нажмите 2, чтобы просмотреть список зарегистрированных больных. |" << endl;
    cout << " |Нажмите 3, чтобы найти больного по ФИО.              |" << endl;
    cout << " |Нажмите 4, чтобы найти больного по регистрационному номеру.    |" << endl;
    cout << " |Нажмите 5, чтобы удалить больного.                    |" << endl;
    cout << " |Нажмите 6, чтобы очистить базу данных больных.        |" << endl;
    cout << " |-----|" << endl;
    cout << " |Врачи поликлиники:                |" << endl;
    cout << " |Нажмите 7, чтобы добавить нового врача.              |" << endl;
    cout << " |Нажмите 8, чтобы просмотреть список врачей.          |" << endl;
    cout << " |Нажмите 9, чтобы найти врача по ФИО.                  |" << endl;

```

```

cout << "|Нажмите 10, чтобы найти врача по фрагменту Должность.      |" << endl;
cout << "|Нажмите 11, чтобы удалить сведения о враче.                    |" << endl;
cout << "|Нажмите 12, чтобы очистить базу данных врачей.                  |" << endl;
cout << "|-----|" << endl;
cout << "|Нажмите 13, чтобы выдать пациенту направление:                  |" << endl;
cout << "|Нажмите 14, чтобы просмотреть выданные направления:            |" << endl;
cout << "|-----|" << endl;
cout << "|Нажмите 15, чтобы вернуть направление у пациента:              |" << endl;
cout << "|-----|" << endl;
cout << "|Нажмите 0, чтобы выйти из программы.                            |" << endl;
cout << "\\_____/" << endl <<
endl;
}

int Int_check(string text)
{
    while (true)
    {
        int value;
        cout << text;
        cin >> value;

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

        if (text == "Выберете пункт меню: ")
        {
            if (value < 0 || value > 15)
            {
                cout << "Ошибка! Повторите ввод!" << endl;
                continue;
            }
            return value;
        }

        if (text == "Введите количество новых больных, которых хотите занести в базу данных
поликлиники: ")
        {
            if (value < 0)
            {
                cout << "Ошибка! Повторите ввод!" << endl << endl;

```



```

        continue;
    }
    return value;
}

if (text == "Введите год рождения больного: ")
{
    if (value < 1900 or value > 2022)
    {
        cout << "Ошибка! Повторите ввод!" << endl << endl;
        continue;
    }
    return value;
}

if (text == "Введите номер участка (1-9): ")
{
    if (value < 1 or value > 9)
    {
        cout << "Ошибка! Повторите ввод!" << endl << endl;
        continue;
    }
    return value;
}

if (text == "Введите номер кабинета доктора(1-24): ")
{
    if (value < 1 or value > 24)
    {
        cout << "Ошибка! Повторите ввод!" << endl << endl;
        continue;
    }
    return value;
}

if (text == "Введите день: ")
{
    if (value < 1 or value > 31)
    {
        cout << "Ошибка! Повторите ввод!" << endl << endl;
        continue;
    }
    return value;
}

if (text == "Введите месяц: ")

```

```

{
    if (value < 1 or value > 12)
    {
        cout << "Ошибка! Повторите ввод!" << endl << endl;
        continue;
    }
    return value;
}

if (text == "Введите год: ")
{
    if (value != 2022)
    {
        cout << "Ошибка! Повторите ввод!" << endl << endl;
        continue;
    }
    return value;
}
return value;
}
}

void Name_check(string& name, string text) // Проверка на имя
{
    while (true)
    {
        cout << text;
        getline(cin, name);
        bool incorrect = false;

        if (name.length() > 25)
        {
            cout << "Ошибка! Повторите ввод" << endl;
            continue;
        }

        if (name[0] == '.' or name[0] == '-')
        {
            cout << "[" << name[0] << "]" << " - Недопустимый символ!" << endl;
            continue;
        }

        if (int('a') <= int(name[0]) && int(name[0]) <= int('я'))
        {
            cout << "[" << name[0] << "]" << " - Недопустимый символ!" << endl;
            continue;
        }
    }
}

```

```

    }

    int check = 0;
    for (int i = 0; i < name.length(); i++)
    {
        if (int('A') <= int(name[i]) && int(name[i]) <= int('Я') || int('a') <=
int(name[i]) && int(name[i]) <= int('я') ||
            int(name[i]) == int(' ') || int(name[i]) == int('.') || int(name[i]) ==
int('ё') || int(name[i]) == int('Ё') ||
            int(name[i]) == int('-'))
        {

        }
        else
        {
            cout << name[i] << " - Недопустимый символ!" << endl;
            incorrect = true;
            break;
        }
    }

    if (incorrect)
    {
        continue;
    }

    else
    {
        break;
    }
}
}

void Address_check(string& ad, string text)
{
    while (true)
    {
        cout << text;
        getline(cin, ad);
        bool incorrect = false;

        if (ad.length() > 50)
        {
            cout << "Ошибка! Повторите ввод" << endl << endl;
            continue;
        }
    }
}

```

```

    if (ad[0] == '.' or ad[0] == '-')
    {
        cout << "[" << ad[0] << "]" << " - Недопустимый символ!" << endl;
        continue;
    }

    int check = 0;
    for (size_t i = 0; i < ad.length(); i++)
    {
        if (int('A') <= int(ad[i]) && int(ad[i]) <= int('Я') || int('a') <= int(ad[i]) &&
int(ad[i]) <= int('я') ||
            int('0') <= int(ad[i]) && int(ad[i]) <= int('9') ||
            int(ad[i]) == int('.') || int(ad[i]) == int(' ') ||
            int(ad[i]) == int('ё') || int(ad[i]) == int('Ё') ||
            int(ad[i]) == int('/') || int(ad[i]) == int('-'))
        {

        }
        else
        {
            cout << ad[i] << " - Недопустимый символ!" << endl;
            incorrect = true;
            break;
        }
    }

    if (incorrect)
    {
        continue;
    }

    else
    {
        break;
    }
}

}

void Place_check(string& pl, string text)
{
    while (true)
    {
        cout << text;
        getline(cin, pl);
        bool incorrect = false;

```

```

if (pl.length() > 50)
{
    cout << "Ошибка! Повторите ввод" << endl << endl;
    continue;
}

if (pl[0] == '.' or pl[0] == '-')
{
    cout << "[" << pl[0] << "]" << " - Недопустимый символ!" << endl;
    continue;
}

int check = 0;
for (size_t i = 0; i < pl.length(); i++)
{
    if (int('A') <= int(pl[i]) && int(pl[i]) <= int('Я') || int('a') <= int(pl[i]) &&
int(pl[i]) <= int('я') ||
        int('0') <= int(pl[i]) && int(pl[i]) <= int('9') ||
        int(pl[i]) == int('.') || int(pl[i]) == int(' ') ||
        int(pl[i]) == int('ё') || int(pl[i]) == int('Ё') ||
        int(pl[i]) == int('/') || int(pl[i]) == int('-'))
    {

    }

    else
    {
        cout << pl[i] << " - Недопустимый символ!" << endl;
        incorrect = true;
        break;
    }
}

if (incorrect)
{
    continue;
}

else
{
    break;
}
}
}

```

```

int Patient::id = 0;

void Filling_Reg_Number(string& RegNumber)
{
    cout << "Заполнение Регистрационного номера: " << endl << endl;

    int district;
    district = Int_check("Введите номер участка (1-9): ");
    string str_district = to_string(district);

    int number;
    number = Patient::id;
    Patient::id++;

    string str_number = to_string(number);

    RegNumber = "0" + str_district + "-";

    for (int i = 0; i < 6 - str_number.length(); i++)
    {
        RegNumber += "0";
    }

    RegNumber += str_number;
}

void Reg_Number_check(string& RegNumber, string text) // Проверка на имя
{
    while (true)
    {
        cout << text;
        getline(cin, RegNumber);
        bool incorrect = false;

        if (RegNumber.length() > 9)
        {
            cout << "Ошибка! Повторите ввод" << endl << endl;
            continue;
        }
        if (RegNumber[0] != '0')
        {
            cout << "Ошибка! Повторите ввод" << endl << endl;
            continue;
        }
        if (RegNumber[2] != '-')
        {

```

```

        cout << "Ошибка! Повторите ввод" << endl << endl;
        continue;
    }

    int check = 0;
    for (size_t i = 0; i < RegNumber.length(); i++)
    {
        if (int('0') <= int(RegNumber[i]) && int(RegNumber[i]) <= int('9') ||
            int(RegNumber[i]) == int('-'))
        {

        }
        else
        {
            cout << RegNumber[i] << " - Недопустимый символ!" << endl;
            incorrect = true;
            break;
        }
    }

    if (incorrect)
    {
        continue;
    }

    else
    {
        break;
    }
}

}

void infoAboutPatient(Patient& new_patient)
{
    string FIO, Place_of_Work, adress, Reg_Number;
    int birth;
    Name_check(FIO, "Введите ФИО больного: ");
    cout << endl;
    birth = Int_check("Введите год рождения больного: ");
    cout << endl;
    Adress_check(adress, "Введите адрес проживания больного: ");
    cout << endl;
    Place_check(Place_of_Work, "Введите место работы(учебы) больного: ");
    cout << endl;

    Filling_Reg_Number(Reg_Number);

```

```

        new_patient.SetInfo(Reg_Number,FIO, adress, Place_of_Work, birth);
    }

int HashFunction(string key)
{
    unsigned int hash = 0;
    unsigned int kod = 0;

    for (int i = 1; i <= size_of_key; i++)
    {
        kod = int(key[i]) * (i * 99);
        hash += 11 * (pow(kod, 3));
    }

    hash %= size_of_spreadsheet;

    return hash;
}

int HashFunction2(string key)
{
    unsigned int hash = 0;

    for (int i = 1; i <= size_of_key; i++)
    {
        hash = int(key[i])*5+i*7;
    }
    hash %= size_of_spreadsheet;

    if (hash % 10 == 5 )
    {
        hash = hash + 2;
    }

    if (hash % 2 == 0)
    {
        hash = hash + 1;
    }

    return hash;
}

bool FillHashSpreadsheet(Patient* array_of_patient, Patient& new_patient, int hashed_key)
{
    bool key_setted = false;

```



```

        if (array_of_patient[hashed_key].RegistrationNumber == "\\0" or
array_of_patient[hashed_key].RegistrationNumber == "delete")
        {
            array_of_patient[hashed_key] = new_patient;
            key_setted = true;

            cout << endl;
            cout << "Регистрационный номер добавлен." << endl << endl;
            return key_setted;
        }

    else
    {
        int hashed_key_2 = HashFunction2(new_patient.RegistrationNumber);
        int adress;

        for (int i = 0; i < 100; i++)
        {
            adress = (hashed_key + hashed_key_2 * i) % 100;
            if (array_of_patient[adress].RegistrationNumber == "\\0" or
array_of_patient[adress].RegistrationNumber == "delete")
            {
                array_of_patient[adress] = new_patient;
                key_setted = true;
                cout << endl;
                cout << "Регистрационный номер добавлен." << endl << endl;
                return key_setted;
            }
        }
    }

    cout << endl;
    cout << "Регистрационный номер НЕ добавлен." << endl << endl;
    return key_setted;
}

void ShowSpreadsheet(Patient* spreadsheet_of_patient)
{
    int number = 1;

    cout << "'Список зарегистрированных пациентов больницы'" << endl << endl;
    cout << "N   |Регистрационный номер|           ФИО пациента|" << endl;
    cout << "----|-----|-----|" << endl;

    for (int i = 0; i < size_of_spreadsheet; i++)

```

```

{
    if (spreadsheet_of_patient[i].Year_of_birth !=0)
    {
        cout << setw(3) << number << "|" << setw(21) <<
spreadsheet_of_patient[i].RegistrationNumber << "|" << setw(25) <<
spreadsheet_of_patient[i].FIO << "|" << endl;
        number++;
    }
}
cout << endl;
}

void Find_Patient_FIO(Patient* spreadsheet_of_patient)
{
    string FIO;
    cout << "'Поиск пациента по ФИО'" << endl << endl;

    Name_check(FIO, "Введите ФИО больного: ");
    cout << endl << endl;

    int amount = 1;
    int found = 0;
    cout << "Список найденных больных: " << endl << endl;

    for (int i = 0; i < size_of_spreadsheet; i++)
    {
        if (spreadsheet_of_patient[i].FIO == FIO)
        {
            found++;
            if (found == 1)
            {
                cout << endl << endl;
                cout << "N |Регистрационный номер|                ФИО пациента|" << endl;
                cout << "---|-----|-----|" << endl;
                found++;
            }

            cout << setw(3) << amount << "|" << setw(21) <<
spreadsheet_of_patient[i].RegistrationNumber << "|" << setw(25) <<
spreadsheet_of_patient[i].FIO << "|" << endl;
            amount++;
        }
    }

    if (found == 0)
    {

```

```

        cout << "Данного пациента в списке нет." << endl << endl;
    }
    cout << endl;
}

int Find_Patient_RegNumber(Patient* spreadsheet_of_patient, string Regnumber) //дописать
{
    int hashed_key = HashFunction(Regnumber);
    int adress = hashed_key;

    if (spreadsheet_of_patient[adress].RegistrationNumber == Regnumber)
    {
        return adress;
    }

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

    else
    {
        int hashed_key_2 = HashFunction2(Regnumber);
        int adress_2;

        for (int i = 0; i < 100; i++)
        {
            adress_2 = (hashed_key + hashed_key_2 * i) % 100;

            if (spreadsheet_of_patient[adress_2].RegistrationNumber == Regnumber)
            {
                return adress_2;
            }

            if (spreadsheet_of_patient[adress_2].RegistrationNumber == "\\0")
            {
                return size_of_spreadsheet;
            }
        }
    }
    return size_of_spreadsheet;
}

bool Can_delete_Patient(Referral_to_Doctor* start, string Reg_Number)
{
    while (start != nullptr)

```

```

{
    if (start->RegistrationNumber_Patient == Reg_Number)
    {
        return false;
    }
    start = start->Next;
}
return true;
}

bool Can_delete_Doctor(Referral_to_Doctor* start, string FIO)
{
    while (start != nullptr)
    {
        if (start->FIO_Doctor == FIO)
        {
            return false;
        }

        if (start->FIO_Doctor[0] != FIO[0])
        {
            start = start->Next_block;
        }

        else
        {
            start = start->Next;
        }
    }
    return true;
}

void Clear_Patient_FIO(Patient* spreadsheet_of_patient, Referral_to_Doctor* start)
{
    string FIO;
    cout << "'Удаление пациента по ФИО'" << endl << endl;

    Name_check(FIO, "Введите ФИО больного: ");
    cout << endl << endl;

    int patient_to_delete = 0;

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

```

```

        patient_to_delete++;
    }
}

if (patient_to_delete == 0)
{
    cout << "Данного пациента в списке нет." << endl << endl;
    return;
}

if (patient_to_delete == 1)
{
    for (int i = 0; i < size_of_spreadsheet; i++)
    {
        if (spreadsheet_of_patient[i].FIO == FIO)
        {
            cout << "Список найденных больных: " << endl << endl;
            cout << "N |Регистрационный номер|          ФИО пациента|" << endl;
            cout << "---|-----|-----|" << endl;
            cout << " 1|" << setw(21) << spreadsheet_of_patient[i].RegistrationNumber <<
            " |" << setw(25) << spreadsheet_of_patient[i].FIO << " |" << endl;

            if (Can_delete_Patient(start, spreadsheet_of_patient[i].RegistrationNumber))
            {
                spreadsheet_of_patient[i].SetZeroes();
                cout << "Данные о пациенте успешно удалены " << endl;
                return;
            }

            cout << endl << "Данному пациенту выдано направление! Удаление невозможно!" <<
endl << endl;

            return;
        }
    }
}

cout << "В базе хранится несколько пациентов с таким ФИО" << endl << endl;
int number_to_delete = 1;

cout << "N |Регистрационный номер|          ФИО пациента|" << endl;
cout << "---|-----|-----|" << endl;

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

```

```

        cout << setw(3) << number_to_delete << "|" << setw(21) <<
spreadsheet_of_patient[i].RegistrationNumber << "|" << setw(25) <<
spreadsheet_of_patient[i].FIO << "|" << endl;
        number_to_delete++;
    }
}
cout << endl;

while (true)
{
    number_to_delete = Int_check("Выберите порядковый номер пациента, которого хотите
удалить:");

    if (number_to_delete > patient_to_delete || number_to_delete<=0)
    {
        cout << "Ошибка ввода! Повторите ввод!" << endl << endl;
        continue;
    }
    break;
}

int compare_to_delete = 1;

for (int i = 0; i < size_of_spreadsheet; i++)
{
    if (spreadsheet_of_patient[i].FIO == FIO)
    {
        if (compare_to_delete == number_to_delete)
        {
            if (Can_delete_Patient(start, spreadsheet_of_patient[i].RegistrationNumber))
            {
                spreadsheet_of_patient[i].SetZeroes();
                cout << "Данные о пациенте успешно удалены " << endl;
                return;
            }

            cout << endl << "Данному пациенту выдано направление! Удаление невозможно!" <<
endl << endl;

            return;
        }
        compare_to_delete++;
    }
}
cout << endl;
}

```

```

void ClearDataBase_Patients(Patient* spreadsheet_of_patient)
{
    for (int i = 0; i < size_of_spreadsheet; i++)
    {
        spreadsheet_of_patient[i].SetZeroes();
    }
}

void Patient_to_File(Patient* spreadsheet_of_patient)
{
    ofstream fout;
    fout.open(path_to_patient);

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

    for (int i = 0; i < size_of_spreadsheet; i++)
    {
        if (spreadsheet_of_patient[i].Year_of_birth != 0)
        {
            fout << spreadsheet_of_patient[i].RegistrationNumber << "\n";
            fout << spreadsheet_of_patient[i].FIO << "\n";
            fout << spreadsheet_of_patient[i].Address << "\n";
            fout << spreadsheet_of_patient[i].Place_of_Work_or_Study << "\n";
            fout << spreadsheet_of_patient[i].Year_of_birth << '\n';
        }
    }
    fout.close();
}

void Patient_from_File(Patient* spreadsheet_of_patient)
{
    ifstream fin;
    fin.open(path_to_patient);

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

    Patient tmp;
    char ex;

```

```

int max_id = -1;
int tmp_id;
int hashed_card;

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

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

    getline(fin, tmp.Address);
    getline(fin, tmp.Place_of_Work_or_Study);
    fin >> tmp.Year_of_birth;
    tmp_id = (int(tmp.RegistrationNumber[3]) - 48) * 100000 +
(int(tmp.RegistrationNumber[4]) - 48) * 10000 + (int(tmp.RegistrationNumber[5]) - 48) * 1000 +
(int(tmp.RegistrationNumber[6]) - 48) * 100 + (int(tmp.RegistrationNumber[7]) - 48) * 10 +
(int(tmp.RegistrationNumber[8]) - 48) * 1;

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

    hashed_card = HashFunction(tmp.RegistrationNumber);
    FillHashSpreadsheet(spreadsheet_of_patient, tmp, hashed_card);
}
fin.close();
Patient::id = max_id + 1;

system("cls");
return;
}

void Position_check(string& position, string text)
{
    while (true)
    {
        cout << text;
        getline(cin, position);
        bool incorrect = false;

```



```

    if (position.length() > 25)
    {
        cout << "Ошибка! Повторите ввод" << endl << endl;
        continue;
    }

    if (position[0] == '.' or position[0] == '-')
    {
        cout << "[" << position[0] << "]" << " - Недопустимый символ!" << endl;
        continue;
    }

    int check = 0;
    for (size_t i = 0; i < position.length(); i++)
    {
        if (int('A') <= int(position[i]) && int(position[i]) <= int('Я') || int('a') <=
int(position[i]) && int(position[i]) <= int('я') ||
            int(position[i]) == int(' ') || int(position[i]) == int('.') ||
int(position[i]) == int('ё') || int(position[i]) == int('Ё') ||
            int(position[i]) == int('-'))
        {

        }
        else
        {
            cout << position[i] << " - Недопустимый символ!" << endl;
            incorrect = true;
            break;
        }
    }

    if (incorrect)
    {
        continue;
    }

    else
    {
        break;
    }
}

}

void Admission_Schedule_check(string& admission_schedule, string text)
{

```

```

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

    if (admission_schedule.length() == 1)
    {
        for (size_t i = 0; i < admission_schedule.length(); i++)
        {
            if (int('8') <= int(admission_schedule[i]) && int(admission_schedule[i]) <=
int('9'))
            {

            }
            else
            {
                cout << admission_schedule[i] << " - Недопустимый символ!" << endl;
                incorrect = true;
                break;
            }
        }
        if (incorrect)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else if (admission_schedule.length() == 2)
    {
        for (size_t i = 0; i < admission_schedule.length(); i++)
        {
            if (int('1') == int(admission_schedule[i]))
            {
                break;
            }

            else
            {
                cout << admission_schedule[i] << " - Недопустимый символ!" << endl;
                incorrect = true;
                break;
            }
        }
    }
}

```

```

    }

    for (size_t i = 1; i < admission_schedule.length(); i++)
    {
        if (int('0') <= int(admission_schedule[i]) && int(admission_schedule[i]) <=
int('7'))
        {

        }
        else
        {
            cout << admission_schedule[i] << " - Недопустимый символ!" << endl;
            incorrect = true;
            break;
        }
    }
    if (incorrect)
    {
        continue;
    }
    else
    {
        break;
    }
}
else
{
    cout << " Ошибка ввода!" << endl;
}

}
}

```

```

void InfoAboutDoctor(Doctor& new_doctor)
{
    string FIO, Position, Admission_Schedule, start_hours, end_hours;
    int Cabinet_Number;

    Name_check(FIO, "Введите ФИО доктора(до 25 символов): ");
    cout << endl;
    Position_check(Position, "Введите должность доктора(до 25 символов): ");
    cout << endl;
    Cabinet_Number = Int_check("Введите номер кабинета доктора(1-24): ");
    cout << endl;

    while (true)

```

```

{
    Admission_Schedule_check(start_hours, "Введите время начала приема (часы): ");
    cout << endl;
    Admission_Schedule_check(end_hours, "Введите время окончания приема (часы): ");

    if (start_hours.length() == 1)
    {
        start_hours = "0" + start_hours;
        break;
    }

    if (end_hours.length() == 1)
    {
        end_hours = "0" + end_hours;
        break;
    }

    if (end_hours.length() == 2)
    {
        break;
    }

    if (start_hours >= end_hours)
    {
        cout << "Ошибка! Введите время приема повторно." << endl << endl;
    }
}
cout << endl;

Admission_Schedule = start_hours + ":00 - " + end_hours + ":00";

new_doctor.SetInfo(FIO, Position, Admission_Schedule, Cabinet_Number);
}

void Find_Doctor_Position(Doctor* p, string Position)
{
    if (p != nullptr)
    {
        Find_Doctor_Position(p->left, Position);

        bool found = true;
        if (Position.length() <= p->Position.length())
        {
            for (int i = 0; i < (p->Position.length() - Position.length() + 1); i++)
            {
                found = true;
            }
        }
    }
}

```

```

        int k = i;

        for (int j = 0; j < Position.length(); j++)
        {

            if (p->Position[k] != Position[j])
            {
                found = false;

                break;
            }
            k++;
        }
        if (found)
        {
            cout << "    |" << setw(25) << p->FIO << "|" << setw(25) << p->Position <<
            "|" << setw(7) << p->Cabinet_Number << "|" << setw(14) << p->Admission_Schedule << "|" <<
endl;

            break;
        }
    }
}
Find_Doctor_Position(p->right, Position);
}
}

```

```

Doctor* Find_Doctor_FIO(Doctor* tmp, string FIO)
{
    while (tmp != nullptr)
    {
        if (tmp->FIO == FIO)
        {
            return tmp;
        }

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

        else
        {
            tmp = tmp->right;
        }
    }
    return nullptr;
}

```

```

}

int height(Doctor* p) // высота
{
    return p ? p->height : 0;
}

int bfactor(Doctor* p) //разность высот левого и правого поддеревьев
{
    return height(p->right) - height(p->left);
}

void fixheight(Doctor* p) // восстановление корректного поля высоты заданного узла
{
    int hl = height(p->left);
    int hr = height(p->right);
    p->height = (hl > hr ? hl : hr) + 1;
}

Doctor* rotate_right(Doctor* p) // правый поворот вокруг p
{
    Doctor* q = p->left;
    p->left = q->right;
    q->right = p;
    fixheight(p);
    fixheight(q);
    return q;
}

Doctor* rotate_left(Doctor* q) // левый поворот вокруг q
{
    Doctor* p = q->right;
    q->right = p->left;
    p->left = q;
    fixheight(q);
    fixheight(p);
    return p;
}

Doctor* balance(Doctor* p) // балансировка
{
    fixheight(p);
    if (bfactor(p) == 2)
    {
        if (bfactor(p->right) < 0)
            p->right = rotate_right(p->right);

```

```

        return rotate_left(p);
    }
    if (bfactor(p) == -2)
    {
        if (bfactor(p->left) > 0)
            p->left = rotate_left(p->left);
        return rotate_right(p);
    }
    return p;
}

Doctor* Insert_Doctor(Doctor* p, Doctor& new_doctor) // вставка ключа k в дерево с корнем p
{
    if (!p)
    {
        Doctor* new_elem_for_insert = new Doctor;
        new_elem_for_insert->SetInfo(new_doctor);
        return new_elem_for_insert;
    }
    if (new_doctor.FIO < p->FIO)
    {
        p->left = Insert_Doctor(p->left, new_doctor);
    }
    else
        p->right = Insert_Doctor(p->right, new_doctor);
    return balance(p);
}

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

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

Doctor* Clear_Doctor_FIO(Doctor* p, string FIO, Referral_to_Doctor* start)// удаление ключа k
из дерева p
{
    if (!p)
    {

```

```

        cout << "Доктор не найден." << endl << endl;
        return nullptr;
    }

    if (FIO < p->FIO)
    {
        p->left = Clear_Doctor_FIO(p->left, FIO, start);
    }

    else if (FIO > p->FIO)
        p->right = Clear_Doctor_FIO(p->right, FIO, start);
    else
    {
        if (Can_delete_Doctor(start, FIO))
        {
            Doctor* q = p->left;
            Doctor* r = p->right;
            delete p;
            cout << "Данные о докторе успешно удалены." << endl << endl;
            if (!r) return q;
            Doctor* min = find_min(r);
            min->right = Clear_Doctor_FIO_min(r);
            min->left = q;
            return balance(min);
        }
        else
        {
            cout << "Удаление невозможно! К доктору имеется направление! " << endl << endl;

        }
    }
    return balance(p);
}

void ClearDataBase_Doctor(Doctor* p)
{
    if (p != nullptr)
    {
        ClearDataBase_Doctor(p->left);
        ClearDataBase_Doctor(p->right);
        delete p;
    }
}

void Show_Doctors_Tree(Doctor* p, int &number)
{

```



```

    if (p != NULL)
    {
        Show_Doctors_Tree(p->left, number);
        number++;
        cout << setw(3) << number << "|" << setw(25) << p->FIO << "|" << setw(25) << p-
>Position << "|" << setw(7) << p->Cabinet_Number << "|" << setw(14) << p->Admission_Schedule
<< "|" << endl;
        Show_Doctors_Tree(p->right, number);
    }
}

int tabs = 0;

void tree_print_beautiful(Doctor* p)
{
    if (p != NULL)
    {
        tabs += 5;
        tree_print_beautiful(p->left);
        for (int i = 0; i < tabs; i++) cout << " ";

        cout << p->FIO << endl;

        tree_print_beautiful(p->right);
        tabs -= 5;
    }
}

void Tree_in_File(Doctor* p, ofstream& fout)
{
    if (p != nullptr)
    {
        Tree_in_File(p->left, fout);
        fout << p->FIO << "\n";
        fout << p->Position << "\n";
        fout << p->Admission_Schedule << "\n";
        fout << p->Cabinet_Number << "\n";
        Tree_in_File(p->right, fout);
    }
}

void Doctors_to_File(Doctor* p)
{
    ofstream fout;
    fout.open(path_to_doctors);

```

```

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

        Tree_in_File(p, fout);
        fout.close();
    }

void Doctors_from_File(Doctor*& p)
{
    ifstream fin;
    fin.open(path_to_doctors);

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

    Doctor new_doctor;

    while (!fin.eof())
    {
        new_doctor.SetZeroes();
        getline(fin, new_doctor.FIO);

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

        getline(fin, new_doctor.Position);
        getline(fin, new_doctor.Admission_Schedule);
        fin >> new_doctor.Cabinet_Number;
        fin.get();
        p = Insert_Doctor(p, new_doctor);
    }

    fin.close();
    system("cls");
    return;
}

string ChekDate()

```

```

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

    while (true)
    {
        cout << "Ввод даты: " << endl;
        day = Int_check("Введите день: ");
        month = Int_check("Введите месяц: ");
        year = Int_check("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 Clear_List(Referral_to_Doctor*& start)
{
    while (start != nullptr)
    {
        Referral_to_Doctor* tmp = start;
        start = start->Next;
        delete tmp;
    }
}

void Layer_List(Referral_to_Doctor*& start, Referral_to_Doctor* writes, int amount_of_writes)
{
    for (int i = 0; i < amount_of_writes; i++)
    {
        if (start == nullptr)
        {
            start = new Referral_to_Doctor(writes[i].RegistrationNumber_Patient,
writes[i].FIO_Doctor, writes[i].Referral_time, writes[i].Referral_date);
        }

        else
        {
            Referral_to_Doctor* tmp = start;

            while (tmp->Next != nullptr)

```

```

        {
            tmp = tmp->Next;
        }
        tmp->Next = new Referral_to_Doctor(writes[i].RegistrationNumber_Patient,
writes[i].FIO_Doctor, writes[i].Referral_time, writes[i].Referral_date);
    }
}

Referral_to_Doctor* tmp = start;
Referral_to_Doctor* tmp_block = start;

while (tmp->Next != nullptr)
{
    if (tmp_block->FIO_Doctor[0] != tmp->FIO_Doctor[0])
    {
        tmp_block->Next_block = tmp;
        tmp_block = tmp;
    }
    tmp = tmp->Next;
}

if (tmp_block->FIO_Doctor[0] != tmp->FIO_Doctor[0])
{
    tmp_block->Next_block = tmp;
    tmp_block = tmp;
}
tmp = tmp->Next;
}

void Swap(Referral_to_Doctor& first, Referral_to_Doctor& second)
{
    Referral_to_Doctor tmp = first;
    first = second;
    second = tmp;
}

void ShakerSort(Referral_to_Doctor*& start)
{
    Referral_to_Doctor* tmp = start;
    int amount_of_writes = 0;

    while (tmp != nullptr)
    {
        amount_of_writes++;
        tmp = tmp->Next;
    }
}

```

```

if (amount_of_writes == 0)
{
    return;
}

Referral_to_Doctor* writes = new Referral_to_Doctor[amount_of_writes];
tmp = start;

for (int i = 0; i < amount_of_writes; i++)
{
    writes[i].RegistrationNumber_Patient = tmp->RegistrationNumber_Patient;
    writes[i].FIO_Doctor = tmp->FIO_Doctor;
    writes[i].Referral_time = tmp->Referral_time;
    writes[i].Referral_date = tmp->Referral_date;

    tmp = tmp->Next;
}
Clear_List(start);

// шейкерная сортировка ))
int leftMark = 1;
int rightMark = amount_of_writes - 1;

while (leftMark <= rightMark)
{
    for (int i = rightMark; i >= leftMark; i--)
    {
        if (writes[i - 1].FIO_Doctor > writes[i].FIO_Doctor)
        {
            swap(writes[i - 1], writes[i]);
        }
    }
    leftMark++;

    for (int i = leftMark; i <= rightMark; i++)
    {
        if (writes[i - 1].FIO_Doctor > writes[i].FIO_Doctor)
        {
            swap(writes[i - 1], writes[i]);
        }
    }
    rightMark--;
}

Layer_List(start, writes, amount_of_writes);

```

```

}

void Issuing_Referrals_to_the_Patient(Referral_to_Doctor*& Next, Patient*
spreadsheet_of_patient, Doctor* p)
{
    string Reg_Number;
    cout << endl << endl << "Выдача направления к доктору" << endl << endl;

    Reg_Number_check(Reg_Number, "Введите регистрационный номер пациента: ");

    int adress = Find_Patient_RegNumber(spreadsheet_of_patient, Reg_Number);

    if (adress == size_of_spreadsheet)
    {
        cout << endl << "Пациент с введенным регистрационным номером отсутствует. Выдача
направления отменена. " << endl;
        return;
    }
    cout << endl;

    string FIO;

    Name_check(FIO, "Введите ФИО доктора(до 25 символов): ");
    Doctor* new_doctor = Find_Doctor_FIO(p, FIO);

    if (new_doctor == nullptr)
    {
        cout << endl << "Доктор с введенным ФИО отсутствует. Выдача направления отменена. " <<
endl;
        return;
    }

    string time;

    while (true)
    {
        Admission_Schedule_check(time, "Введите время приема: ");
        cout << endl;

        if (time.length() == 1)
        {
            time = "0" + time;
            break;
        }
        break;
    }
}

```

```

time = time + ":00";

string date = ChekDate();

string start_time, end_time;
start_time = end_time = "\0";

for (int i = 0; i < 5; i++)
{
    start_time += new_doctor->Admission_Schedule[i];
}

for (int i = 8; i < 13; i++)
{
    end_time += new_doctor->Admission_Schedule[i];
}

if (time < start_time or time > end_time)
{
    cout << "Доктор не принимает в веденное время!" << endl;
    return;
}

if (Next == nullptr)
{
    Next = new Referral_to_Doctor(Reg_Number, FIO, time, date);
    return;
}

else
{
    Referral_to_Doctor* tmp = Next;
    while (tmp->Next != nullptr)
    {
        if (tmp->FIO_Doctor == FIO and tmp->Referral_date == date and tmp->Referral_time
== time)
        {
            cout << "Данное время занято для записи" << endl;
            return;
        }
        tmp = tmp->Next;
    }

    if (tmp->FIO_Doctor == FIO and tmp->Referral_date == date and tmp->Referral_time ==
time)

```



```

    {
        cout << "Данное время занято для записи" << endl;
        return;
    }

    tmp->Next = new Referral_to_Doctor(Reg_Number, FIO, time, date);
}
cout << "Направление выдано" << endl;
}

void Return_Referrals_to_the_Patient(Referral_to_Doctor*& start, Patient*
spreadsheet_of_patient, Doctor* p)
{
    string Reg_Number;
    cout << endl << endl << "Возврат направления к доктору" << endl << endl;

    Reg_Number_check(Reg_Number, "Введите регистрационный номер пациента: ");

    int adress = Find_Patient_RegNumber(spreadsheet_of_patient, Reg_Number);

    if (adress == size_of_spreadsheet)
    {
        cout << endl << "Пациент с введенным регистрационным номером отсутствует. Возврат
направления отменен. " << endl;
        return;
    }
    cout << endl;

    string FIO;

    Name_check(FIO, "Введите ФИО доктора(до 25 символов): ");
    Doctor* new_doctor = Find_Doctor_FIO(p, FIO);

    if (new_doctor == nullptr)
    {
        cout << endl << "Доктор с введенным ФИО отсутствует. Возврат направления отменен. " <<
endl;
        return;
    }

    string date = ChekDate();

    Referral_to_Doctor* tmp = start;

    while (tmp != nullptr)
    {

```

```

        if (tmp->Next_block != nullptr)
        {
            tmp->Next_block = nullptr;
        }
        tmp = tmp->Next;
    }

    if (start == nullptr)
    {
        cout << "Выданные направления в базе данных отсутствуют." << endl << endl;
        return;
    }

    Referral_to_Doctor* first = start;
    Referral_to_Doctor* second = start->Next;

    if (second == nullptr and first->RegistrationNumber_Patient == Reg_Number and first-
>FIO_Doctor == FIO and first->Referral_date == date)
    {
        delete start;
        start = nullptr;
        cout << "Направление успешно удалено." << endl << endl;
        return;
    }

    while (second != nullptr)
    {
        if (second->RegistrationNumber_Patient == Reg_Number and second->FIO_Doctor == FIO and
second->Referral_date == date)
        {
            first->Next = second->Next;
            delete second;
            cout << "Направление успешно удалено." << endl << endl;
            return;
        }

        first = first->Next;
        second = second->Next;
    }
    ShakerSort(start);

    cout << "Направление найти не удалось." << endl << endl;
}

void Show_Referrals(Referral_to_Doctor* start)
{

```

```

int number = 0;
cout << "Список выданных направлений: " << endl << endl;

    cout << " N|Регистрационный номер|                ФИО доктора| Время приема| Дата приема|"
<< endl;
    cout << "----|-----|-----|-----|-----|"
<< endl;
    while (start != nullptr)
    {
        number++;
        cout << setw(3) << number << "|" << setw(21) << start->RegistrationNumber_Patient <<
        "|" << setw(25) << start->FIO_Doctor << "|" << setw(13) << start->Referral_time << "|" <<
        setw(12) << start->Referral_date << "|" << endl;
        start = start->Next;
    }
    cout << endl << endl;
}

void Show_Referrals_Patient(Referral_to_Doctor* start, string Reg_Number)
{
    cout << "Доктора к которым пациент получил направления: " << endl << endl;

    int number = 0;
    cout << " N|                ФИО доктора|" << endl;
    cout << "----|-----|" << endl;

    while (start != nullptr)
    {
        if (start->RegistrationNumber_Patient == Reg_Number)
        {
            number++;
            cout << setw(3) << number << "|" << setw(25) << start->FIO_Doctor << "|" << endl;
        }
        start = start->Next;
    }
}

void Show_Referrals_Doctors(Referral_to_Doctor* start, string FIO, Patient*
spreadsheet_of_patient)
{
    cout << "Пациенты, направленные к данному доктору: " << endl << endl;
    cout << "N |Регистрационный номер|                ФИО пациента|" << endl;
    cout << "----|-----|-----|" << endl;

    int number = 0;

```

```

while (start != nullptr)
{
    if (start->FIO_Doctor == FIO)
    {
        number++;
        int adress = Find_Patient_RegNumber(spreadsheet_of_patient, start-
>RegistrationNumber_Patient);
        cout << setw(3) << number << "|" << setw(21) << start->RegistrationNumber_Patient
<< "|" << setw(25) << spreadsheet_of_patient[adress].FIO << "|" << endl;
    }

    if (start->FIO_Doctor[0] != FIO[0])
    {
        start = start->Next_block;
    }
    else
    {
        start = start->Next;
    }
}

}

void Referral_to_Doctor_from_File(Referral_to_Doctor*& start)
{
    ifstream fin;
    fin.open(path_to_referral);

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

    string reg_number, fio_doctor, referral_time, referral_date;

    while (!fin.eof())
    {
        reg_number = fio_doctor = referral_time = referral_date = "\\0";

        fin >> reg_number;
        fin.get();
        getline(fin, fio_doctor);

        if (fio_doctor == "\\0")
        {
            break;

```

```

    }

    fin >> referral_time;
    fin >> referral_date;

    if (start == nullptr)
    {
        start = new Referral_to_Doctor(reg_number, fio_doctor, referral_time,
referral_date);
    }

    else
    {
        Referral_to_Doctor* tmp = start;

        while (tmp->Next != nullptr)
        {
            tmp = tmp->Next;
        }
        tmp->Next = new Referral_to_Doctor(reg_number, fio_doctor, referral_time,
referral_date);
    }
}
fin.close();
return;
}

```

```

void Referral_to_Doctor_to_File(Referral_to_Doctor* start)
{
    ofstream fout;
    fout.open(path_to_referral);

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

    while (start != nullptr)
    {
        fout << start->RegistrationNumber_Patient << "\n";
        fout << start->FIO_Doctor << "\n";
        fout << start->Referral_time << "\n";
        fout << start->Referral_date << "\n";
        start = start->Next;
    }
}

```

```

        fout.close();
    }

int main()
{
    system("color 0F");

    setlocale(LC_ALL, "Russian");
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    int menu = 100;

    Patient*Database_Patients = new Patient[size_of_spreadsheet];
    Patient_from_File(Database_Patients);

    Doctor* Database_Doctors = NULL;
    Doctors_from_File(Database_Doctors);

    Referral_to_Doctor* Next = nullptr;
    Referral_to_Doctor_from_File(Next);

    while (menu != 0)
    {
        Menu();
        menu = Int_check("Выберете пункт меню: ");

        switch (menu)
        {
        case 1:
        {
            system("cls");

            int amount_of_patient;
            cout << endl;

            amount_of_patient = Int_check("Введите количество новых больных, которых хотите занести в базу данных поликлиники: ");
            cout << endl;

            for (int i = 0; i < amount_of_patient; i++)
            {
                Patient new_patient;
                infoAboutPatient(new_patient);
                int hashed_registrat = HashFunction(new_patient.RegistrationNumber);
                FillHashSpreadsheet(Database_Patients, new_patient, hashed_registrat);
            }
        }
        }
    }
}

```

```

    }

    if (amount_of_patient != 0)
    {
        cout << endl;
        cout << "Данные о новых больных добавлены" << endl << endl;
    }
    system("pause");
    break;
}
case 2:
{
    system("cls");

    ShowSpreadsheet(Database_Patients);
    system("pause");
    break;
}
case 3:
{
    system("cls");
    ShowSpreadsheet(Database_Patients);
    cout << endl;

    Find_Patient_FIO(Database_Patients);
    system("pause");
    break;
}
case 4:
{
    system("cls");
    ShowSpreadsheet(Database_Patients);
    cout << endl;

    string Reg_Number;
    Reg_Number_check(Reg_Number, "Введите регистрационный номер пациента: ");

    int adress = Find_Patient_RegNumber(Database_Patients, Reg_Number);

    if (adress == size_of_spreadsheet)
    {
        cout << endl << "Пациент с веденным регистрационным номером отсутствует. Поиск
по регистрационному номеру отменен." << endl;
        system("pause");
        break;
    }
}

```

```

        cout << "N |Регистрационный номер|                ФИО пациента|
Адрес|                Место работы(учебы)| Год рождения|" << endl;
        cout << "----|-----|-----|-----|
-----|-----|-----|-----|
|" << endl;
        cout << " 1|" << setw(21) << Database_Patients[address].RegistrationNumber << "|"
<< setw(25) << Database_Patients[address].FIO << "|" << setw(50) <<
Database_Patients[address].Address << "|" << setw(50) <<
Database_Patients[address].Place_of_Work_or_Study<< "|" << setw(13) <<
Database_Patients[address].Year_of_birth << "|" << endl;
        cout << endl;

        ShakerSort(Next);
        Show_Referrals_Patient(Next, Reg_Number);
        system("pause");
        break;
    }
    case 5:
    {
        system("cls");

        SwowSpreadsheet(Database_Patients);
        cout << endl << endl;

        ShakerSort(Next);
        Clear_Patient_FIO(Database_Patients, Next);
        system("pause");
        break;
    }
    case 6:
    {
        ShakerSort(Next);

        if (Next == nullptr)
        {
            ClearDataBase_Patients(Database_Patients);
            cout << "База данных пациентов полностью очищена" << endl << endl;
            system("pause");
        }

        else
        {
            cout << "Базу данных очистить нельзя! Имеются ввданные направления!" << endl
<< endl;
        }
    }
}

```



```

        system("pause");
        break;
    }
case 7:
{
    system("cls");

    int amount_of_patient;
    cout << endl;

    amount_of_patient = Int_check("Введите количество новых врачей, которых хотите
занести в базу данных поликлиники: ");
    cout << endl;

    for (int i = 0; i < amount_of_patient; i++)
    {
        Doctor new_doctor;
        InfoAboutDoctor(new_doctor);
        Database_Doctors = Insert_Doctor(Database_Doctors, new_doctor);
    }

    if (amount_of_patient != 0)
    {
        cout << endl;
        cout << "Данные о новых врачах добавлены" << endl << endl;
    }

    system("pause");
    break;
}
case 8:
{
    system("cls");

    int number = 0;
    cout << "'Список зарегистрированных докторов больницы'" << endl << endl;
    cout << "      |                ФИО доктора|                Должность|  Н каб. |  График
приема|" << endl;
    cout << "----|-----|-----|-----|-----
-----|" << endl;
    Show_Doctors_Tree(Database_Doctors, number);
    cout << endl << endl;

    tree_print_beautiful(Database_Doctors);
    cout << endl;

```

```

        system("pause");
        break;
    }
    case 9:
    {
        system("cls");

        int number = 0;

        cout << "'Список зарегистрированных докторов больницы'" << endl << endl;
        cout << "   N|                ФИО доктора|                Должность| N каб.| График
приема|" << endl;
        cout << "----|-----|-----|-----|-----
-----|" << endl;
        Show_Doctors_Tree(Database_Doctors, number);
        cout << endl << endl;

        string FIO;

        Name_check(FIO, "Введите ФИО доктора(до 25 символов): ");
        Doctor* new_doctor = Find_Doctor_FIO(Database_Doctors, FIO);

        if (new_doctor == nullptr)
        {
            cout << endl << "Доктор с введенным ФИО отсутствует. Поиск по ФИО отменен." <<
endl;

            system("pause");
            break;
        }

        cout << endl << "Найденный доктор: " << endl;
        cout << "   N|                ФИО доктора|                Должность| N каб.| График
приема|" << endl;
        cout << "----|-----|-----|-----|-----
-----|" << endl;
        cout << "   1|" << setw(25) << new_doctor->FIO << "|" << setw(25) << new_doctor-
>Position << "|" << setw(7) << new_doctor->Cabinet_Number << "|" << setw(14) << new_doctor-
>Admission_Schedule << "|" << endl;
        cout << endl << endl;

        ShakerSort(Next);

        Show_Referrals_Doctors(Next, FIO, Database_Patients);
        cout << endl << endl;
        system("pause");
        break;
    }
}

```

```

    }
    case 10:
    {
        system("cls");

        int number = 0;
        cout << "'Список зарегистрированных докторов больницы'" << endl << endl;
        cout << "      |                ФИО доктора|                Должность| N каб.| График
приема|" << endl;
        cout << "----|-----|-----|-----|-----
-----|" << endl;
        Show_Doctors_Tree(Database_Doctors, number);

        cout << endl << endl << "'Поиск доктора по фрагменту Должность'" << endl << endl;

        string Position;
        Position_check(Position, "Введите должность доктора(до 25 символов): ");
        cout << endl;

        cout << "Найденный доктор: " << endl << endl;
        cout << "      |                ФИО доктора|                Должность| N каб.| График
приема|" << endl;
        cout << "----|-----|-----|-----|-----
-----|" << endl;
        Find_Doctor_Position(Database_Doctors, Position);

        cout << endl;
        system("pause");
        break;
    }
    case 11:
    {
        system("cls");

        int number = 0;
        cout << "'Список зарегистрированных докторов больницы'" << endl << endl;
        cout << "      |                ФИО доктора|                Должность| N каб.| График
приема|" << endl;
        cout << "----|-----|-----|-----|-----
-----|" << endl;

        Show_Doctors_Tree(Database_Doctors, number);

        cout << endl << endl << "'Удаление доктора по ФИО'" << endl << endl;

        string FIO;

```

```

        Name_check(FIO, "Введите ФИО доктора(до 25 символов): ");
        cout << endl;

        ShakerSort(Next);
        Database_Doctors = Clear_Doctor_FIO(Database_Doctors, FIO, Next);

        system("pause");
        break;
    }
    case 12:
    {
        ShakerSort(Next);

        if (Next == nullptr)
        {
            ClearDataBase_Doctor(Database_Doctors);

            Database_Doctors = nullptr;

            cout << "База данных врачей полностью очищена" << endl << endl;
            system("pause");
        }

        else
        {
            cout << "Базу данных очистить нельзя! Имеются выданные направления!" << endl
<< endl;
        }
        system("pause");
        break;
    }
    case 13:
    {
        system("cls");
        ShowSpreadsheet(Database_Patients);
        cout << endl;

        int number = 0;
        cout << "'Список зарегистрированных докторов больницы'" << endl << endl;
        cout << "      |                ФИО доктора|                Должность|  N каб.| График
приема|" << endl;
        cout << "----|-----|-----|-----|-----|
-----|" << endl;
        Show_Doctors_Tree(Database_Doctors, number);

        Issuing_Referrals_to_the_Patient(Next, Database_Patients, Database_Doctors);

```

```

        system("pause");
        break;
    }
    case 14:
    {
        system("cls");

        Show_Referrals(Next);
        system("pause");
        break;
    }
    case 15:
    {
        system("cls");

        ShakerSort(Next);
        Show_Referrals(Next);
        Return_Referrals_to_the_Patient(Next, Database_Patients, Database_Doctors);
        system("pause");
        break;
    }
    default:
        break;
    }
}

Patient_to_File(Database_Patients);
delete[]Database_Patients;

Doctors_to_File(Database_Doctors);
ClearDataBase_Doctor(Database_Doctors);

Referral_to_Doctor_to_File(Next);

system("pause");
return 0;
}

```