Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Инженерно-экономический факультет
Кафедра экономической информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
по дисциплине
“Основы конструирования программ”
на тему

“УЧЁТ КЛИЕНТОВ ПЛАТНОЙ КЛИНИКИ”

Студент группы 972302 Пятницкий С. В.

Руководитель Голда О. А.

Минск-2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Поликлиника – место, куда мы приходим если у нас появились жалобы к своему здоровью, чтобы нам помогли. Ведь если вовремя не обращаться к врачу, то это может привести к необратимым последствиям. Правильное определение диагноза возможно лишь в том случае, когда врач изучит вашу историю болезни и обращений, чтобы определить уязвимые места в организме и проанализировать изменение ваших показателей. Для удобства работы работников поликлиники и повышения их эффективности, этот процесс требует систематизации, а, следовательно, в соответствующем программном обеспечении.

В связи с большим количеством информации об работниках поликлиники, её клиентах и обращениях, необходимостью хранить одновременно огромное количество данных, значительно упрощающих работу. Популярность данных систем заключается в удобстве и практичности.

В поликлинике хранится информация об истории болезней более тысячи пациентов: перенесённые болезни, противопоказания, рекомендации по правильному образу жизни и т.д. Врачу, определяя диагноз, необходимо изучить огромное количество информации. Упрощение этого процесса невозможно без использования специальных приложений. Они весомо облегчают работу, ведь сотрудникам не нужно просматривать множество данных и самостоятельно выбирать те, которые подходят под описание, нужно всего лишь отсортировать список по требуемому критерию.

Поликлиники нуждаются в особом контроле, т.к. от них зависят жизни людей. Для этого администрация поликлиник тщательно следят за заполнением информации. В настоящее время вся информация о работниках, пациенте, его история болезни, противопоказаниях и лечении может храниться в компьютере. Подобные приложения позволяют легко и быстро вносить изменения, вести учет и многое другое.

Администрация поликлиники должна иметь возможность просматривать, редактировать, сравнивать записи об врачах, чтобы увеличивать эффективность работников, а в результате и поликлиники, контролировать добросовестное выполнение своих обязанностей работниками. Также это требуется для того, чтобы контролировать квалифицированность врачей, т.к. от их опыта зависят жизни тысяч людей.

Основная цель работы заключается в снижении количества ошибок при обработке информации об клиентах платной клиники.

Поставленная цель потребовала решение следующих задач:

– ознакомиться с (предметной областью);
– ознакомиться со структурой хранения данных;
– разработать пользовательские функции приложения;
– разработать функциональную тему задачи;
– описать работу программы (разработать руководство пользователя).

Объектом исследования курсового проекта является процесс учета клиентов платной клиники.

Ключевые слова: ФУНКЦИЯ, СТРУКТУРА, АЛГОРИТМ.

1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ОБ СПОЛЬЗУЕМЫХ АЛГОРИТМАХ

1.1Сортировка методом Хоара

Быстрая сортировка, сортировка Хоара (с английского «quicksort») – наиболее известный и эффективный алгоритм сортировки. Алгоритм, по принципу функционирования, входит в класс обменных сортировок (сортировка перемешиванием, пузырьковая сортировка и др.), выделяясь при этом высокой скоростью работы. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. (Таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов.)

Общая идея алгоритма состоит в следующем:

- выбрать из массива элемент, называемый опорным;
- разбиение массива на несколько меньших подмассивов;
- сортировка подмассивов.

Рисунок 1.1 - Пример быстрой сортировки

Первым этапом является выбор опорного элемента. Опорным может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов.

Вторым этапом сортировки происходит сравнивание всех остальных элементов с опорным и разбиение массива на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие». На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

В третьем этапе сортировки для отрезков «меньших» и «больших» значений выполняется рекурсивно та же последовательность операций, если длина отрезка больше единицы. После завершение сортировки отрезков происходит их объединение в отсортированный массив (рисунок 1.1).

Быструю сортировку обязательно стоит рассматривать первой при выборе метода внутренней сортировки достаточно большого объёма данных. Алгоритм этой сортировки содержит сложную фазу разбиения и простую фазу слияния. В худшем случае выполненная работа эквивалентна работе при сортировке выбором.

1.2Линейный поиск

Линейный, последовательный поиск — самый простой алгоритм поиска в программировании. Данный алгоритм является простейшим алгоритмом поиска и, в отличие, например, от двоичного поиска, не накладывает никаких ограничений и имеет простейшую реализацию (рисунок 1.2). Поиск значения функции осуществляется простым сравнением очередного рассматриваемого значения (как правило, поиск происходит слева направо, то есть от меньших значений аргумента к большим) и, если значения совпадают, то поиск считается завершённым. В связи с малой эффективностью по сравнению с другими алгоритмами линейный поиск обычно используют, только если отрезок поиска содержит очень мало элементов, тем не менее, линейный поиск не требует дополнительной памяти или обработки/анализа функции, так что

может работать в потоковом режиме при непосредственном получении данных из любого источника.

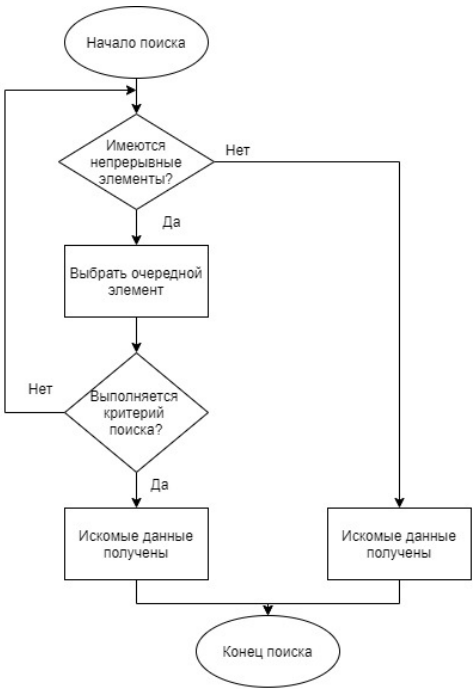


Рисунок 1.2 - Принцип работы линейного поиска

1.3XOR-шифрование

Самым простым и одним из самых эффективных (при правильном использовании) алгоритмов шифрования является так называемое XOR-шифрование (рисунок 1.3). Простейший шифр на основе бинарной логики, который обладает высокой криптографической стойкостью. Без знания ключа, расшифровать его невозможно. Идея алгоритма заключается в том, что к каждому символу исходного применяется побитовая логическая операция XOR (в языке C операция XOR обозначается специальным знаком «^»). Операция XOR обладает симметричностью. Это значит, что если зашифровать один и тот же символ 2 раза с одним и тем же ключом, то на выходе получим сам этот файл без изменений. Из этого факта становится ясно, что для шифрования и расшифровывания будет использоваться одна и та же функция, что существенно упрощает реализацию алгоритма. Если же ключи при шифровке и дешифровке различаются, то на выходе будет получен некорректный символ.

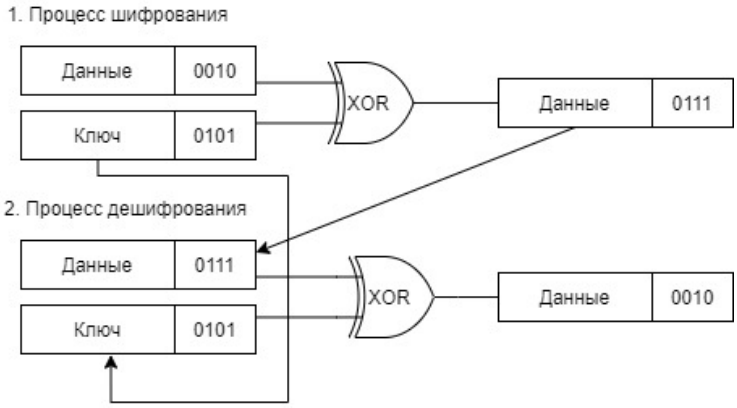


Рисунок 1.3 - Принцип работы XOR-шифрования

2. ОПИСАНИЕ ОРГАНИЗАЦИИ СТРУКТУР ХРАНИМЫХ ДАННЫХ

Структура – это сгруппированные под одним именем одна или несколько переменных (возможно, различных типов), объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации. Структуры упрощают написание и понимание принципов работы программ, а также помогают сгруппировать данные, объединяемые каким-либо общим понятием.

Программа, созданная в этом проекте, имеет 5 основные структуры:

Doctors;

Customers;

Appeals;

Date.

Структура Customers и Appeals имеют в себе вложенную структуру Data.

Структура Accounts.

```
typedef struct Accounts{
        char* login;
        char* password;
        int IDdoc;
        int mode;
        struct Accounts* prev;
}Accounts;
```

Структура Accounts содержит поле login типа char*, которое хранит логин, поле password типа char* хранит пароль, поле IDdoc типа int хранит ID, поле mode типа int хранит категорию.

Структура Doctors.

```
typedef struct Doctors{
   int ID;
   char* surname;
   char* name;
   char* middleName;
   char* specialty;
   char* category;
   struct Doctors* prev;
};
```

Структура Doctors содержит поле ID типа int, которое хранит личный номер врача, поле surname типа char* хранит фамилию, поле name типа char*

хранит имя, поле middleName типа char* хранит отчество, поле specialty типа char* хранит специальность, поле category типа char* хранит категорию доктора. Эта структура предназначена для хранения ФИО, специальности и категории врачей.

Структура Customers.

```
typedef struct Customers{
   int ID;
   char* surname;
   char* name;
   char* middleName;
   struct Date dateBirthday;
   struct Customers* prev;
};
```

Структура Customers содержит поле ID типа int, которое хранит личный номер пациента, поле surname типа char* хранит фамилию, поле name типа char* хранит имя, поле middleName типа char* хранит отчество, поле dateBirthday типа Date хранит дату рождения пациента. Эта структура предназначена для хранения ФИО и даты рождения пациента.

Структура Appeals.

```
typedef struct Appeals{
   int IDdoc;
   int IDcust;
   Date dateAppeal;
   char* diagnosis;
   int costOfTreatment;
   struct Appeals* prev;
};
```

Структура Appeals содержит поле IDdoc типа int, которое хранит ID доктора, поле IDcust типа int хранит ID пациента, поле DateAppeals типа Date хранит дату обращения, поле diagnosis типа char* хранит диагноз пациента, поле costOfTreatment типа int хранит стоимость лечения. Эта структура предназначена для хранения личных номеров докторов и пациентов, дату приёма, диагноз и стоимости лечения пациента.

Структура Date.

```
typedef struct Date{
   int day;
   int month;
   int year;
};
```

Структура Date содержит поле day типа int, которое хранит день, поле month типа int хранит месяц, поле year типа int хранит год. Эта структура предназначена для хранения даты.

3. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ ПРИЛОЖЕНИЯ

Прототипы функций, использующихся в данном приложении:

```
char* encryption(char*); //шифрование данных
bool checkStr(char**, int);  //проверка на ввод
char* writeStr(int); //ввод данных
char* readFromFile(FILE*);  // считывание одной строки из файла
char* strconv(char*); //привидение строки в нужную форму
void readingDataDoc(Doctors** , FILE*);  //считывание информации из файла
void readingDataCust(Customers** , FILE*);  //считывание информации в структуры
void readingDataApp(Appeals**  FILE*);  //считывание информации в структуры
void readingDataAccount(Accounts**, FILE*);  //считывание информации в структуры
void fprintfDoc(Doctors** ,  FILE*);  //вывод в файл данных врача
FILE* rewriteDoc(Doctors** , FILE*);  //перезапись структуры в файл
void fprintfCust(Customers** , FILE*);  //вывод в файл данных пациента
FILE* rewriteCust(Customers** , FILE*);  //перезапись структуры в файл
void fprintfApp(Appeals** , FILE*);  //вывод в файл данных о приёме
FILE* rewriteApp(Appeals** , FILE*);  //перезапись структуры в файл
bool compare(char** , FILE*);  //поиск введённых данных в файле
bool ReadAuthCheck(FILE* , char* , char* , int);  //проверка логина и пароля
int AuthFunction(int); //ввод логина и пароля
void registration(); //регистрация нового пользователя
void EditPass(Accounts**, int);  //изменение пароля
void dataTableDoc(Doctors**); //все данные врачей в табличной форме
void dataTableCust(Customers**); //все данные пациентов в табличной форме
void dataTableApp(Appeals**); //все данные о приёмах в табличной форме
void pushDoc(Doctors** , FILE*); //добавление нового врача
void pushCust(Customers** , FILE*); //добавление нового пациента
void PushAppealCust(Appeals** , Customers** , FILE* , FILE* , int); //новый приём
void popDoc(Doctors**); //удаление врача
void popCust(Customers**); //удаление пациента
void editDataDoc(Doctors**); //изменение данных врача
void editDataCust(Customers**); //изменение данных пациента
void editDataApp(Appeals**); //изменение данных о приёме
void DoctorsHat();  //шапка для вывода данных врача
void CustomersHat();  //шапка для вывода данных пациента
void AppealsHat();  //шапка для вывода данных приёма
void SearchIDDoc(Doctors*);  //поиск врача по ID
void SearchSurnameDoc(Doctors*);  //поиск врача по фамилии
void SearchNameDoc(Doctors*);  //поиск врача по имени
void SearchMiddleNameDoc(Doctors*);  //поиск врача по отчеству
void SearchSpecialtyDoc(Doctors*);  //поиск врача по специальности
void SearchCategoryDoc(Doctors*);  //поиск врача по категории
void SearchMenuDoc(Doctors*);  //меню поиска врачей
void SortIDUpDoc(Doctors**);  //сортировка врачей по ID по возрастанию
```

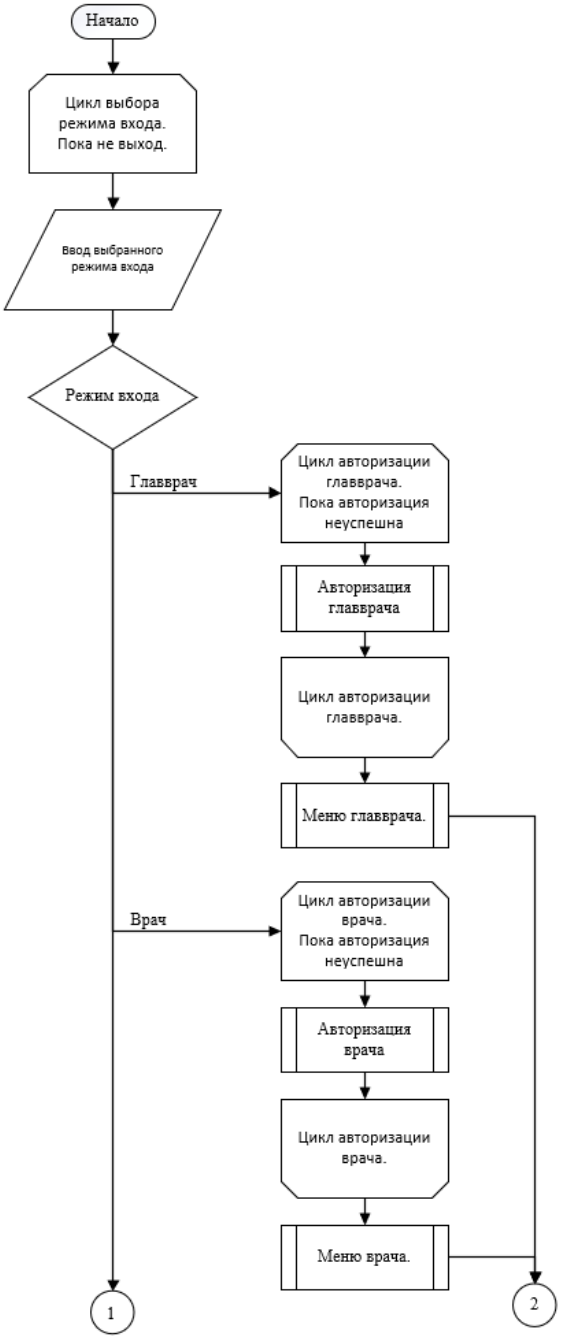
```
void SortIDDownDoc(Doctors**);  //сортировка врачей по ID по убыванию
void SortSurnameDoc(Doctors**);  //сортировка врачей по фамилии по алфавиту
void SortNameDoc(Doctors**);  //сортировка врачей по имени по алфавиту
void SortMiddleNameDoc(Doctors**);  //сортировка врачей по отчеству по алфавиту
void SortSpecialtyDoc(Doctors**);  //сортировка врачей по специальностям
void SortCategoryDoc(Doctors**);  //сортировка врачей по категории
void SortMenuDoc(Doctors*);  //меню сортировки врачей
void Task(Appeals**); //выполнение задачи
void SearchIDCust(Customers*);  //поиск пациента по ID
void SearchSurnameCust(Customers*);  //поиск пациента по фамилии
void SearchNameCust(Customers*);  //поиск пациента по имени
void SearchMiddleNameCust(Customers*);  //поиск пациента по отчеству
void SearchDateCust(Customers*);  //поиск пациента по дате рождения
void SearchMenuCust(Customers*);  //меню поиска пациентов
void SortIDUpCust(Customers**);  //сортировка ID пациентов по возрастанию
void SortIDDownCust(Customers**);  //сортировка ID пациентов по убыванию
void SortSurnameCust(Customers**);  //сортировка фамилий пациентов по алфавиту
void SortNameCust(Customers**);  //сортировка имён пациентов по алфавиту
void SortMiddleNameCust(Customers**);  //сортировка отчеств пациентов
void SortDateDownCust(Customers**);  //сортировка даты по убыванию
void SortDateUpCust(Customers**);  //сортировка даты по возрастанию
void SortMenuCust(Customers*);  //меню сортировки пациентов
void SearchIDAppDoc(Appeals*);  //поиск по ID доктора
void SearchIDAppCust(Appeals*);  //поиск по ID пациента
void SearchDiagnosisApp(Appeals*);  //поиск диагноза
void SearchMenuApp(Appeals*);  //меню поиска приёмов
void SortIDUpApp(Appeals**);  //сортировка ID приёма по возрастанию
void SortIDDownApp(Appeals**);  //сортировка ID приёма по убыванию
void SortDateDownApp(Appeals**);  //сортировка даты приёма от новой к старой
void SortDateUpApp(Appeals**);  //сортировка даты приёма от старой к новой
void SortCoastUpApp(Appeals** head);  //сортировка стоимости по возрастанию
void SortCoastDownApp(Appeals**);  //сортировка стоимости по убыванию
void SortMenuApp(Appeals**);  //меню сортировки приёмов
char manage_menu();  //меню для выбора поиск/сортировка
void ManageDataDoc(Doctors*);  ////выбор действий с данными врачей
void ManageDataCust(Customers*);  //выбор действий с данными пациентов
void ManageDataApp(Appeals*);  //выбор действий с данными приёмов
void UserSwitchDataMenu(Customers* , Appeals*);  //меню поиска и сортировки
char data_menu();  // меню для выбора просматриваемых данных
char main_menu();  //главное меню
void user_menu(int);  //меню врача
void admin_menu(int);  //меню главврача
void regist_menu(int);  //меню медрегистратора
```

Функция int main() – функция программы, содержащая главное меню. С помощью оператора switch() осуществляется выбор нужного действия, а затем вызов соответствующей функции.

4. ФУНКЦИОНАЛЬНАЯ СХЕМА ЗАДАЧИ, СХЕМЫ АЛГОРИТМОВ РАБОТЫ ПРИЛОЖЕНИЯ

4.1Функция main()

Функция main() является главной функцией программы. Данная функция выполняет вызов функции ввода и проверку данных аккаунтов пользователей в соответствии с категорией пользователя. Рассмотрим блок-схему данной функции. Функция организации главного меню представлена на рисунке 4.1



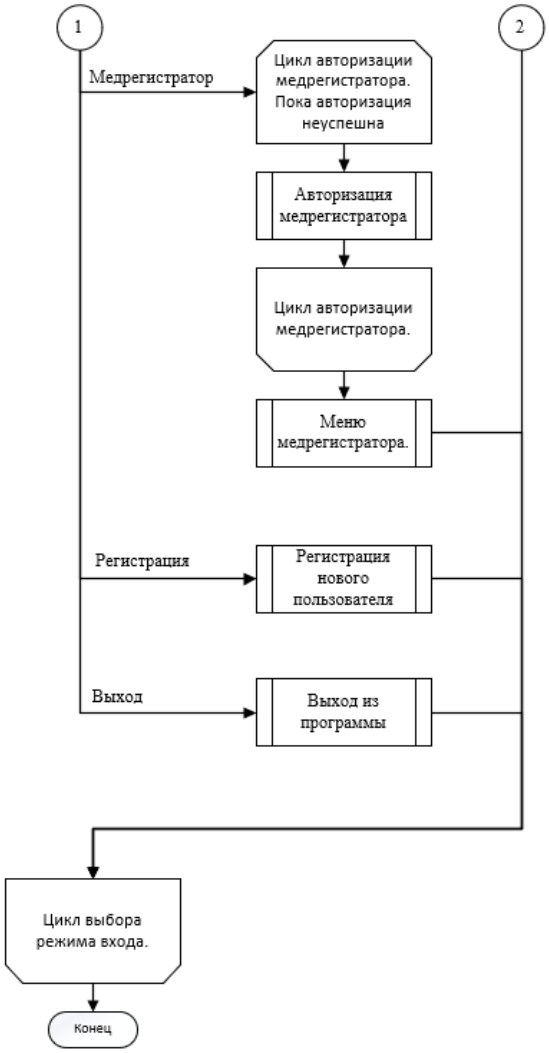


Рисунок 4.1 – Схема организации главного меню

Функция main() представлена следующим образом:

```
int main() {
   int ID;
   system("chcp 1251>null");
   while (1) {
      switch (main_menu()) {
      case '1':
         if((ID = AuthFunction(1)) != 0) admin_menu(ID);
         break;
      case '2':
         if ((ID = AuthFunction(2)) != 0) user_menu(ID);
         break;
      case '3':
         if ((ID = AuthFunction(3)) != 0) regist_menu();
         break;
      case '4':
         registration();
```

```
            break;
        case '5':
            return 0;
        }
    }
}
```

4.2Функция AuthFunction(int)

Функция AuthFunction(int) позволяет пользователю войти в систему, пои правильном вводе логина и пароля. Эта функция принимает ввод логина, пароля и проверяет введённые данные с данными, которые находятся в файле с логинами и паролями всех пользователей. Если же в системе нет введённых администраторов, то производится ввод такового. Это реализовано для начальной настройки и использования системы. Для безопасности данные шифруются и сравнение производится уже в зашифрованном виде. Рассмотрим блок-схему данной функции (рисунок 4.2).

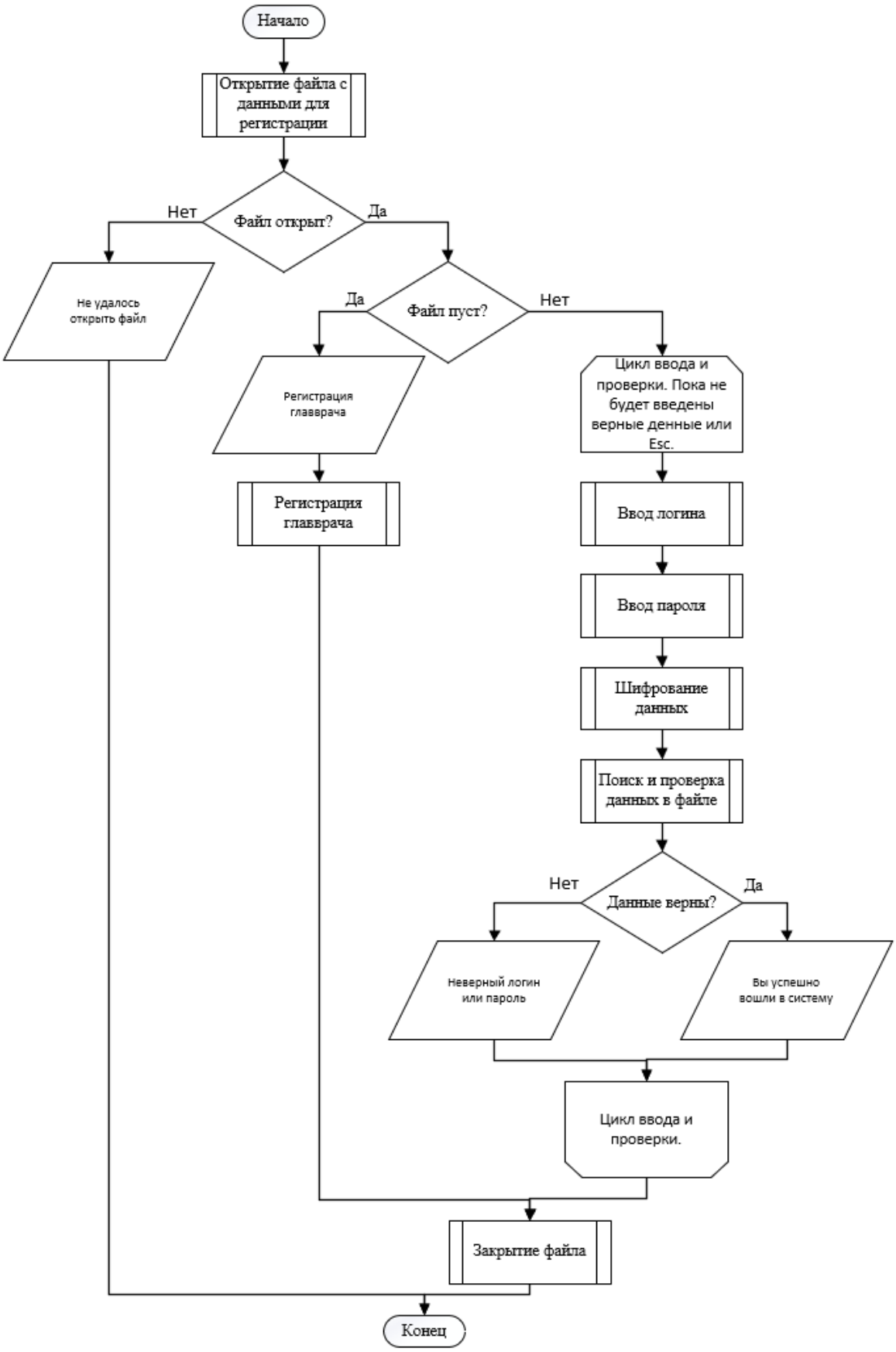


Рисунок 4.2 – Схема организации проверки логина и пароля

Функция AuthFunction(int) представлена следующим образом:

```
int AuthFunction(int mode) {
   FILE* authFile,* DoctorsFile = NULL;
   bool Auth = false, EscCheck = false, check;
   char* inputLogin = NULL, * inputPassword = NULL;
   int ID = 0;
   authFile = fopen("Authentification.txt", "a+");
   if (authFile == NULL) {
      printf("\tОшибка открытия файла!\n");
   }
   else {
      fseek(authFile, 0, SEEK_END);
      if (ftell(authFile) == 0 && mode == 1) {
         DoctorsFile = fopen("ListOfDoctors.txt", "a+");
         if (DoctorsFile == NULL) {
            printf("\tОшибка открытия файла!\n");
         }
         else {
            printf("\tРегистрация главврача\n");
            system("pause");
            system("cls");
            pushDoc(&headDoc, DoctorsFile);
            if (DoctorsFile != NULL) {
               rewriteDoc(&headDoc, DoctorsFile);
               fclose(DoctorsFile);
               memset(headDoc, 0, sizeof(Doctors));
               free(headDoc);
            }
         }
      }
      do {
         check = false;
         EscCheck = false;
         do {
            printf("Введите логин: ");
            inputLogin = writeStr(0);
            if (strcmp(inputLogin, "EscMode") == 0) EscCheck = true;
            else {
               if (strlen(inputLogin) < MinLogSymbol || strlen(inputLogin) > MaxLogSymbol) {
                  printf("\tВозможный размер логина от %d до %d символов!\n", MinLogSymbol, MaxLogSymbol);
                  system("pause");
                  system("cls");
               }
```

```
                check = checkStr(&inputLogin, 4);
            }
        } while (strlen(inputLogin) < MinLogSymbol || strlen(inputLogin) > MaxLogSymbol && EscCheck == false && check == false);
        check = false;
        if (EscCheck == false) {
            do {
                printf("Введите пароль: ");
                inputPassword = writeStr(1);
                if (strcmp(inputPassword, "EscMode") == 0) EscCheck = true;
                else {
                    if (strlen(inputPassword) < MinPassSymbol || strlen(inputPassword) > MaxPassSymbol) {
                        printf("\tВозможный размер пароля от %d до %d символов!\n", MinPassSymbol, MaxPassSymbol);
                        system("pause");
                        system("cls");
                    }
                    check = checkStr(&inputPassword, 4);
                }
            } while (EscCheck != true && strlen(inputPassword) < MinPassSymbol || strlen(inputPassword) > MaxPassSymbol && check == false);
        }
        if (EscCheck != true) {
            inputLogin = encryption(inputLogin);
            inputPassword = encryption(inputPassword);
            Auth = ReadAuthCheck(authFile, inputLogin, inputPassword, mode);
            if (Auth == true) {
                fseek(authFile, -3, SEEK_CUR);
                ID = atoi(readFromFile(authFile));;
                printf("\tВы успешно вошли в систему\n\n");
            }
            else {
                printf("\tНеверный логин или пароль\n\n");
            }
            system("pause");
            system("cls");
        }
    } while (Auth == false && EscCheck == false);
    if (EscCheck == false) {
        free(inputLogin);
        free(inputPassword);
    }
    if (authFile != NULL) fclose(authFile);
    else printf("\tОшибка закрытия файла!\n");
```

```
        }
        return ID;
    }
```

4.3Функция registration(int)

Функция registration(int) отвечает за добавления учетной записи нового пользователя. Данную запись имеет право создать любой пользователь, но для этого администратор должен добавить ФИО и логин пользователя в систему, т.к. образом реализована зависимость регистрации от администратора, чтобы нельзя было зарегестрироваться и пользоваться системой любым пользователям. Для безопасности данные шифруются и сравнение производится уже в зашифрованном виде. Блок-схема данной функции предоставлена на рисунке 4.3.

Пример реализации данной функции в программе:

```
void registration() {
    FILE* authFile, * regFile;
    bool EscCheck = false, CheckLog = false, CheckPass = false, check = false;
    char* inputLogin = NULL, * inputPassword = NULL;
    EscCheck = false;
    authFile = fopen("Authentification.txt", "a+");
    if (authFile == NULL) {
        printf("\tОшибка открытия файла!\n");
    }
    else {
        check = false;
        do {
            printf("Введите логин: ");
            inputLogin = writeStr(0);
            if (strcmp(inputLogin, "EscMode") == 0) EscCheck = true;
            else {
                check = checkStr(&inputLogin, 4);
                inputLogin = encryption(inputLogin);
                if (strlen(inputLogin) < MinLogSymbol || strlen(inputLogin) > MaxLogSymbol)
{
                    printf("Возможный размер логина от %d до %d символов!\n",
MinLogSymbol, MaxLogSymbol);
                    system("pause");
                    system("cls");
                }
                else if (compare(&inputLogin, authFile) == true) {
                    printf("Данный логин недоступен!\n");
```

```
                system("pause");
                system("cls");
            }
            else CheckLog = true;
        }
    } while (CheckLog == false && EscCheck == false || check == false);
    regFile = fopen("RegFile.txt", "a+");
    if (regFile == NULL) {
        printf("\tОшибка создания / открытия файла!\n");
        system("pause");
        system("cls");
    }
    else {
        if (EscCheck == false) {
            fseek(authFile, 0, SEEK_END);
            if (ftell(authFile) == 0) {
                fprintf(regFile, "%s %d %d\n", inputLogin, 1^225, 1^225);
                fclose(regFile);
                regFile = fopen("RegFile.txt", "r+");
            }
            if (regFile == NULL) {
                printf("\tОшибка создания / открытия файла!\n");
                system("pause");
                system("cls");
            }
            else {
                if (compare(&inputLogin, regFile) == true) {
                    int poz = ftell(regFile);
                    if (EscCheck == false) {
                        check = false;
                        do {
                            printf("Введите пароль: ");
                            inputPassword = writeStr(0);
                            if (strcmp(inputPassword, "EscMode") == 0) EscCheck = true;
                            else {
                                if (strlen(inputPassword) < MinPassSymbol || strlen(inputPassword)
> MaxPassSymbol) {
                                    printf("Возможный размер пароля от %d до %d символов!\n",
MinPassSymbol, MaxPassSymbol);
                                    system("pause");
                                    system("cls");
                                }
                                else if (compare(&inputPassword, authFile) == true) {
                                    printf("\tДанный пароль недоступен!\n");
                                    system("pause");
```

```
                        system("cls");
                    }
                    else CheckPass = true;
                    check = checkStr(&inputPassword, 4);
                }
            } while (EscCheck != true && CheckPass != true || check == false);
            printf("\tРегистрация выполнена!\n");
            system("pause");
            system("cls");
        }
        fseek(regFile, poz, SEEK_SET);
        inputPassword = encryption(inputPassword);
        fprintf(authFile, "%s %s %s ", inputLogin, inputPassword,
readFromFile(regFile));
        fprintf(authFile, "%s\n", readFromFile(regFile));

      }
      else {
        printf("\tВы не можете зарегестрироваться! Обратитесь к
главврачу.\n");
        system("pause");
        system("cls");
      }
    }
  }
  if (EscCheck == false || strcmp(inputLogin, "adm") != 0) {
    free(inputLogin);
    free(inputPassword);
  }
  if (authFile != NULL) fclose(authFile);
  else printf("\tОшибка закрытия файла!\n");
 }
}
}
```

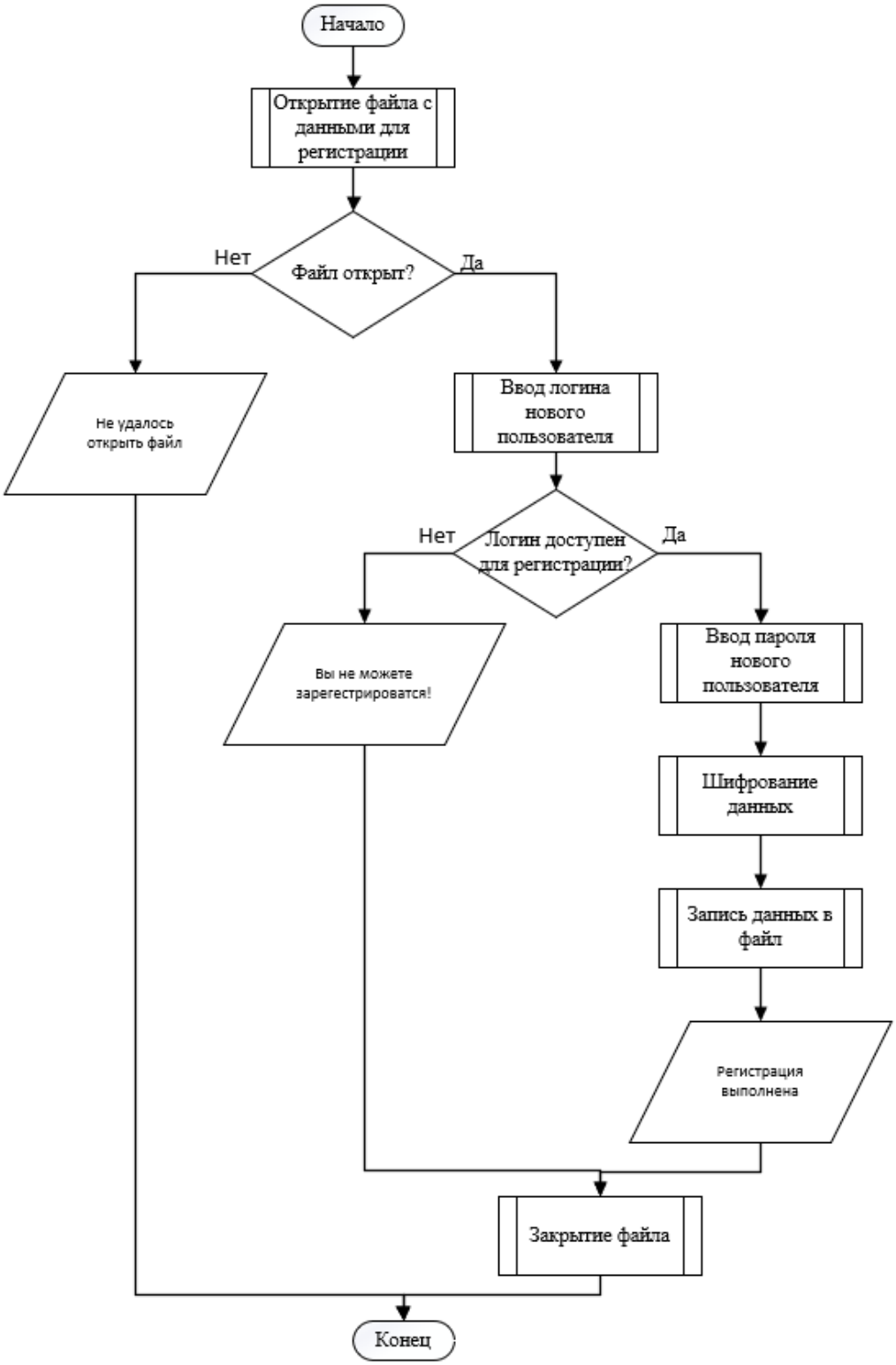


Рисунок 4.3 - Схема организации регистрации нового пользователя

5. ОПИСАНИЕ ПРОГРАММЫ

После запуска программы в консоли отображается меню, в котором пользователю предлагается осуществить вход под главврачом, врачом, зарегистрироваться в системе или же завершить работу программы. (рисунок 5.1)

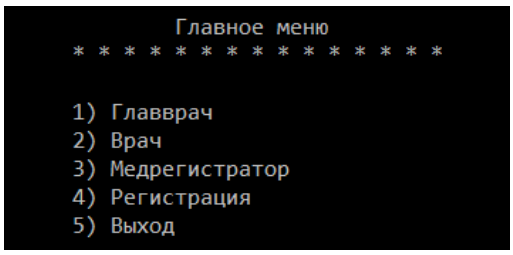


Рисунок 5.1 – Меню 1-го уровня

5.1 Вход в режиме главврача

Администратор вводит логин и пароль (рисунок 5.2).

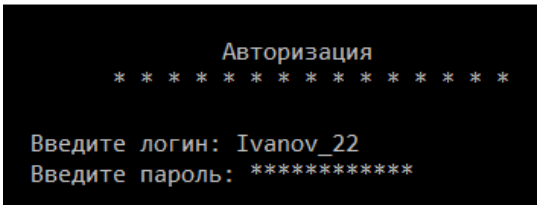


Рисунок 5.2 – Ввод данных главврача

У логина и пароля есть ограничения по количеству символов для большей безопасности системы. При вводе логина или пароля некорректной длины будет выводе сообщение об ошибке (рисунок 5.3 и 5.4).

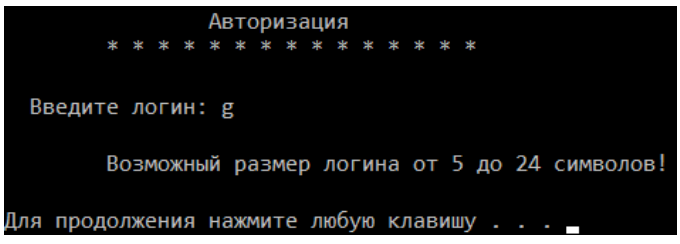


Рисунок 5.3 – Ошибка ввода логина

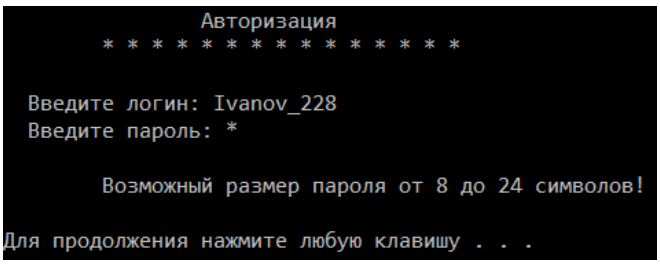


Рисунок 5.4 – Ошибка ввода пароля

Если логин или пароль введены неверно, то главврач увидит на экране сообщение об ошибке (рисунок 5.5) и может ввести данные заново.

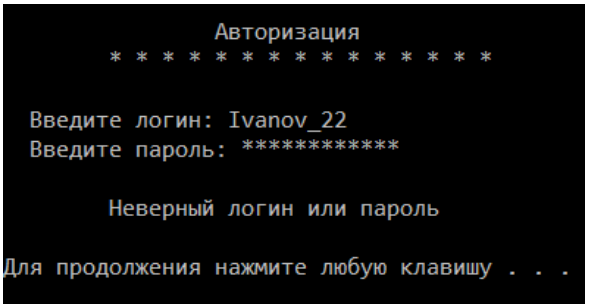


Рисунок 5.5 – Ошибка при авторизации

Если администратор ввёл правильный логин и пароль, то главврач увидит соответствующее сообщение (рисунок 5.6) и откроется меню главврача (рисунок 5.7). Меню администратора организовано соответствующими своему функционалу управляющими блоками.

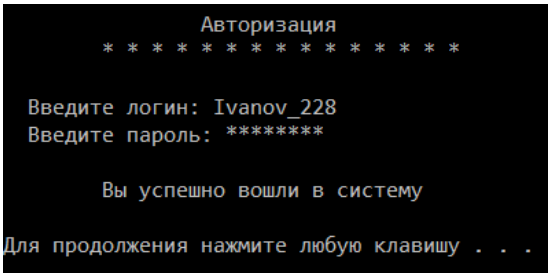


Рисунок 5.6 – Выполнен вход в систему

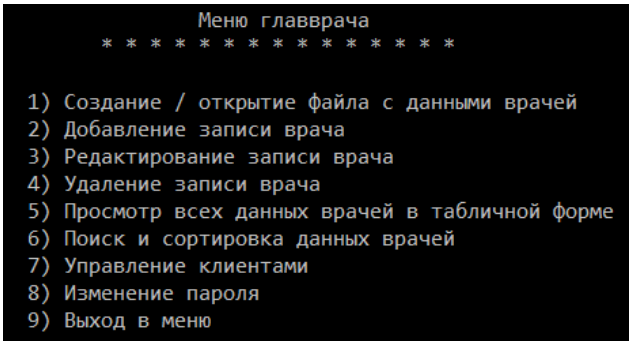


Рисунок 5.7 – Меню Главврача

5.1.1 Создание / открытия файла с данными

Для того чтобы начать работать с данными необходимо выбрать пункт номер один. Если файл создан, выведется соответствующее сообщение (Рис. 5.8).

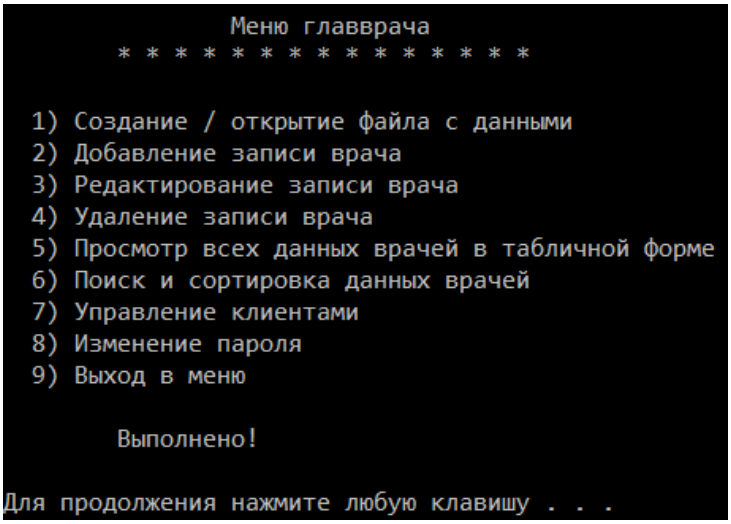


Рисунок 5.8 – Открытие файла

Выбрав это пункт меню повторно будет выведено сообщение о том, что файлу уже открыт (рисунок 5.9).

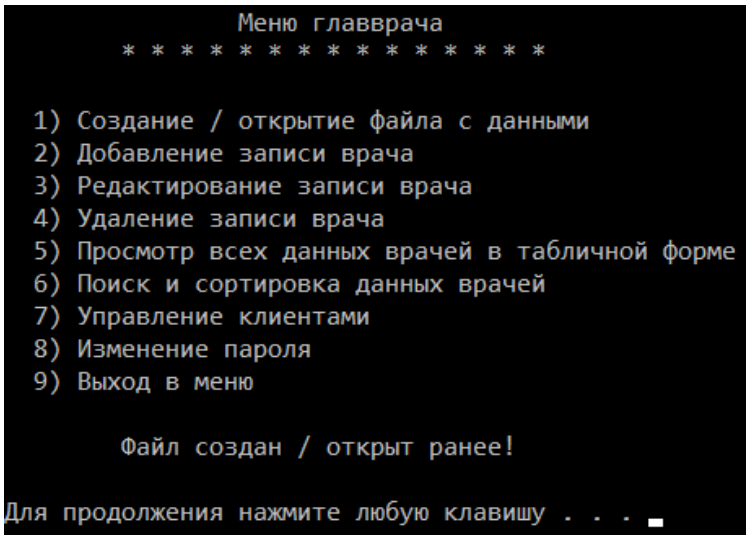


Рисунок 5.9 – Уведомление об открытом файле

При выборе других пунктов Меню при закрытом файле будет выведено соответствующее сообщение (рисунок 5.10).

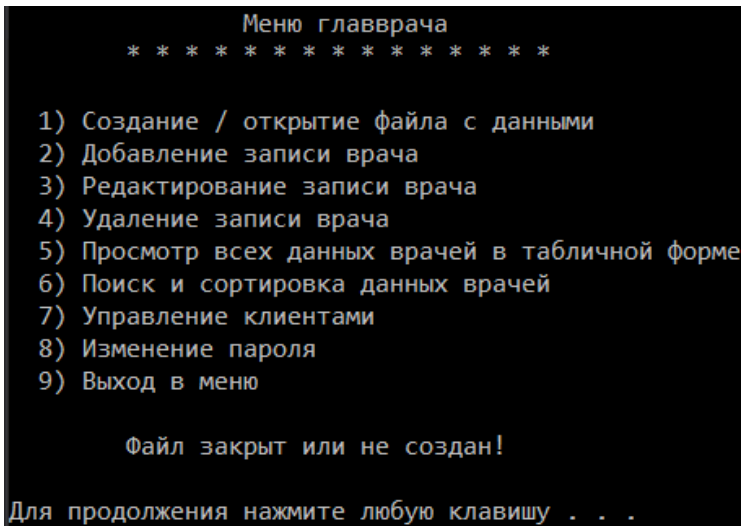


Рисунок 5.10 – Ошибка открытия файла

5.1.2 Добавление записи врача

При выборе 2-го пункта меню главврач добавляет данные о новом враче, а именно ФИО, специальность, категорию и логин. При успешном добавлении врача выводится соответствующее уведомление (рисунок 5.11).

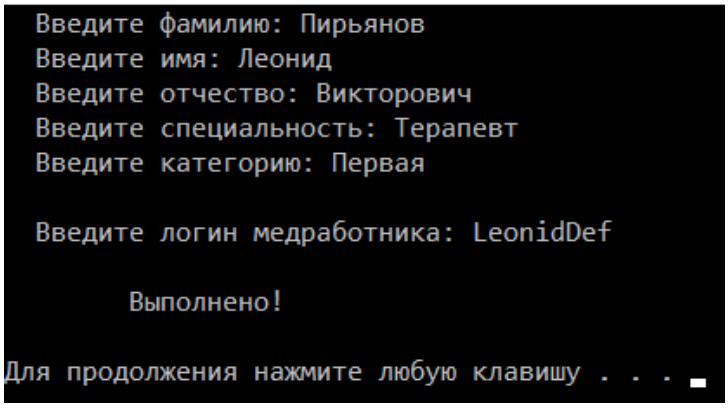


Рисунок 5.11 – Добавление врача

При вводе данных, не соответствующих требованиям, выведется соответствующее сообщение об ошибке (рисунок 5.12).

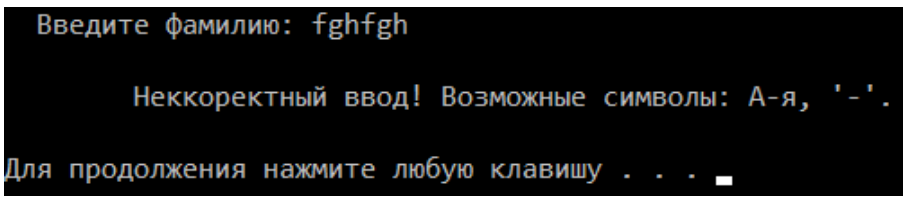


Рисунок 5.12 – Ошибка ввода

5.1.3 Меню редактирования записи врача

Для входа в данное меню необходимо выбрать пункт номер четыре. Появится таблица со списком всех врачей и запрос на ввод ID врача для редактирования (рисунок 5.14). Если такого кода нет в списке, то главврач увидит соответствующее сообщение на экране (рисунок 5.13). Если же код с таким номером найден, то на экране отобразится меню, где необходимо выбрать критерий изменения информации. После этого необходимо ввести значение, которое будет новым для данного врача (рисунок 5.15).

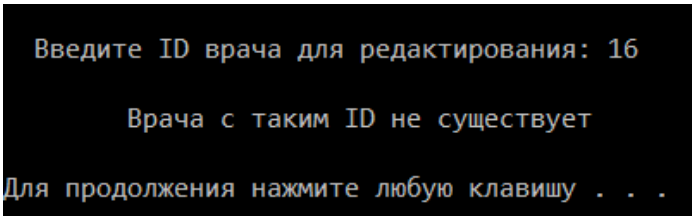


Рисунок 5.13 – Ошибка ввода данных

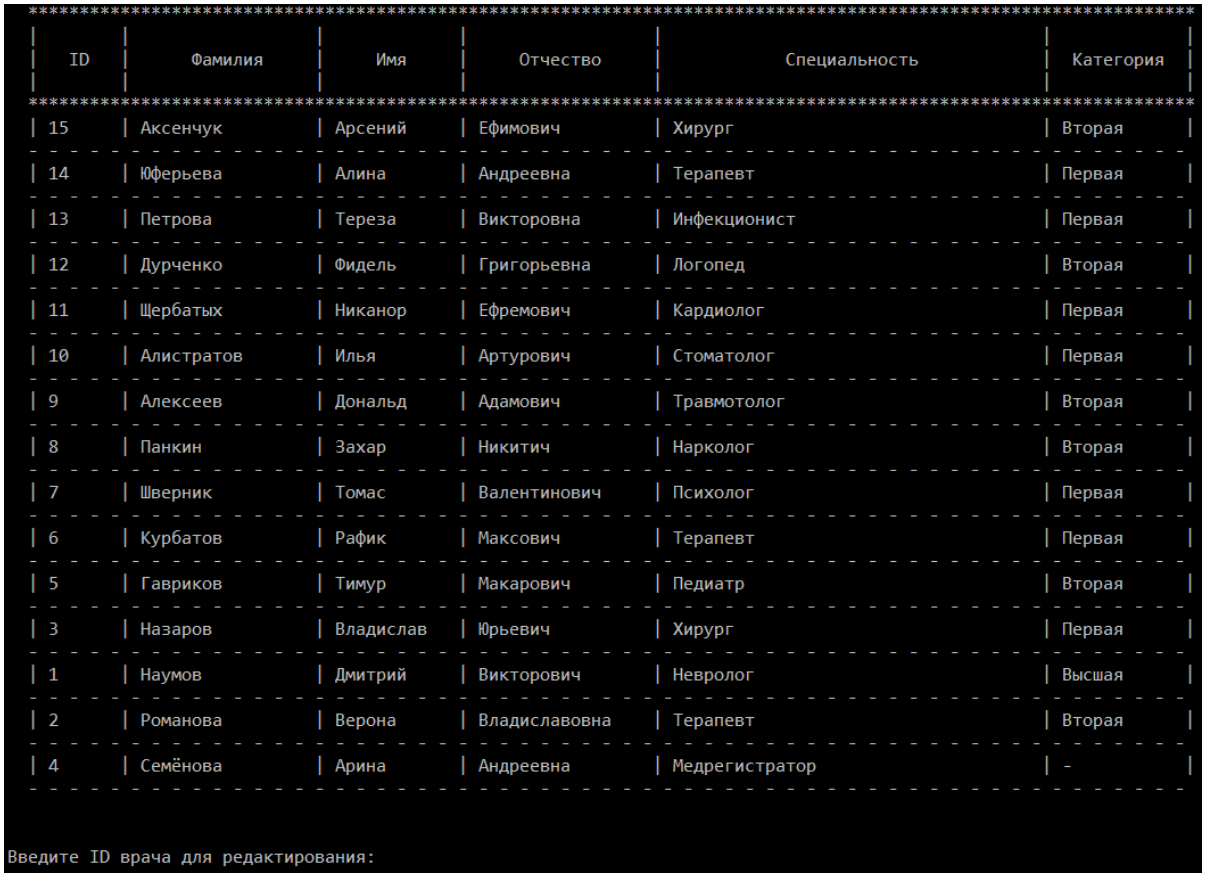


Рисунок 5.14 – Ввод ID врача для редактирования

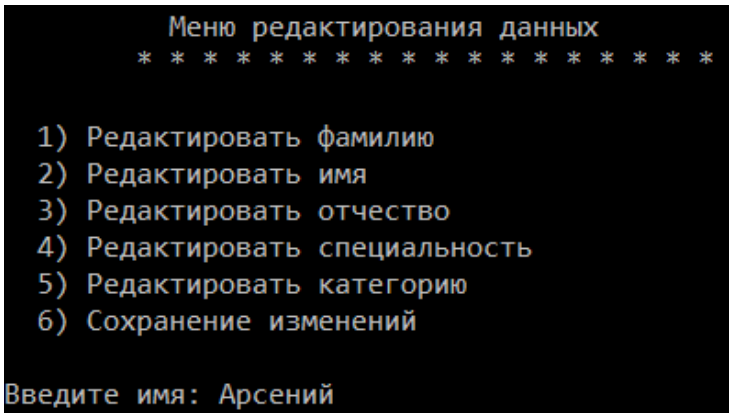


Рисунок 5.15 – Ввод нового значения

5.1.4 Удаление записи

При выборе 4-го пункта меню появится таблица со списком всех врачей и запрос на ввод ID врача для удаления (рисунок 5.18). Если такого кода нет в списке, то главврач увидит соответствующее сообщение на экране (рисунок 5.16). Если же код с таким номером найден, то врач с таким ID будет удалён и будет выведено соответствующее уведомление (рисунок 5.17).

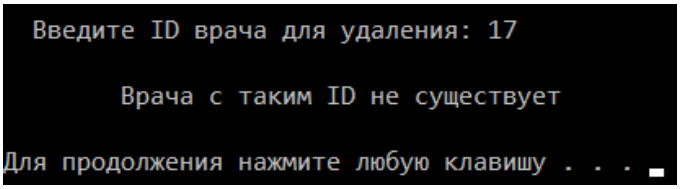


Рисунок 5.16 – Уведомление об удалении пользователя

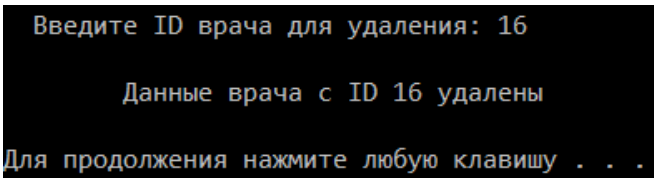
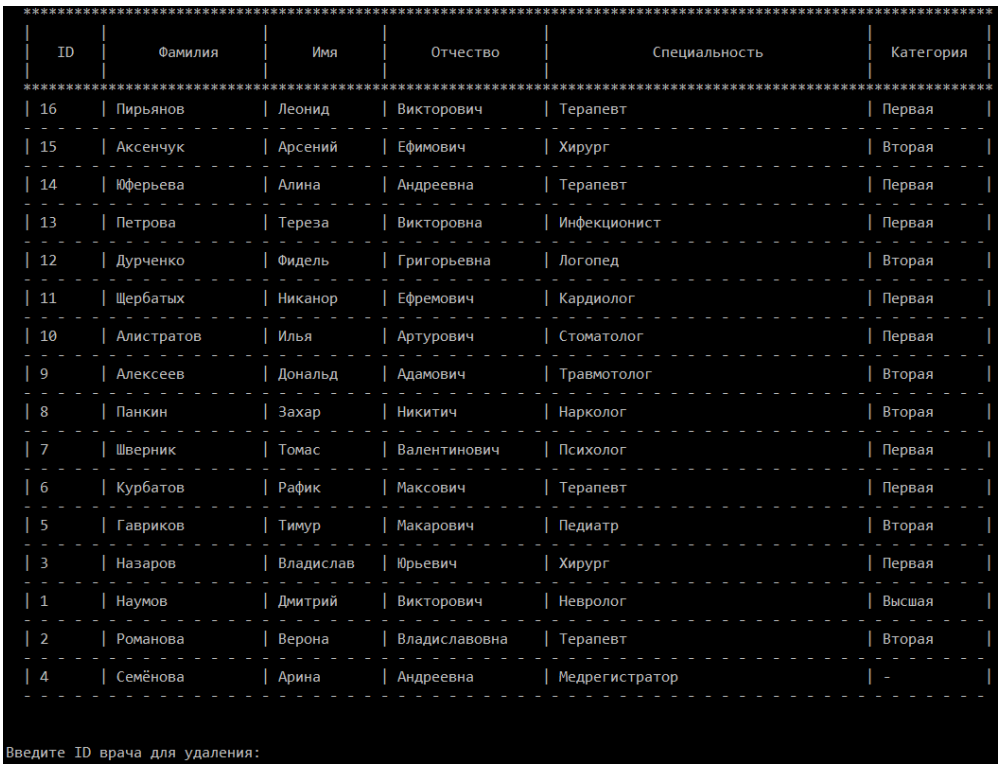


Рисунок 5.17 – Удаление врача



ID	Фамилия	Имя	Отчество	Специальность	Категория
16	Пирьянов	Леонид	Викторович	Терапевт	Первая
15	Аксенчук	Арсений	Ефимович	Хирург	Вторая
14	Юферьева	Алина	Андреевна	Терапевт	Первая
13	Петрова	Тереза	Викторовна	Инфекционист	Первая
12	Дурченко	Фидель	Григорьевна	Логопед	Вторая
11	Щербатых	Никанор	Ефремович	Кардиолог	Первая
10	Алистратов	Илья	Артурович	Стоматолог	Первая
9	Алексеев	Дональд	Адамович	Травмотолог	Вторая
8	Панкин	Захар	Никитич	Нарколог	Вторая
7	Шверник	Томас	Валентинович	Психолог	Первая
6	Курбатов	Рафик	Максович	Терапевт	Первая
5	Гавриков	Тимур	Макарович	Педиатр	Вторая
3	Назаров	Владислав	Юрьевич	Хирург	Первая
1	Наумов	Дмитрий	Викторович	Невролог	Высшая
2	Романова	Верона	Владиславовна	Терапевт	Вторая
4	Семёнова	Арина	Андреевна	Медрегистратор	-

Введите ID врача для удаления:

Рисунок 5.18 – Ввод ID врача для удаления

5.1.5 Просмотр всех данных врачей в табличной форме

При выборе 5-го пункта меню появится таблица со списком всех врачей (рисунок 5.20). Если в системе нет врачей, то выдаст соответствующее уведомление (рисунок 5.19).

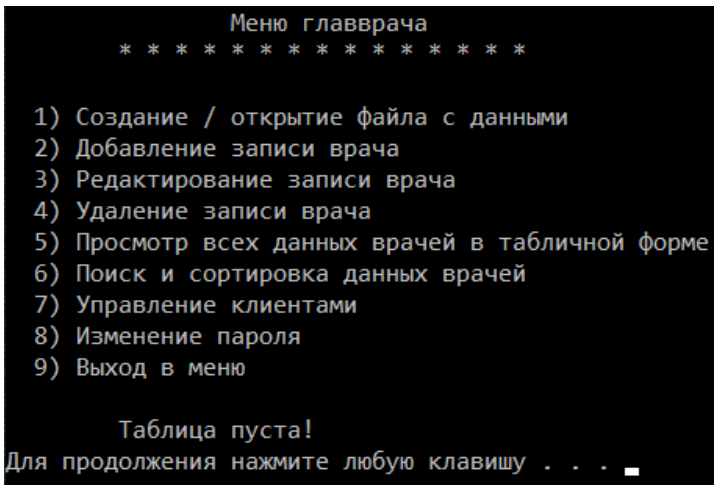
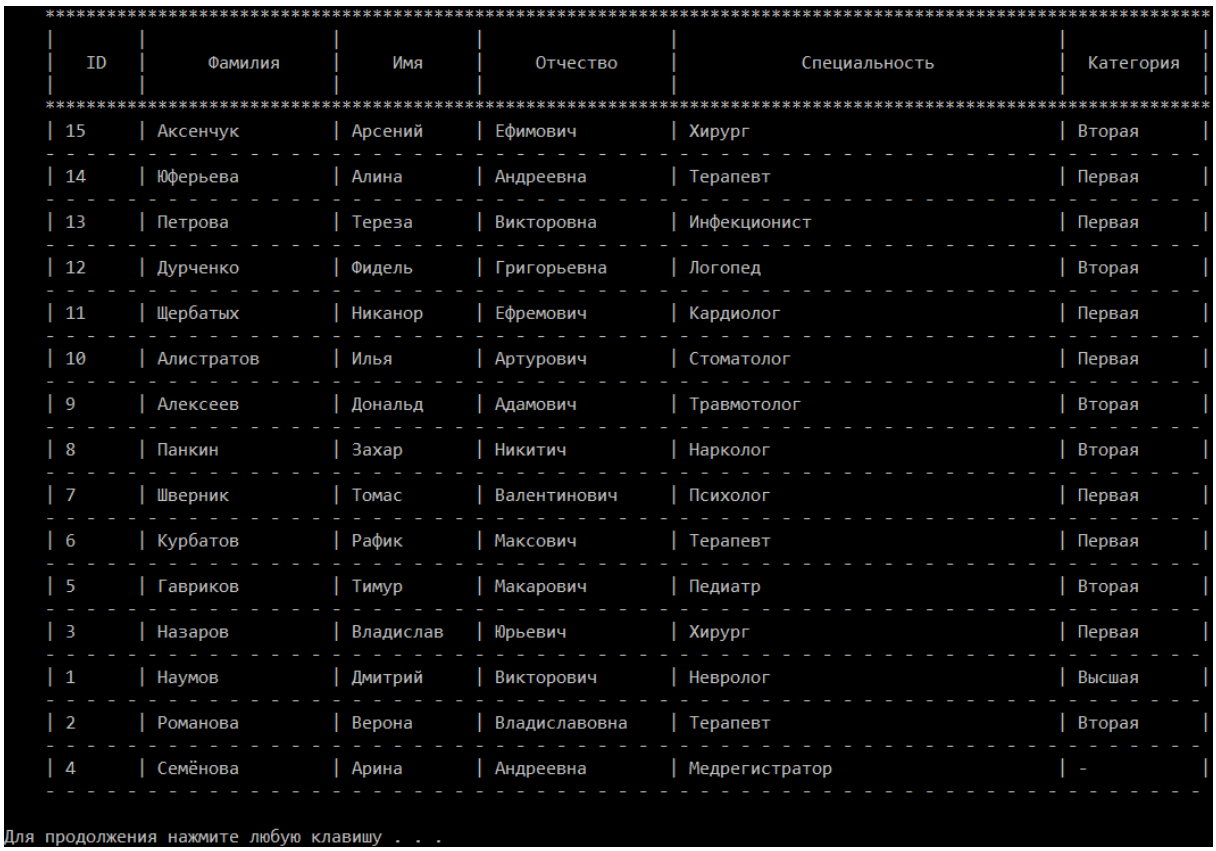


Рисунок 5.19 – Уведомление о пустой таблицы



ID	Фамилия	Имя	Отчество	Специальность	Категория
15	Аксенчук	Арсений	Ефимович	Хирург	Вторая
14	Юферьева	Алина	Андреевна	Терапевт	Первая
13	Петрова	Тереза	Викторовна	Инфекционист	Первая
12	Дурченко	Фидель	Григорьевна	Логопед	Вторая
11	Щербатых	Никанор	Ефремович	Кардиолог	Первая
10	Алистратов	Илья	Артурович	Стоматолог	Первая
9	Алексеев	Дональд	Адамович	Травмотолог	Вторая
8	Панкин	Захар	Никитич	Нарколог	Вторая
7	Шверник	Томас	Валентинович	Психолог	Первая
6	Курбатов	Рафик	Максович	Терапевт	Первая
5	Гавриков	Тимур	Макарович	Педиатр	Вторая
3	Назаров	Владислав	Юрьевич	Хирург	Первая
1	Наумов	Дмитрий	Викторович	Невролог	Высшая
2	Романова	Верона	Владиславовна	Терапевт	Вторая
4	Семёнова	Арина	Андреевна	Медрегистратор	-

Рисунок 5.20 – Вывод таблицы с данными врачей

5.1.6 Поиск и сортировка данных врачей

Для выполнения этой задачи потребуется выбрать 6-й пункт меню. На экране появляется меню выбора поиска, сортировки или выхода в предыдущее меню (рисунок 5.21). Выбрав пункт меню поиска данных, на экран выводится таблица данных всех врачей и меню для выбора данных, по которым будет осуществляться поиск данных (рисунок 5.22). После этого необходимо ввести данные для поиска (рисунок 5.24). Если программа находит одну или несколько записей, то она выводит их на экран (рисунок 5.23), в противном случае появится сообщение о том, что такая запись не найдена (рисунок 5.25).

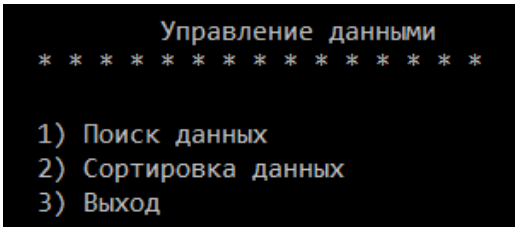


Рисунок 5.21 – Меню для поиска и сортировки данных

ID	Фамилия	Имя	Отчество	Специальность	Категория
4	Семёнова	Арина	Андреевна	Медрегистратор	-
2	Романова	Верона	Владиславовна	Терапевт	Вторая
1	Наумов	Дмитрий	Викторович	Невролог	Высшая
3	Назаров	Владислав	Юрьевич	Хирург	Первая
5	Гавриков	Тимур	Макарович	Педиатр	Вторая
6	Курбатов	Рафик	Максович	Терапевт	Первая
7	Шверник	Томас	Валентинович	Психолог	Первая
8	Панкин	Захар	Никитич	Нарколог	Вторая
9	Алексеев	Дональд	Адамович	Травмотолог	Вторая
10	Алистратов	Илья	Артурович	Стоматолог	Первая
11	Щербатых	Никанор	Ефремович	Кардиолог	Первая
12	Дурченко	Фидель	Григорьевна	Логопед	Вторая
13	Петрова	Тереза	Викторовна	Инфекционист	Первая
14	Юферьева	Алина	Андреевна	Терапевт	Первая
15	Аксенчук	Арсений	Ефимович	Хирург	Вторая

1) Поиск по ID врача
2) Поиск по фамилии
3) Поиск по имени
4) Поиск по отчеству
5) Поиск по специальности
6) Поиск по категории
7) Выход

Рисунок 5.22 – Меню для выбора параметра поиска

ID	Фамилия	Имя	Отчество	Специальность	Категория
10	Алистратов	Илья	Артурович	Стоматолог	Первая

Для продолжения нажмите любую клавишу . . .

Рисунок 5.23 – Вывод результата поиска

ID	Фамилия	Имя	Отчество	Специальность	Категория
4	Семёнова	Арина	Андреевна	Медрегистратор	-
2	Романова	Верона	Владиславовна	Терапевт	Вторая
1	Наумов	Дмитрий	Викторович	Невролог	Высшая
3	Назаров	Владислав	Юрьевич	Хирург	Первая
5	Гавриков	Тимур	Макарович	Педиатр	Вторая
6	Курбатов	Рафик	Максович	Терапевт	Первая
7	Шверник	Томас	Валентинович	Психолог	Первая
8	Панкин	Захар	Никитич	Нарколог	Вторая
9	Алексеев	Дональд	Адамович	Травмотолог	Вторая
10	Алистратов	Илья	Артурович	Стоматолог	Первая
11	Щербатых	Никанор	Ефремович	Кардиолог	Первая
12	Дурченко	Фидель	Григорьевна	Логопед	Вторая
13	Петрова	Тереза	Викторовна	Инфекционист	Первая
14	Юферьева	Алина	Андреевна	Терапевт	Первая
15	Аксенчук	Арсений	Ефимович	Хирург	Вторая

Введите ID врача:

Рисунок 5.24 – Ввод данных для поиска

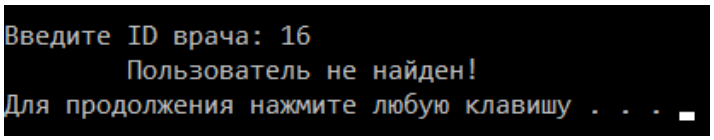


Рисунок 5.25 – Оповещение об ошибке поиска

Выбрав пункт меню сортировки данных, выводится таблица данных всех врачей и меню для выбора данных, по которым будет осуществляться поиск данных (рисунок 5.26). После этого на экран выводится отсортированная по выбранному параметру таблица с данными врачей. На рисунке 5.27 предоставлен пример сортировки фамилий по алфавиту.

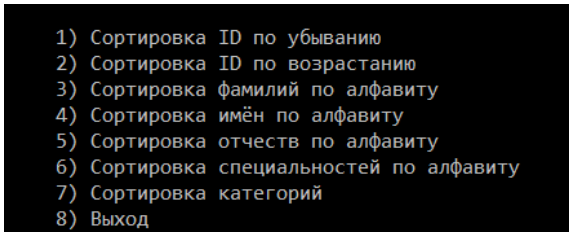


Рисунок 5.26 – Меню для выбора параметра сортировки

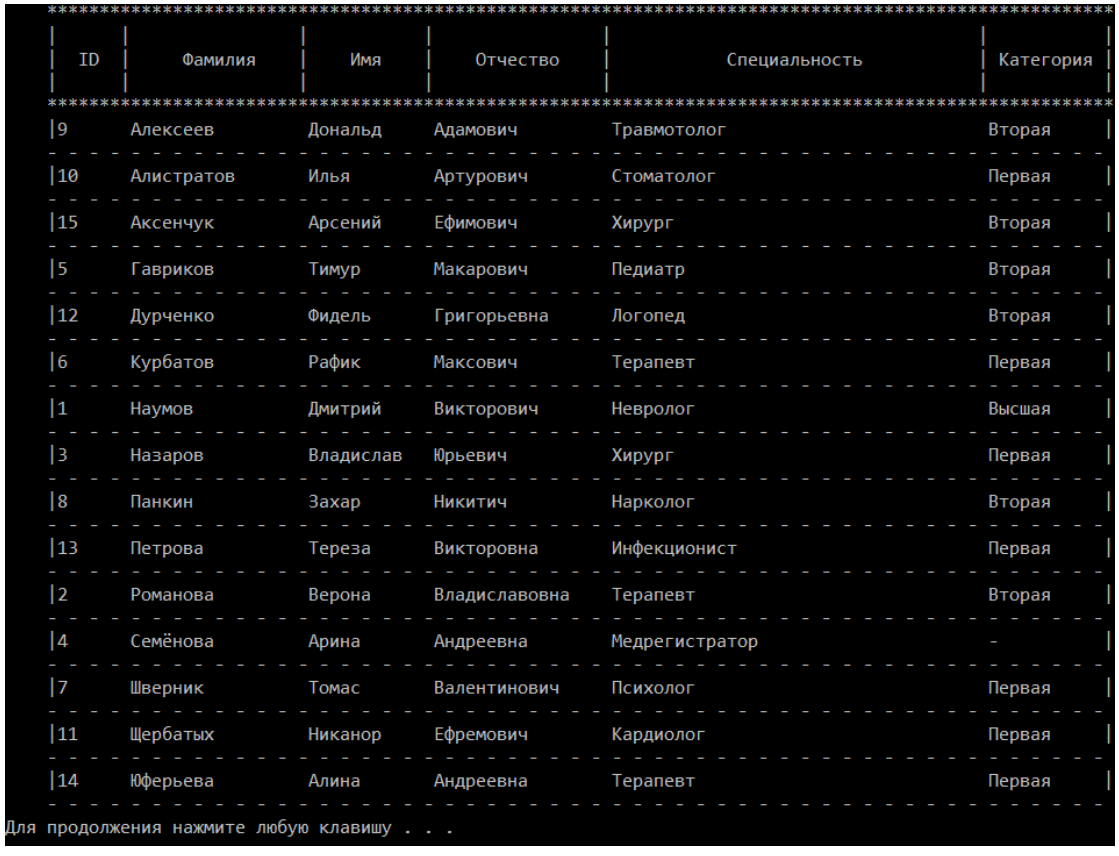


Рисунок 5.27 – Вывод результата сортировки данных

5.1.7 Управление клиентами

При выборе 7-го пункта меню открывается меню врача, которое выполняет функции работы с пациентами.

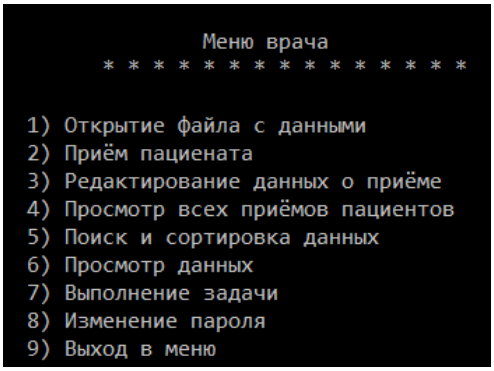


Рисунок 5.28 – Меню управления клиентами

5.1.8 Изменение пароля

В 8-м пункте меню происходит изменение пароля авторизованного пользователя (рисунок 5.28). При неправильном вводе старого пароля на экран выводится соответствующее уведомление (рисунок 5.29).

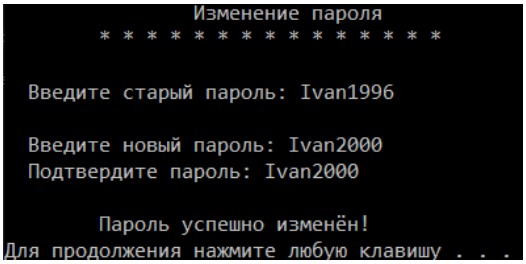


Рисунок 5.28 – Изменение пароля пользователя

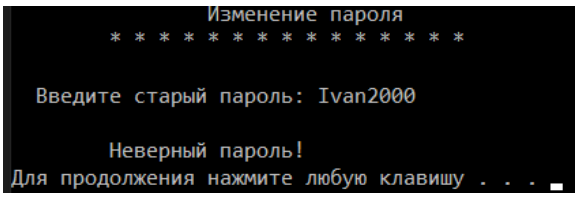


Рисунок 5.29 – Уведомление о неправильно вводе

5.2 Вход в режиме пользователя

Вход врача в систему осуществляется так же, как и вход главврача. Если врач с таким логином существует и пароль введен, верно, то на экране

появится меню врача (рисунок 5.30). В данном меню пользователь может выбрать пункт приёма пациента, редактирования и удаления данных, просмотра данных в табличной форме, поиск и сортировка данных. Также есть пункт выполнения задачи.

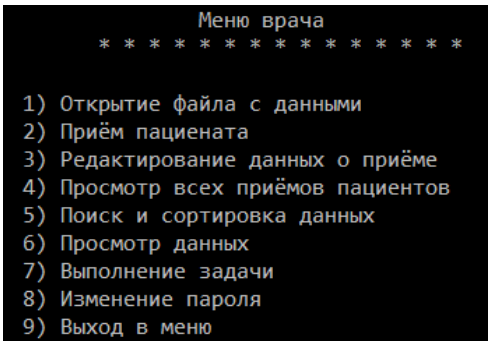


Рисунок 5.30 – Меню врача

5.2.1 Приём пациентов

При выборе пункта меню под номером 2 осуществляется добавление нового приёма, производится ввод и проверка данных пациента на наличие их в системе. Если такого пациента нет, то выводится соответствующее сообщение (рисунок 5.31). При совпадении данных производится ввод диагноза и стоимости лечения пациента (рисунок 5.32).

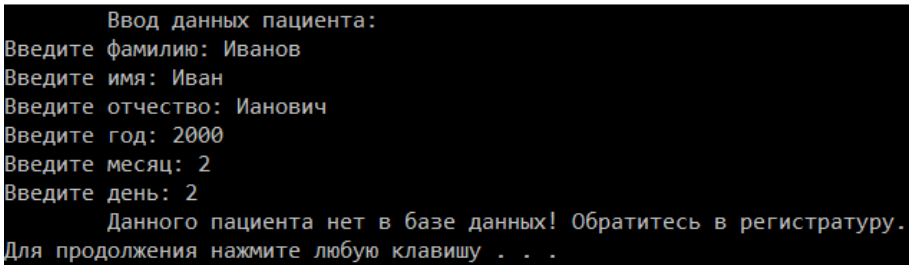


Рисунок 5.32 – Ввод нового приёма

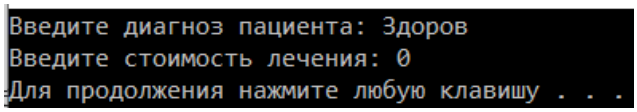


Рисунок 5.33 – Ввод данных приёма

5.2.2 Выполнение задачи

При выборе 7-го пункта меню производится вывод на экран топ 5 самых частых диагнозов (рисунок 5.34).

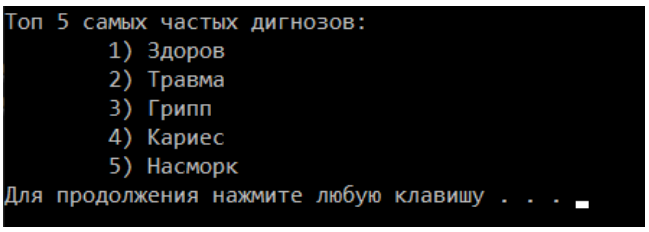


Рисунок 5.34 – Выполнение задачи

5.3 Вход в режиме медрегистратора

Вход врача в систему осуществляется так же, как и вход главврача. Если медрегистратор с таким логином существует и пароль введен, верно, то на экране появится меню медрегистратора (рисунок 5.35). В данном меню пользователь может выбрать пункт добавление, удаление и редактирование записи пациента, просмотр всех данных в табличной форме, изменение пароля. Также есть поиск и сортировка данных.

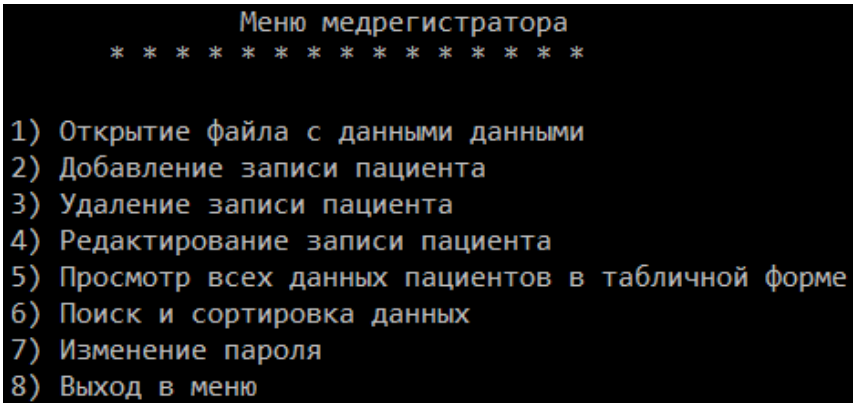


Рисунок 5.35 – Меню медрегистратора

ЗАКЛЮЧЕНИЕ

В наше время никто уже не может представить свою жизнь без использования информационных технологий. С каждым годом их влияние на жизнь человека становится всё сильнее.

В данной курсовой работе была исследована тема учета клиентов платной клиники. Сейчас невозможно представить работу врача без использования программ для учёта пациентов, т.к. это очень удобно и позволяет тратить меньше времени на заполнение документации, а соответственно больше уделять времени пациентам, что положительно сказывается на их лечении. Чем более простой в использовании, удобной и практичной является работа с информацией, тем более удобной и продуктивной будет работа врачей. Поэтому важно досконально продумать структуру программы по усовершенствованию работы для того, чтобы врачей направляли все свои силы на лечение пациентов.

В результате выполнения работы был разработан эффективный алгоритм, на основе которого и было создано приложение. Для создания приложения потребовалось разработать пользовательские функции, которые дают возможность добавлять, редактировать, удалять, сортировать данные, осуществлять поиск и фильтрацию информации по различным критериям. Стоит заметить, что в данном приложении реализовано разделение доступа между обычными пользователями и администрацией. Это сделано для разделения обязанностей пользователей, ограничения доступа к некоторым функциям программы, чтобы обеспечить программу достаточным уровнем безопасности, уменьшить количество ошибок со стороны пользователей, которые не всегда знакомы с работой программы и могут неправильно её эксплуатировать. Также в программе предусмотрены всевозможные проверки на ошибочные ситуации, чтобы исключить ошибочного использования программы и сделать её более удобной в использовании различными пользователями.

Итак, в конечном итоге было разработано приложение, которое совмещает в себе удобный интерфейс, лёгкое для понимания меню и различные функции работы с данными, содержащими различную информацию о врачах, пациентах и приёмах. Благодаря эффективному алгоритму и хорошей проектировке, данная программа сможет в дальнейшем легко совершенствоваться и редактироваться.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Луцик, Ю. А. Основы алгоритмизации и программирования: язык Си: учебно-метод. пособие / Ю. А. Луцик, А. М. Ковальчук, Е. А. Сасин. – Минск: БГУИР, 2015. – 170 с.

[2] Дейтел Х. М., Дейтел П. Дж. Как программировать на С: Четверное издание. Пер. с англ. ─ М.: ООО «Бином-Пресс»2006г. ─ 912 с.

[3] Батура М. П. Основы алгоритмизации и программирования. Язык СИ: учеб. Пособие / М. П. Батура, В. Л. Бусько, А. Г. Корбит, Т. М. Кривоносова. ─ Минск: БГУИР, 2007. ─ 240 с.

ПРИЛОЖЕНИЕ А
ЛИСТИНГ КОДА ПРОГРАММЫ

1.CourseWork_OKP.c

```
int main() {
   int ID;
   system("chcp 1251>null");
   while (1) {
      switch (main_menu()) {
      case '1':
         if((ID = AuthFunction(1)) != 0) admin_menu(ID);
         break;
      case '2':
         if ((ID = AuthFunction(2)) != 0) user_menu(ID);
         break;
      case '3':
         if ((ID = AuthFunction(3)) != 0) regist_menu(ID);
         break;
      case '4':
         registration();
         break;
      case '5':
         return 0;
      }
   }
}
```

CoursFLib.c

```
void readingDataDoc(Doctors** head, FILE* Doc) {
   Doctors* p = NULL;
   int count = 0, tmp = 0;
   fseek(Doc, 0, SEEK_END);
   if (ftell(Doc) == 0);
   else {
      p = malloc(sizeof(Doctors));
      if ((p = malloc(sizeof(Doctors))) == NULL) {
         printf("\tОшибка выделения памяти для стека!\n");
      }
      tmp = ftell(Doc);
      fseek(Doc, 0, SEEK_SET);
      do {
         if (count == 0) {
            p->prev = NULL;
            p->ID = atoi(readFromFile(Doc));
            p->surname = readFromFile(Doc);
            p->name = readFromFile(Doc);
            p->middleName = readFromFile(Doc);
            p->specialty = readFromFile(Doc);
            p->category = readFromFile(Doc);
```

```
            count++;
        }
        else {
            p = malloc(sizeof(Doctors));
            p->prev = *head;
            p->ID = atoi(readFromFile(Doc));
            p->surname = readFromFile(Doc);
            p->name = readFromFile(Doc);
            p->middleName = readFromFile(Doc);
            p->specialty = readFromFile(Doc);
            p->category = readFromFile(Doc);
            count++;
        }
        *head = p;
    } while (ftell(Doc) < tmp);
  }
}
void readingDataCust(Customers** head, FILE* Doc) {
  Customers* p = NULL;
  int count = 0, tmp = 0;
  fseek(Doc, 0, SEEK_END);
  if (ftell(Doc) == 0);
  else {
    p = malloc(sizeof(Customers));
    if ((p = malloc(sizeof(Customers))) == NULL) {
        printf("\tОшибка выделения памяти для стека!\n");
    }
    tmp = ftell(Doc);
    fseek(Doc, 0, SEEK_SET);
    do {
        if (count == 0) {
            p->prev = NULL;
            p->ID = atoi(readFromFile(Doc));
            p->surname = readFromFile(Doc);
            p->name = readFromFile(Doc);
            p->middleName = readFromFile(Doc);
            p->dateBirthday.day = atoi(readFromFile(Doc));
            p->dateBirthday.month = atoi(readFromFile(Doc));
            p->dateBirthday.year = atoi(readFromFile(Doc));
            count++;
        }
        else {
            p = malloc(sizeof(Customers));
            p->prev = *head;
            p->ID = atoi(readFromFile(Doc));
            p->surname = readFromFile(Doc);
            p->name = readFromFile(Doc);
            p->middleName = readFromFile(Doc);
            p->dateBirthday.day = atoi(readFromFile(Doc));
            p->dateBirthday.month = atoi(readFromFile(Doc));
            p->dateBirthday.year = atoi(readFromFile(Doc));
            count++;
```

```
            }
            *head = p;
        } while (ftell(Doc) < tmp);
    }
}
void readingDataApp(Appeals** head, FILE* Doc) {
    Appeals* p = NULL;
    int count = 0, tmp = 0;
    fseek(Doc, 0, SEEK_END);
    if (ftell(Doc) == 0);
    else {
        p = malloc(sizeof(Appeals));
        if ((p = malloc(sizeof(Appeals))) == NULL) {
            printf("\tОшибка выделения памяти для стека!\n");
        }
        tmp = ftell(Doc);
        fseek(Doc, 0, SEEK_SET);
        do {
            if (count == 0) {
                p->prev = NULL;
                p->NumApp = atoi(readFromFile(Doc));
                p->IDdoc = atoi(readFromFile(Doc));
                p->IDcust = atoi(readFromFile(Doc));
                p->dateAppeal.day = atoi(readFromFile(Doc));
                p->dateAppeal.month = atoi(readFromFile(Doc));
                p->dateAppeal.year = atoi(readFromFile(Doc));
                p->diagnosis = readFromFile(Doc);
                p->costOfTreatment = atoi(readFromFile(Doc));
                count++;
            }
            else {
                p = malloc(sizeof(Appeals));
                p->prev = *head;
                p->NumApp = atoi(readFromFile(Doc));
                p->IDdoc = atoi(readFromFile(Doc));
                p->IDcust = atoi(readFromFile(Doc));
                p->dateAppeal.day = atoi(readFromFile(Doc));
                p->dateAppeal.month = atoi(readFromFile(Doc));
                p->dateAppeal.year = atoi(readFromFile(Doc));
                p->diagnosis = readFromFile(Doc);
                p->costOfTreatment = atoi(readFromFile(Doc));
                count++;
            }
            *head = p;
        } while (ftell(Doc) < tmp);
    }
}
void readingDataAccount(Accounts** head, FILE* Doc) {
    Accounts* p = NULL;
    int count = 0, tmp = 0, countN = 0;
    fseek(Doc, 0, SEEK_END);
    if (ftell(Doc) == 0) {
```

```
        printf("\tФайл пуст!\n");
        system("pause");
        system("cls");
    }
    else {
        p = malloc(sizeof(Accounts));
        if ((p = malloc(sizeof(Accounts))) == NULL) {
            printf("\tОшибка выделения памяти для стека!\n");
        }
        tmp = ftell(Doc);
        fseek(Doc, 0, SEEK_SET);
        do {
            if (count == 0) {
                p->login = encryption(readFromFile(Doc));
                p->password = encryption(readFromFile(Doc));
                p->mode = atoi(encryption(readFromFile(Doc)));
                p->IDdoc = atoi(encryption(readFromFile(Doc)));
                p->prev = NULL;
                count++;
            }
            else {
                p = malloc(sizeof(Doctors));
                p->prev = *head;
                p->login = encryption(readFromFile(Doc));
                p->password = encryption(readFromFile(Doc));
                p->mode = atoi(encryption(readFromFile(Doc)));
                p->IDdoc = atoi(encryption(readFromFile(Doc)));
                count++;
            }
            *head = p;
        } while (ftell(Doc) < tmp);
    }
}

void fprintfDoc(Doctors** p,  FILE* File) {

    fprintf(File, "%d ", (*p)->ID);
    fprintf(File, "%s ", (*p)->surname);
    fprintf(File, "%s ", (*p)->name);
    fprintf(File, "%s ", (*p)->middleName);
    fprintf(File, "%s ", (*p)->specialty);
    fprintf(File, "%s\n", (*p)->category);
}
void rewriteDoc(Doctors** head, FILE* File) {
    Doctors* p = NULL;
    if (File != NULL) {
        fclose(File);
        File = fopen("ListOfDoctors.txt", "w+");
    }
    if (File == NULL) {
        printf("\tОшибка создания / открытия файла!\n");
    }
```

```
    else {
        if (*head == NULL);
        else {
            p = *head;
            do {
                fprintfDoc(&p, File);
                p = p->prev;
            } while (p != NULL);
        }
    }
}

void fprintfAccount(Accounts** p, FILE* File) {;
    fprintf(File, "%s ", encryption((*p)->login));
    fprintf(File, "%s ", encryption((*p)->password));
    fprintf(File, "%d ", (*p)->mode ^ 225);
    fprintf(File, "%d\n", (*p)->IDdoc ^ 225);
}
FILE* rewriteAccount(Accounts** head, FILE* File) {
    Accounts* p = *head;
    if (File != NULL) {
        fclose(File);
        File = fopen("Authentification.txt", "w+");
    }
    if (File == NULL) {
        printf("\tОшибка создания / открытия файла!\n");
    }
    else {
        if (*head == NULL);
        else {
            p = *head;
            do {
                fprintfAccount(&p, File);
                p = p->prev;
            } while (p != NULL);
        }
    }
Return File;
}

void fprintfCust(Customers** p, FILE* File) {
    fprintf(File, "%d ", (*p)->ID);
    fprintf(File, "%s ", (*p)->surname);
    fprintf(File, "%s ", (*p)->name);
    fprintf(File, "%s ", (*p)->middleName);
    fprintf(File, "%d ", (*p)->dateBirthday.day);
    fprintf(File, "%d ", (*p)->dateBirthday.month);
    fprintf(File, "%d\n", (*p)->dateBirthday.year);
}
FILE* rewriteCust(Customers** head, FILE* File) {
    Customers* p = NULL;
    if (File != NULL) {
```

```
        fclose(File);
        File = fopen("ListOfCustomers.txt", "w+");
    }
    if (File == NULL) {
        printf("\tОшибка создания / открытия файла!\n");
    }
    else {
        if (*head == NULL);
        else {
            p = *head;
            do {
                fprintfCust(&p, File);
                p = p->prev;
            } while (p != NULL);
        }
    }
Return File;
}

void fprintfApp(Appeals** p, FILE* File) {
    fprintf(File, "%d ", (*p)->NumApp);
    fprintf(File, "%d ", (*p)->IDdoc);
    fprintf(File, "%d ", (*p)->IDcust);
    fprintf(File, "%d ", (*p)->dateAppeal.day);
    fprintf(File, "%d ", (*p)->dateAppeal.month);
    fprintf(File, "%d ", (*p)->dateAppeal.year);
    fprintf(File, "%s ", (*p)->diagnosis);
    fprintf(File, "%d\n", (*p)->costOfTreatment);
}
FILE* rewriteApp(Appeals** head, FILE* File) {
    Appeals* p = NULL;
    if (File != NULL) {
        fclose(File);
        File = fopen("ListOfAppeals.txt", "w+");
    }
    if (File == NULL) {
        printf("\tОшибка создания / открытия файла!\n");
    }
    else {
        if (*head == NULL);
        else {
            p = *head;
            do {
                fprintfApp(&p, File);
                p = p->prev;
            } while (p != NULL);
        }
    }
Return File;
}

bool compare(char** str, FILE* File, int mode) {
```

```
    fseek(File, 0, SEEK_END);
    int tmp = ftell(File);
    if (tmp != 0) {
        fseek(File, 0, SEEK_SET);
        do {
            if (mode == 2) readFromFile(File);
            if ((strcmp(*str, readFromFile(File))) == 0) return true;
        } while (ftell(File) < tmp);
        return false;
    }
    else return false;
}

void DoctorsHat() {
    puts("
**************************************************************************************
**************************");
    puts("    |     |             |         |             |                         |         |");
    puts("    | ID |    Фамилия    |   Имя    |   Отчество    |         Специальность          |
Категория |");
    puts("    |     |             |         |             |                         |         |");
    puts("
**************************************************************************************
**************************");
}
void CustomersHat() {
    puts("
*********************************************************************************"
);
    puts("    |       |              |           |              |             |");
    puts("    |   ID   |     Фамилия     |    Имя     |    Отчество     | Дата рождения |");
    puts("    |       |              |           |              |             |");
    puts("
*********************************************************************************"
);
}
void AppealsHat() {
    puts("
**************************************************************************************
*****************************************");
    puts("    |          |        |         |           |                                  |                |");
    puts("    |   Номер   |    ID   |    ID    | Дата приёма |                 Диагноз                  |
Стоимость лечения |");
    puts("    | обращения | Доктора | Пациента |           |                                  |
|");
    puts("    |          |        |         |           |                                  |                |");
    puts("
**************************************************************************************
*****************************************");
}
```

https://youtu.be/lwHVrMjTtn4