Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина: Технология разработки программного обеспечения

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему

**Разработка программы учета выпускаемой**

**предприятием продукции**

Студент:

гр.­­­­­ 113802 Скачко А. Г.

Руководитель:

к.т.н. Кабариха В.А.

Минск 2022

# Содержание

Содержание 2

1 Требования к программе 3

1.1 Исходные требования к курсово работе 3

1.2 Функциональные требования 3

1.3 Требования к програмнной реализации 7

2 Конструирование программы 9

2.1 Разработка структуры программы 9

2.2 Выбор способа организации данных 9

2.3 Разработка перечня пользовательских функций программы 11

3 Разработка алгоритмов работы программы 18

3.1 Алгоритм функции main 18

3.2 Алгоритм функции surnamePatientsEdit 18

3.3 Алгоритм функции deletePatient 18

4 Описание работы программы 19

4.1 Вход в программу 17

4.3 Модуль администратора 20

4.4 Модуль пользователя 30

4.5 Исключительные ситуации 38

Приложение А (обязательное) Модульная структура программы 46

Приложение Б (обязательное) Алгоритм функции main 47

Приложение В (обязательное) Алгоритм функции surnamePatientsEdit 48

Приложение Г (обязательное) Алгоритм функции deletePatient 49

Приложение Д (обязательное) Листинг кода с комментариями 50

# Требования к программе

Разработать программу учета выпускаемой предприятием продукции.

Сведения о выпущенной продукции включают: дату; номер цеха; наименование продукции; количество выпущенных единиц; ФИО ответсвенного по цеху в данный день

Индивидуальное задание: для заданного цеха необходимо вывести количество выпущенных изделий по каждому наименованию за требуемый период времени (требуемый период времени вводится с клавиатуры).

Реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя, как минимум три вида поиска, как минимум три вида сортировки.

## 1.1 Исходные требования к курсовой работе

1. Язык программирования С++.
2. Среда разработки Microsoft Visual Studio версии 2010 и выше.
3. Вид приложения – консольное.
4. Парадигма программирования – процедурная.
5. Способ организации данных – структуры.
6. Способ хранения данных – текстовые файлы.
7. Каждая логически завершенная задача программы должна быть реализована в виде функции.
8. К защите курсовой работы предоставляются: консольное приложение и пояснительная записка.
9. Текст пояснительной записки оформляется в соответствии со стандартом предприятия «СТП 01–2017».

## 1.2 Функциональные требования к курсовой работе

В начале программы осуществляется чтение информации из файлов в массивы, далее все операции с данными производятся посредством массивов, однако при каждом изменении информации дополнительно происходит запись информации из массива в файл.

Первым этапом работы программы является авторизация – предоставление прав доступа. В рамках данного этапа необходимо считать данные из файла, содержащего учетные записи пользователей следующего вида:

* login;
* hash;
* salt;
* role;
* access.

После ввода пользователем своих персональных данных (логина и пароля) и сверки с данными, находящимися в файле учетных записей, необходимо предусмотреть возможность входа в качестве администратора (role = 1) или в качестве пользователя (role = 0).

Если файла с учетными записями не существует, то необходимо его программно создать и сделать учетную запись главного администратора.

Регистрация новых пользователей осуществляется самим пользователем путем ввода желаемых логина и пароля, и ожидания подтверждения администратором новой учетной записи или главным администратором при редактировании учетной записи. Для реализации этого способа в структуре учетных записей аккаунтов предусмотрено поле access. По умолчанию access = 0 при попытке зарегистрироваться; далее администратор при подтверждении заявки меняет значение на access = 1 и тем самым подтверждает новую учетную запись; пользователь может осуществить вход в систему. Также реализована проверка новых учетных записей на уникальность логина.

По соображениям безопасности пароли учетных записей за хешированы. Выполнено двойное хеширование пароля с солью (соль – случайная строка, специально сгенерированная для данной учетной записи). И соль, и результат применения хеш-функции к паролю с солью храниться в файле с учетными записями.

Вторым этапом работы программы является собственно работа с данными, которая становится доступной только после прохождения авторизации. Данные хранятся в отдельном файле продукции.

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

Допускается создавать несколько учетных записей администраторов, только в том случаем, если администратор существующей учетной записи меняет role = 1 вручную.

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

1. Работа с учетными записями:

* Просмотр всех учетных записей;
* Редактирование учетных записей:

1. Изменение логина;
2. Изменение роли;
3. Одобрение доступа пользователя;

* Удаление учётной записи;

1. Обработка данных продукции:

* Просмотр информации об всей продукции (выполняется автоматически);
* Поиск:

1. Поиск по году, когда продукт был произведён;
2. Поиск по названию продукции;
3. Поиск по количеству выпущенных единиц;
4. Поиск по имени ответсвенного по цеху в этот день;

* Сортировка:

1. Сортировка по году, когда продукция была произведена;
2. Сортировка по номеру цеха;
3. сортировка по количеству выпущенных единиц;

3. Вывод количество выпущенных изделий по определённому наименовани

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

1. Просмотр информации обо продукции;
2. Вывод количество выпущенных изделий по определённому наименованию
3. Поиск:

* поиск по году, кода продукт был произведён;
* поиск по названию продкции;
* поиск по колчеству выпущенных единиц;
* поиск по имени ответсвенного по цеху в этот день;

1. Сортировка:

* сортировка по году, когда продукция была произведена;
* сортировка по номеру цеха;
* сортировка по количеству выпущенных единиц;

1. Изменение собственного пароля;

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

Предусмотреть:

1. Обработку исключительных ситуаций:
   * файла с данными для чтения не существует;
   * пользователь с таким именем уже существует;
   * ничего не найдено по результатам поиска;
   * номер удаляемой записи, выбор действия, номер редактирования записи и т.д. выходит за пределы массива.
   * введенные данные пользователем нелогичны (отрицательная цена, несуществующий возраст и т.д.);
2. Возможность возврата назад (навигация);
3. Обратная связь с пользователем, вывод сообщения об успешности удаления/добавления/редактирования записи.

## 1.3 Требования к программной реализации

1. Все переменные и константы должны иметь осмысленные имена в рамках тематики варианта к курсовой работе. Переменным рекомендуется присваивать имена, состоящие из букв нижнего регистра; для формирования составного имени используется нижнее подчеркивание (например, number\_of\_students) или “верблюжья аннотация”. Константам рекомендуется присваивать имена, состоящие из букв верхнего регистра (например, SIZE\_ARR\_OF\_ACCOUNTS).
2. Имена функций должны быть осмысленными и строится по принципу «глагол + существительное». Если функция выполняет какую-либо проверку и возвращает результат типа bool, то ее название должно начинаться с глагола is (например, isFileExist, isUnicLogin).
3. Не допускается использование оператора прерывания goto.
4. Код не должен содержать неименованных числовых констант (так называемых «магических» чисел), неименованных строковых констант (например, имен файлов и др.). Подобного рода информацию следует выносить в глобальные переменные с атрибутом const. По правилам хорошего стиля программирования тексты всех информационных сообщений, выводимых пользователю в ответ на его действия, также оформляются как константы.
5. Код необходимо комментировать (как минимум в части нетривиальной логики).
6. Код не должен дублироваться – для этого существуют методы и функции.
7. Одна функция решает только одну задачу (например, не допускается в одной функции считывать данные из файла и выводить их на консоль – это две разные функции). При этом внутри функции возможен вызов других функций.
8. Выполнение операций чтения/записи/ в файл должно быть сведено к минимуму (т.е. после однократной выгрузки данных из файла в массив дальнейшая работа ведется с этим массивом, а не происходит многократное считывание данных из файла в каждой функции).
9. Следует избегать длинных функций и глубокой вложенности: текст функции должен умещаться на один экран, а вложенность блоков и операторов должна быть не более трёх.
10. Следует избегать длинных функций: текст функции должен умещаться на один экран (размер текста не должен превышать 25-50 строк).
11. Минимизировать область видимости переменной, сделав ее как можно более локальной.
12. Следует выносить код логически независимых модулей в отдельные .cpp файлы и подключать их с помощью заголовочных .h файлов.

# Конструирование программы

Реализация программы будет осуществляться на языке С++ в IDE-среде Microsoft Visual Studio 2022. Программа будет компилироваться и использоваться в операционных системах семейства Microsoft Windows версии 7 и выше.

## 2.1 Разработка структуры программы

Согласно требованиям к программе, необходимо наличие исполняемой программы, которая работает с файлом пользователей workingWithUsers и файлом продуктов workWithData. Следовательно, с точки зрения верхней архитектуры программы, можно выделить два основных модуля: модуль работы с массивом продуктов arrayOfProduct и модуль работы с массивом учетных записей arrayOfUsers.

В приложении А показана модульная структура программы.

Функция авторизации пользователя заключается в проверке существования в файле учетных записей введённого логина и соответствующего ему пароля, а также наличие доступа для данной учетной записи. Также авторизация пользователя подразумевает получение его роли из файла и предоставление ему соответствующего функционала (функционала пользователя, или функционала администратора, или функционала главного администратора).

После успешной авторизации пользователя создаётся пользовательская сессия с соответствующими привилегиями, согласно роли пользователя. Сессия администратора имеет доступ к модулю управления файлом аккаунтов, и файлов продукта, а сессия пользователя, в свою очередь, имеет доступ только к модулю управления массивом продукта на складе (только обработка информации).

## 2.2 Выбор способа организации данных

Для представления в программе объекта аккаунтов вводится структура account, содержащая nickname, saltedHashPassword, salt, access и role аккаунта.

* saltedHashPassword - осуществляет преобразование массива входных данных произвольной длины в выходную битовую строку установленной длины, выполненную определённым алгоритмом.
* salt - строка данных (16 символов), которая передаётся хеш-функции вместе с входным массивом данных для вычисления хэша.
* nickname - персональное имя для входа в программу
* access - доступ аккаунта, если “0” – у данного аккаунта доступа нет, он не может войти в программу, если “1” – у данного аккаунта есть доступ и при правильной авторизации (ввода логина и пароля) он может войти в систему.
* role - роль аккаунта. В зависимости от роли выбирается определенный функционал для данного пользователя: 1) “0” - имеет доступ только к модулю управления массивом продуктов (только обработка информации); 2) “1” - администратор имеет доступ к модулю управления файлом учетных записей arrayOfUsers и к обработке информации из массива продуктов arrayOfProduct.

Для представления в программе данных о продукте вводится структура ProductInfo, содержащая dayWhenProductCreate, workShopNumber, productName, numberOfProductsProductd, responsiblePerson и подструктуру Date, содержащую day, month, year:

* dayWhenProductCreate – день, месяц и год создания продукта.
* workshopNumber – номер цеха.
* productName – название продукта.
* numberOfProductsProduced – количество продукта
* responsiblePerson – имя ответсвенного по цеху в этот день
* подструктура Date – дата создание продукта: day – день создания, month – месяц создания, year – год создания.

Программа обслуживает данные, хранимые в двух файлах: файле учетных записей account и файле продукта productFile. На рисунке 1 показано хранение учетных записей в файле account. На рисунке 2 показано хранение данных продуктов в файле productFile.

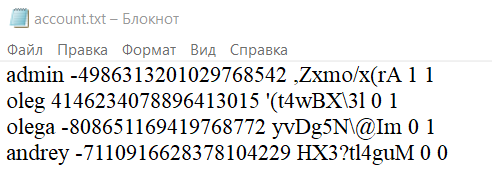


Рисунок 1 –Хранение учетных записей в файле workWithUsers

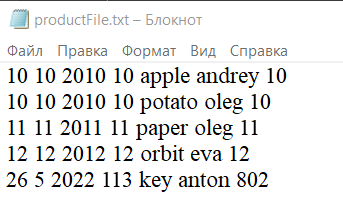


Рисунок 2 – Хранение данных продуктов в файле workWithData

В качестве способа объединения входных данных используются динамически создаваемые массивы:

* для структуры Users используется массив arrayOfUsers, а количество учетных записей в массиве обозначается переменной sizeArrayOfUsers;
* для структуры Product используется массив arrayOfProduct, а количество продукта в массиве обозначается переменной sizeArrayOfData;

Для хранения текстовых данных выбрана кодировка ANSI.

## 2.3 Разработка перечня пользовательских функций программы

**Выход из программы –** функция выхода из программы, при выборе этой функции.

**Регистрация –** функция регистрации, пользователь вводит желаемый логин и пароль и при корректно введенных данных, по умолчанию присваивается введенный логин, хешируется пароль, предоставляется роль пользователя без доступа. Далее происходит ожидание подтверждения заявки от администратора. Заявка вноситься в массив arrayOfUsers и храниться в файле с учетными записями, до тех пор, пока ее не удалит главный администратор. В случае подтверждения администратором заявки или редактирования учетной записи главным администратором, пользователь получает доступ для пользования программой в функционале пользователя.

**Авторизация –** функция авторизации заключается в проверке существования в файле аккаунтов введённого логина и соответствующего ему пароля (хеша). Также авторизация подразумевает получение роли из файла и при наличии доступа (access = “1”) у данной учетной записи предоставляется ему соответствующий функционал (функционал пользователя, функционал администратора). В случае некорректно введённых данных, либо же данных не имеющих доступа, вход в программу не возможен, пользователю предоставляется попытка повторного ввода либо же возврат в главное меню.

**Модуль пользователя** (предоставляется меню возможного функционала для пользователя)**:**

* **Просмотр информации об продуктах –** пользователь может посмотреть данные об всех продуктах.
* **Поиск -** предоставляется меню поиска (**поиск по дню когда продукт был произведен, поиск по номеру цеха, поиск по имени продукта, поиск по количеству выпущенного продукта, поиск по ответвенного в день производства продукта),** выбирается вид поиска, вводятся данные, которые необходимо найти, в случае успешного поиска выводиться найденная информация, в случае некорректно введённых данных либо же если данные не найдены, то предлагается выбор на повторный поиск или возврат в главное меню пользователя.
* **Сортировка –** предоставляется меню сортировки (**сортировка по дню когда продукт был произведен, сортировка по номеру цеха, сортировка количеству выпущенного продукта),** выбирается по какому признаку сортировать данные, затем предоставляется выбор вида сортировки от А до Я, либо от Я до А, либо же вернуться назад в главное меню сортировки, в зависимости от выбора выводиться соответствующая информация.
* **Выход из функционала пользователя –** при подтверждении действия происходит выход из аккаунта пользователя и возврат в главное меню.

**Модуль администратора** (предоставляется меню возможного функционала для администратора)**:**

* **Работа с данными продукта** (предоставляется меню возможного функционала для работы с данными продуктов):

1. **Просмотр –** администратор может просмотреть данные об всей продукции.
2. **Добавление –** администратор может добавлять новые продукты вручную. Вводятся все необходимые данные и проверяются на корректный ввод. В случае корректно введенных данных, поступает запрос на подтверждение действия. В случае подтверждения действия, размер массива arrayOfProdut увеличивается, данные присваиваются в структуру и записываются в файл и массив productFile, в случае отказа на добавление размер массива остается прежним и данные не добавляются в массив.
3. **Удаление –** администратор может удалять существующих продукты. Выводятся все продукты из массива arrayOfProduct, он выбирает какого именно продукт удалить. Поступает запрос на подтверждения действия, размер массива уменьшается, и данный продукт удаляется из массива и файла productFile, в случае отказа на удаление размер массива остается прежним, и выбранный пациент не удаляется.
4. **Выход из работы с данными продуктов –** происходит выход из работы с данными продуктов и перенаправление в главное меню администратора.

* **Работа с учетными записями** (предоставляется меню возможного функционала для работы с учетными записями):

1. **Просмотр всех учетных записей –** выводятся данные (логин, доступ, роль) всех учетных записей в массиве accounts.
2. **Подтверждение –** выводятся все учетные записи без доступа (access = “0”). Администратор выбирает какую заявку одобрить, предоставляется запрос на подтверждения действия. В случае подтверждения данной учетной записи присваивается доступ (access = “1”) и изменения перезаписываются в массив и файл accounts и пользователю данной учетной записи предоставляется возможность входа в программу, в случае отказа выбранная учетная запись не активируется.
3. **Выход из работы с учетными записями –** происходит выход из работы с учетными записями и перенаправление в главное меню администратора

**– Обработка информации о продуктах** (предоставляется меню возможного функционала для обработки информации о продукта)**:**

* **Просмотр информации об продуктах –** пользователь может посмотреть данные об всех продуктах.
* **Поиск -** предоставляется меню поиска (**поиск по дню когда продукт был произведен, поиск по номеру цеха, поиск по имени продукта, поиск по количеству выпущенного продукта, поиск по ответвенного в день производства продукта),** выбирается вид поиска, вводятся данные, которые необходимо найти, в случае успешного поиска выводиться найденная информация, в случае некорректно введённых данных либо же если данные не найдены, то предлагается выбор на повторный поиск или возврат в главное меню пользователя.
* **Сортировка –** предоставляется меню сортировки (**сортировка по дню когда продукт был произведен, сортировка по номеру цеха, сортировка количеству выпущенного продукта),** выбирается по какому признаку сортировать данные, затем предоставляется выбор вида сортировки от А до Я, либо от Я до А, либо же вернуться назад в главное меню сортировки, в зависимости от выбора выводиться соответствующая информация.

**– Выход из функционала администратора –** при подтверждении действия происходит выход из аккаунта и возврат в главное меню.

# Разработка алгоритмов работы программы

## 3.1 Алгоритм функции main

Функция main является точкой входа в программу, вызывает функцию авторизаии. В приложении Б показан алгоритм данной функции.

## Алгоритм функции surnamePatientsEdit

Функция surnamePatientsEdit является функцией редактирования фамилии пациента. В приложении В показан алгоритм данной функции.

## 3.3 Алгоритм функции deletePatient

Функция deletePatient является функцией удаления пациента из массива и файла. Алгоритм данной функции показан в приложении Г.

# описание работы программы

## 4.1 Вход в программу

При запуске программа пытается найти и открыть файлы с учетными записями и данными о продукте. Если файл найден, программа начинает выводить Главное меню.

* **Главное меню входа** - после успешного входа в программу, программа запускает диалоговое взаимодействие с пользователем путём отображения пронумерованных пунктов меню и запросом у пользователя ввода желаемого номера (рис.8).

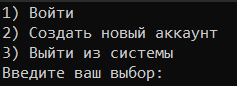


Рисунок 8 – Главное меню входа

* **Войти** - пользователю предоставляется возможность ввести логин и пароль для входа в программу. При корректном вводе и при правильном вводе пароля и логина, пользователь заходит в свой аккаунт, с функционалом соответствующей роли (рис.9).



Рисунок 9 – Авторизация

* **Регистрация -** пользователю предоставляется возможность зарегистрироваться в программу. Он вводит логин, пароль и повторяет пароль для подтверждения. При корректном вводе данных и при вводе уникального логина, пользователь записывается в файл учетных записей и автоматически подается заявка на активацию аккаунта (рис.10).

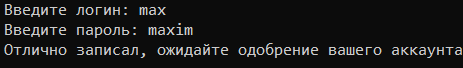


Рисунок 10 – Регистрация

* **Выход из программы -** при выборе выхода из программы поступает запрос на подтверждения действия и осуществляется выход из программы (рис.11).



Рисунок 11 – Выход из программы

## 4.2 Модуль администратора

После успешного входа в функционал администратора программа запускает диалоговое взаимодействие с администратором путём отображения пронумерованных пунктов меню и запросом у администратора ввода желаемого номера (рис.12).

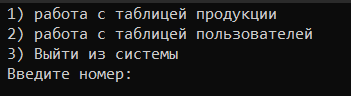


Рисунок 12 – Главное меню администратора

* **Работа с данными продукта -** программа запускает диалоговое взаимодействие с администратором путём отображения пронумерованных пунктов меню «Работы с данными продукта» и запросом у администратора ввода желаемого номера (рис.13).

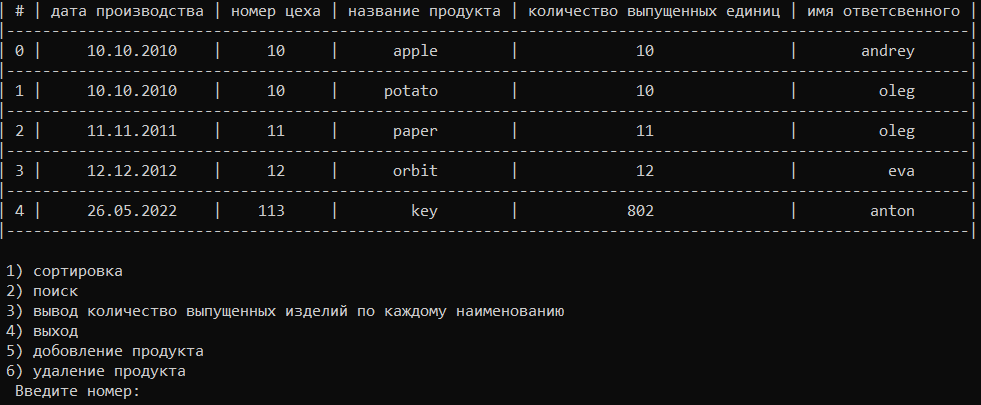


Рисунок 13 – Меню «Работы с данными пациентов» для администратора

1. **Просмотр всей продукции** – выводится информация о всех пациентах (рис.14).

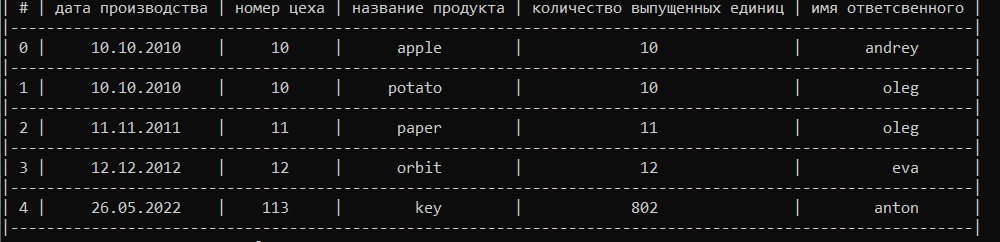


Рисунок 14 – Вывод всей продукции

1. **Добавить продукт –** нужно заполнить корректно все данные, а затем поступает запрос на подтверждение действия (рис.15).

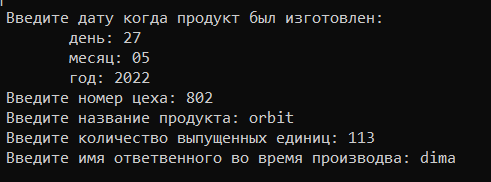
****

Рисунок 15 – Добавление пациента

1. **Удалить продукт -** выводится информация о всех продуктах, и администратор выбирает номер пациента, которого следует удалить (рис.16).

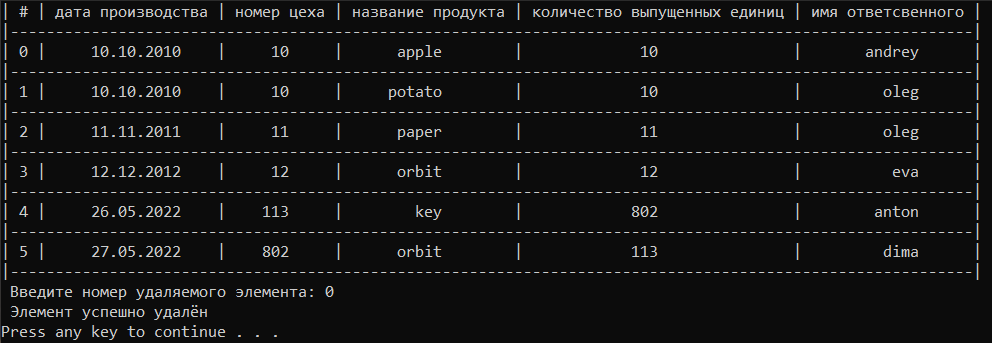


Рисунок 16 – Удаление продукта

Затем поступает запрос на подтверждения удаления данного пациента (рис.53).

====================================================

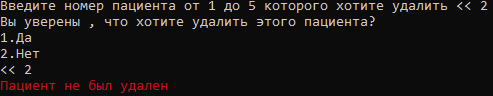


Рисунок 53 – Подтверждение удаления данного пациента

1. **Редактирование пациентов -** выводится информация о всех пациентах, и администратор выбирает номер пациента, которого хочет отредактировать (рис.54).

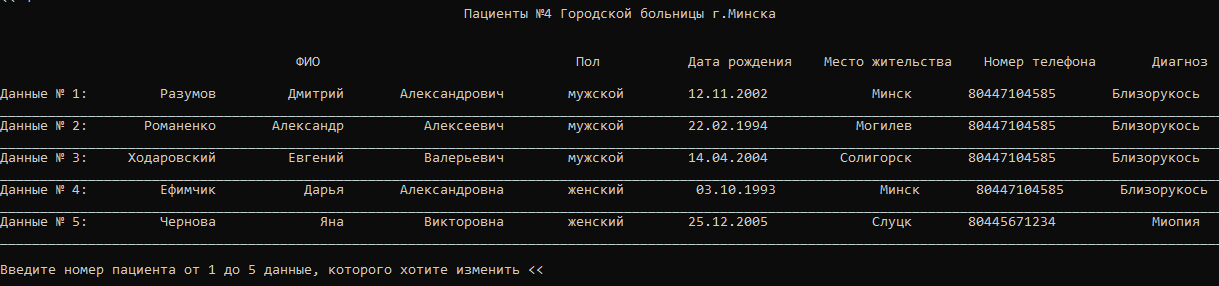


Рисунок 54 - Информация о всех пациентах для редактирования

Затем выводятся данные пациента которого будем редактировать, меню редактирования для пациента и производиться ввод критерия редактирования (рис.55). Далее показаны примеры редактирования (рис.56 – рис.63).

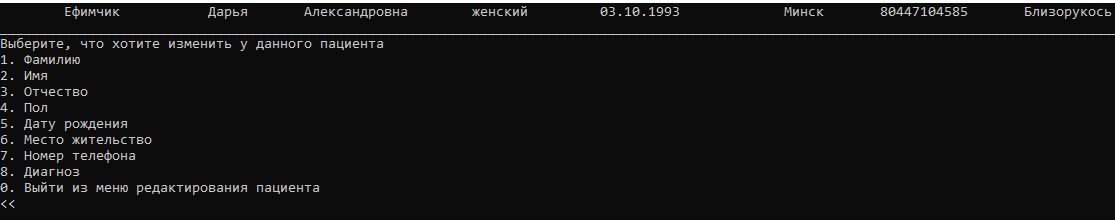


Рисунок 55 – Меню редактирования данных для данного пациента

На рисунке 56 показан пример редактирования фамилии у пациента. Программа просит ввести новые данные и подтвердить изменения.

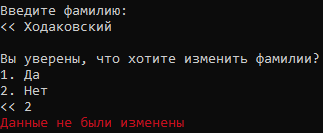


Рисунок 56 – Редактирование фамилии пациента

На рисунке 57 показан пример редактирования имени у пациента. Программа просит ввести новые данные и подтвердить изменения.

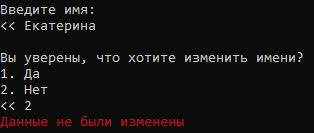


Рисунок 57 – Редактирование имени пациента

На рисунке 58 показан пример редактирования отчества у пациента. Программа просит ввести новые данные и подтвердить изменения.

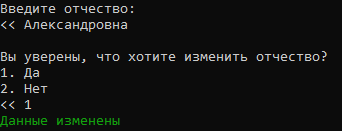


Рисунок 58 – Редактирование отчества пациента

На рисунке 59 показан пример изменения гендерного пола у пациента. Программа просит подтвердить изменения.

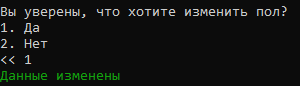


Рисунок 59 – Изменение гендерного пола у пациента

На рисунке 60 показан пример редактирования даты рождения у пациента. Программа просит ввести новые данные и подтвердить изменения.

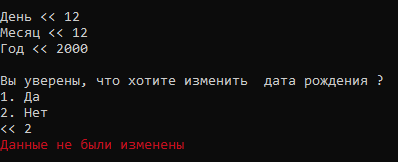


Рисунок 60 – Редактирование даты рождения пациента

На рисунке 61 показан пример редактирования места жительства пациента. Программа просит ввести новые данные и подтвердить изменения.

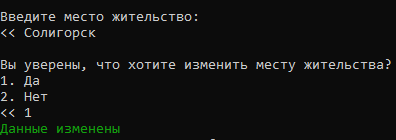


Рисунок 61 – Редактирование места жительства пациента

На рисунке 62 показан пример редактирования номера телефона пациента. Программа просит ввести новые данные и подтвердить изменения.

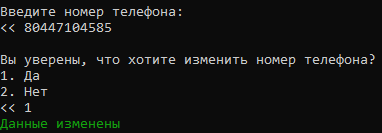


Рисунок 62 – Редактирование номера телефона пациента

На рисунке 63 показан пример редактирования диагноза пациента. Программа просит ввести новые данные и подтвердить изменения.

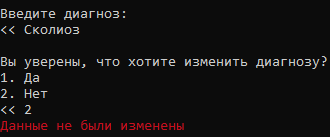


Рисунок 63 – Редактирование диагноза пациента

* **Работа с учетными записями -** программа запускает диалоговое взаимодействие с администратором путём отображения пронумерованных пунктов меню «Работы с учетными записями» и запросом у администратора ввода желаемого номера (рис.64).

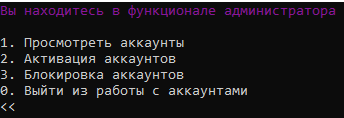
****

Рисунок 64 – Меню «Работы с учетными записями» для администратора

1. **Просмотр аккаунтов -** выводится информация о всех учетных записях (логин, доступ, роль) (рис.65).

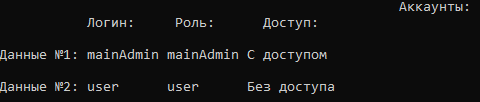


Рисунок 65 – Просмотр информации о всех учетных записях

1. **Активация аккаунта –** выводятся все аккаунты без доступа входа в программу, и администратор выбирает какую заявку одобрить (дать доступ учетной записи), затем поступает запрос на подтверждения действия для активации данного аккаунта (рис.66).

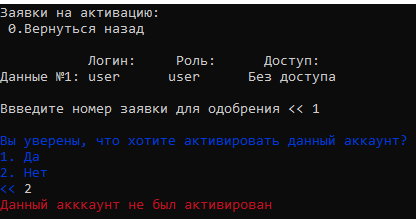


Рисунок 66 – Активация аккаунта

1. **Блокировка аккаунта -** выводятся все аккаунты с доступом для входа в программу, и администратор выбирает какой аккаунт заблокировать (без доступа), затем поступает запрос на подтверждения действия для блокировки данного аккаунта (рис.67).

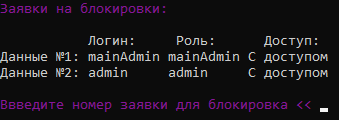


Рисунок 67 – Блокировка аккаунта

* **Обработка данных пациентов –** снимки экрана обработки данных о пациентах соответствуют снимкам в модуле главного администратора (рис.22 – рис.45).
* **Изменение своего пароля -** снимки экрана изменение своего пароля соответствуют снимкам в модуле главного администратора (рис.46).
* **Выход из функционала администратора -** снимки экрана выхода из функционала администратора соответствуют снимкам в модуле главного администратора (рис.47).

## 4.3 Модуль пользователя

После успешного входа в функционал пользователя программа запускает диалоговое взаимодействие с пользователем путём отображения пронумерованных пунктов меню и запросом у пользователя ввода желаемого номера (рис. 68).

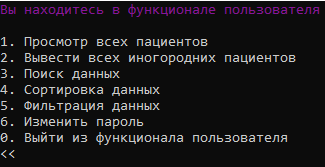
****

Рисунок 68 – Главное меню пользователя

* **Просмотр всех пациентов –** снимки экрана просмотра всех пациентов совпадают со снимками экрана в модуле главного администратора (рис.23).
* **Просмотр всех иногородних пациентах –** снимки экрана просмотра всех иногородних пациентов совпадают со снимками экрана в модуле главного администратора (рис.24).
* **Поиск данных –** снимки экрана поиска данных совпадают со снимками экрана в модуле главного администратора (рис.25 – рис.29).
* **Сортировка данных –** снимки экрана сортировки данных совпадают со снимками экрана в модуле главного администратора (рис.30 – рис.40).
* **Фильтрация данных** – снимки экрана фильтрации данных совпадают со снимками экрана в модуле главного администратора (рис.41 – рис.45).
* **Изменение пароля –** снимки экрана изменения своего пароля совпадают со снимками экрана в модуле главного администратора (рис.46).
* **Выход из функционала пользователя –** снимки экрана выхода из функционала пользователя совпадают со снимками экрана в модуле главного администратора (рис.47).

## 4.4 Исключительные ситуации

Обработка исключительных ситуаций, как правило, занимает существенную часть кода программы. Это неизбежно, потому что программа без обработки ошибок и пограничных ситуаций выглядит для пользователя некачественной и может приводить к множеству негативных последствий: повреждению данных, неверной трактовке результатов работы программы, нагрузкой на службу поддержки или разработчика и т.п. В связи с этим, в программу встроено достаточно много проверок данных и результатов выполнения функций, обработок программных исключений, сопровождающихся уведомлением пользователя о проблеме и, при возможности, логичной реакцией на ситуацию либо прерыванием работы программы, если программа не смогла обработать исключительную ситуацию.

На рисунке 69 показана первая, простейшая исключительная ситуация: введён неверный логин пользователя, либо пароль, либо у данного аккаунта нет доступа для входа в программу. Программа сообщает об этом пользователю и предлагает повторить попытку либо вернуться в главное меню авторизации.

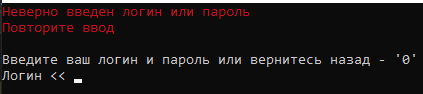


Рисунок 69 – Неверный ввод логина или пароля при входе

Во всех меню при вводе несуществующего пункта меню программа выведет соответствующее сообщение и попросит повторить ввод либо вернуться назад. На рисунке 70 показан пример отображения ошибки при вводе несуществующего пункта меню.

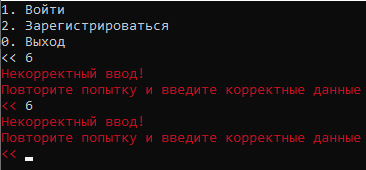


Рисунок 70 – Неверный ввод пункта меню

При вводе пользователем несуществующего номера учетной записи или номера пациента для таких операций, как редактирование, удаление, блокировка, активация и т.д., программа выведет соответствующее сообщение. На рисунке 71 показан пример отображения ошибки при вводе несуществующего номера пациента.

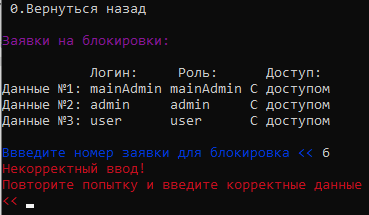


Рисунок 71 – Ошибка при вводе несуществующего номера пациента или аккаунта

На рисунке 72 показан пример вывода сообщения при попытке создания новой учетной записи (регистрации) с логином, который уже существует (занят), программа выведет соответствующее сообщение об ошибке.

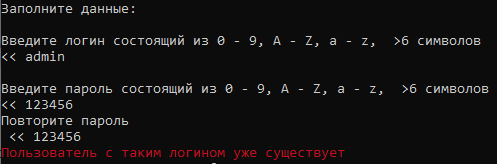


Рисунок 72 – Занятый логин

На рисунке 73 показан пример вывода сообщения при попытке изменения логина у учетной записи на такое, которое уже существует.

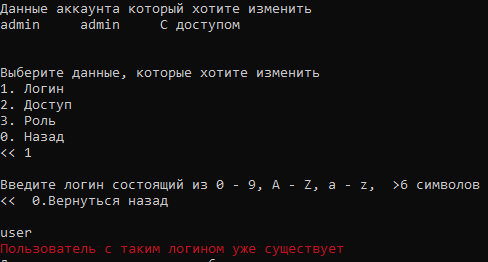


Рисунок 73 – Редактирование логина, на который уже занят

На рисунке 74 показан пример вывода сообщения при попытке создания учетной записи (регистрации) и вводе пароля и логина с недопустимыми символами или недопустимой длины, программа выведет соответствующее сообщение об ошибке.

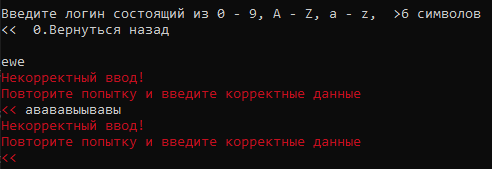


Рисунок 74 – Ввод данных несоответствующий формату

На рисунке 75 показан пример вывода сообщения, если при поиске данные не были найдены. Предлагается выбор вернуться назад или повторить попытку и ввести другую искомую строку.

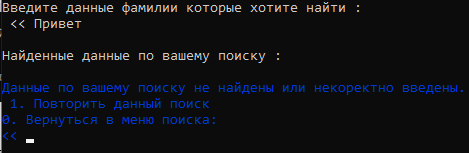


Рисунок 75 – Данные не найдены при поиске

На рисунке 76 показан пример вывода сообщения, если ввод искомых данных не совпадает с допустимым форматом ввода.

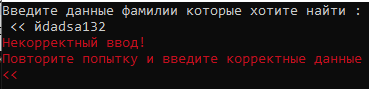


Рисунок 76 – Недопустимый формат ввода в искомую строку

На рисунке 77 показан пример вывода сообщения при вводе данных для фильтрации по возрасту и диагнозу, таких пациентов нет.

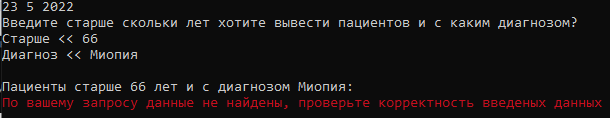


Рисунок 77 – При фильтрации пациенты не найдены

На рисунке 78 показан пример вывода сообщения при попытке регистрации, введенные данные, не совпадают с форматом вводимой строки либо не совпадают при повторном вводе.

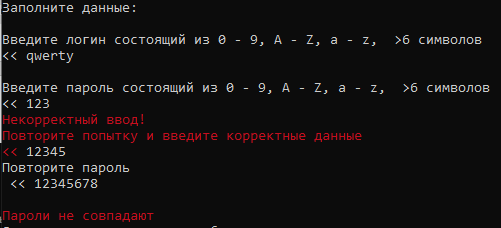


Рисунок 78 – Неверный пароль при регистрации

На рисунке 79-80 показаны примеры вывода сообщений, если пациенты отсутствуют в массиве пациентов, программа выводит советующее сообщение, для функций с пациентами кроме добавления пациента.

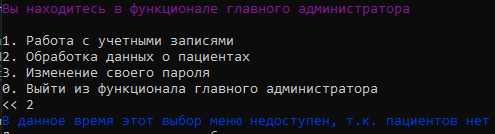


Рисунок 79 – Пациентов нет в массиве

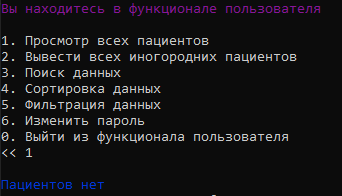


Рисунок 80 – Пациентов нет в массиве

На рисунке 81 показан пример вывода сообщения, если нет заявок для регистрации.

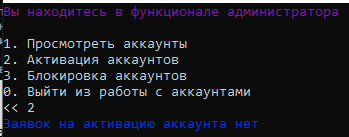


Рисунок 81 – Заявок на активацию нет

На рисунке 82 показан вывод сообщения при добавлении или редактировании даты рождения у пациента, для того чтобы исключить некорректный ввод даты рождения (к примеру 30 февраля, -4 мая и т.д.), программы выводит соответствующее сообщение.

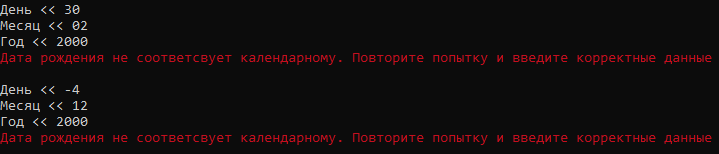


Рисунок 82 – Дата рождения не соответствует календарному типу

На рисунке 83 показан пример вывода сообщения при добавлении или редактировании номера телефона пациента, если номер содержит какие-либо символа кроме цифр или неправильную длину строки (не 11 или не 13 символов) программа выводит соответствующее сообщение.

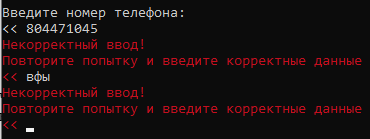


Рисунок 83 – Некорректный ввод номер телефона

На рисунке 84 показан пример вывода сообщения при попытке заблокировать или редактировать свой аккаунт, в котором мы находимся.

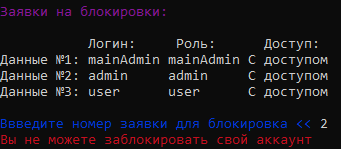


Рисунок 84 – Попытка заблокировать свой аккаунт

На рисунке 85-86 показан пример вывода сообщения при попытке заблокировать или отредактировать аккаунт главного администратора.

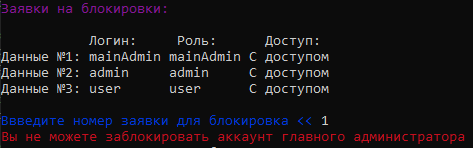


Рисунок 85 – Попытка заблокировать аккаунт главного администратора

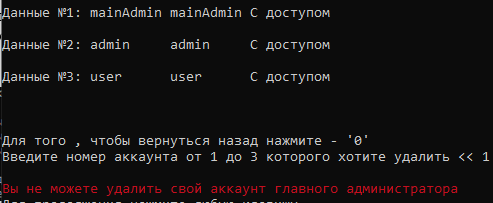


Рисунок 86 – Попытка удалить аккаунт главного администратора

На рисунке 87 показан пример вывода сообщения, если нет иногородних пациентов.

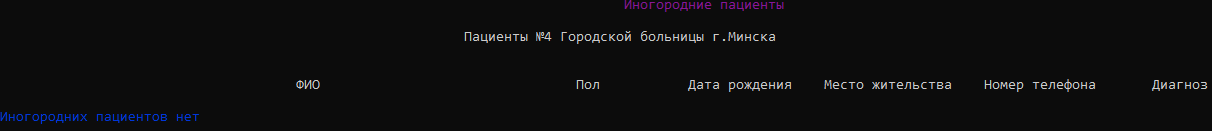


Рисунок 87 – Иногородних пациентов нет

На рисунке 88 показан пример вывода сообщения, если нет пациентов выбранного пола при фильтрации.

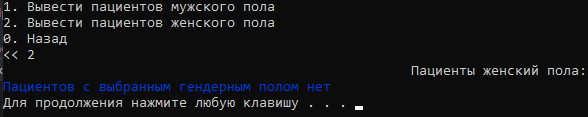


Рисунок 88 – Нет пациентов с выбранными гендерным полом

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Модульная структура программы (к пункту 2.1)**



Рисунок 3 – Модульная структура программы

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Алгоритм функции main (к пункту 3.1)**



Рисунок 4 – Блок-схема алгоритма функции main

**ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Алгоритм функции surnamePatintsEdit (к пункту 3.2)**



Рисунок 5 – Блок-схема алгоритма функции surnamePatientsEdit

**ПРИЛОЖЕНИЕ Г**

**(обязательное)**

**Алгоритм функции deletePatient (к пункту 3.3)**



Рисунок 6 – Блок-схема алгоритма функции deletePatient

**ПРИЛОЖЕНИЕ Д**

**(обязательное)**

**Листинг кода с комментариями**

**Файл Main.cpp**

//Подключаем заголовочные файлы старт

#include"Functions.h"

#include "Autorization.h"

int main() {

//Руссификация консоли

SetConsoleCP(1251); //Установка кодовой страницы win-cp 1251 и поток ввода

SetConsoleOutputCP(1251); //Установка кодовой страницы win-cp 1251 и поток ввода

//Считывание файлов пациентов и учетных записей в массив

int amountOfAccounts = DATA\_EMPTY; //Количество учетных записей в файле

string fileAccounts = "accounts.txt"; //Название для файла с учетными записями

account\* accounts = new account[RESERVE\_AMOUNT\_OF\_ACCOUNTS]; //Создание структуры для записи учетных записей из файла

openingFileAccounts(accounts, amountOfAccounts, fileAccounts); //Открытие файла учетных записей

int amountOfPatients = DATA\_EMPTY; //Количество пациентов в файле

string filePatients = "patients.txt"; //Название файла для пациентов

dataPatients\* patients = new dataPatients[RESERVE\_AMOUNT\_OF\_ACCOUNTS]; //Создание структуры для записи пациентов из файла

openingFilePatients(patients, amountOfPatients, filePatients); //Открытие файла с пациентами

autorization(accounts, amountOfAccounts, patients, amountOfPatients); //Функция авторизации

return 0;

}

**Основыне функции из файла Admin.cpp**

//Работа с учетными записями

void workWithAccount(account\*(&accounts), int& amountOfAccounts, int activeAccount) {

menuWorkWithAccount(); //Меню работы с учетными записями

int choiceWorkWithAccounts = inputNumber(EXIT\_FROM\_WORK\_WITH\_ACCOUNTS, BLOCKING\_ACCOUNTS); //Выбор действия и проверка на численный ввод

switch (choiceWorkWithAccounts) { //Передаем выбор действия

case VIEW\_ACCOUNTS: { //Просмотр всех учетных записей

printAllAccounts(accounts, amountOfAccounts);

break;

}

case ACTIVATION\_ACCOUNTS: { //Активация неодобренных заявок

activationAccount(accounts, amountOfAccounts, activeAccount);

break;

}

case BLOCKING\_ACCOUNTS: { //Блокировка аккаунтов

blockingAccount(accounts, amountOfAccounts, activeAccount);

break;

}

case EXIT\_FROM\_WORK\_WITH\_ACCOUNTS: { //Выход из работы с учетными записями

return;

}

}

system("pause");

workWithAccount(accounts, amountOfAccounts, activeAccount);

}

//Работа с данными пациентов

void workWithPatients(dataPatients\* (&patients), int& amountOfPatients) {

menuWorkWithPatients(); //Вызов меню работы с базой данных

int choiceWorkWithPatients = inputNumber(EXIT\_FROM\_WORKING\_WITH\_PATIENTS, EDIT\_PATIENTS); //Выбор действия и проверка на численный ввод

switch (choiceWorkWithPatients) { //Передаем выбор действия

case VIEW\_PATIENTS: { //Просмотр всех пациентов

if (amountOfPatients != DATA\_EMPTY) { //Если есть пациенты в массиве

printAllPatients(patients, amountOfPatients);

}

else {

coutBlueText(NO\_PATIENTS);

}

break;

}

case ADD\_PATIENTS: { //Добавление пациента

addPatient(patients, amountOfPatients);

break;

}

case DELETE\_PATIENTS: { //Удаление пациента

if (amountOfPatients != DATA\_EMPTY) {//Если есть пациенты в массиве

deletePatient(patients, amountOfPatients);

}

else {

coutBlueText(NO\_PATIENTS);

}

break;

}

case EDIT\_PATIENTS: { //Редактирование пациентов

if (amountOfPatients != DATA\_EMPTY) {//Если есть пациенты в массиве

editDataPatient(patients, amountOfPatients);

}

else {

coutBlueText(NO\_PATIENTS);

}

break;

}

case EXIT\_FROM\_WORKING\_WITH\_PATIENTS: { //Выход из работы с данными пациентами

return;

}

}

system("pause");

workWithPatients(patients, amountOfPatients);

}

//Обработка данных о пациентах

void proccesingPatientsData(dataPatients\* (&patients), int& amountOfPatients) {

menuProccesingPatientsData(); //Вызов меню режима обработки данных о пациентах

int choiceProccessingPatientsData = inputNumber(EXIT\_FROM\_WORKING\_WITH\_PATIENTS, FILTRATION\_PATIENTS); //Выбор действия и проверка на численный ввод

switch (choiceProccessingPatientsData) { //Передаем выбор действия от пользователя

case VIEW\_PATIENTS\_ALL: { //Просмотр всех пациентов

printAllPatients(patients, amountOfPatients);

break;

}

case VIEW\_NONRESIDENT\_PATIENTS: { //Иногородние пациенты

printNonresidentPatients(patients, amountOfPatients);

break;

}

case SEARCH\_PATIENTS: { //Поиск

search(patients, amountOfPatients);

break;

}

case SORT\_PATIENTS: { //Сортировка

sort(patients, amountOfPatients);

break;

}

case FILTRATION\_PATIENTS: { //Фильтрация

filtration(patients, amountOfPatients);

break;

}

case EXIT\_FROM\_WORKING\_WITH\_PATIENTS: {

return;

}

}

system("pause");

proccesingPatientsData(patients, amountOfPatients);

}

//Функционал администратора

void admin(account\* (&accounts), int& amountOfAccounts, int activeAccount, dataPatients\* (&patients), int& amountOfPatients) {

menuAdmin(); //Вызов главного меню админа

int choiceAdmin = inputNumber(EXIT\_FROM\_ADMIN, EDIT\_MY\_PASSWORD); //Выбор действия админа и проверка на численный ввод

system("cls");

switch (choiceAdmin) { //Передаем выбор действия от пользователя

case WORK\_WITH\_PATIENTS: { //Работа с данными пациентов

workWithPatients(patients, amountOfPatients);

break;

}

case WORK\_WITH\_ACCOUNTS: { //Работа с учетными записями

workWithAccount(accounts, amountOfAccounts, activeAccount);

break;

}

case DATA\_PROCCESING\_FOR\_PATIENTS: { //Обработка данных о пациентах

if (amountOfPatients != DATA\_EMPTY) { //Если есь пациенты в массиве

proccesingPatientsData(patients, amountOfPatients);

}

else {

coutBlueText(NO\_PATIENTS);

system("pause");

}

break;

}

case EDIT\_MY\_PASSWORD: { //Изменение пароля админа

passwordEdit(accounts, amountOfAccounts, activeAccount);

break;

}

case EXIT\_FROM\_ADMIN: { //Выход из аккаунта админа

exitFromAccount(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients, admin);

return;

}

}

admin(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients);

}

**Основные функции из файла Autorization.cpp**

//Функция авторизации

void autorization(account\*(&accounts), int &amountOfAccounts, dataPatients\*(&patients), int &amountOfPatients) {

mainMenu(); //Главное меню авторизации

int choiceMain = inputNumber(EXIT, REGISTR); //Выбор действия авторизации и проверка на корректный ввод

system("cls");

switch (choiceMain) { //Передаем выбор из главного меню

case ENTRANCE: { //Вход

entrance(accounts, amountOfAccounts, patients, amountOfPatients);

break;

}

case REGISTR: { //Регистрация

registration(accounts, amountOfAccounts, patients, amountOfPatients);

break;

}

case EXIT: { //Выход из программ

exitFromProgram(accounts, amountOfAccounts, patients, amountOfPatients);

break;

}

}

}

//Сверяем введенные логин и пароль

int checkLoginAndPasswordForEntrance(account\* (&accounts), int &amountOfAccounts, string hash, string validationLogin, dataPatients\* (&patients), int &amountOfPatients) {

bool input = false; //Для проверки на корректный ввод для входа в систему

for (int numberOfAccount = 0; numberOfAccount < amountOfAccounts; numberOfAccount++) {

if (validationLogin == accounts[numberOfAccount].login && accounts[numberOfAccount].hash == hash && accounts[numberOfAccount].access == ACCESS) { //Сверяем введеные данные с существующими

if (accounts[numberOfAccount].role == USER) { //Если данные пользователя

user(accounts, amountOfAccounts, numberOfAccount, patients, amountOfPatients);

input = true; //Вошли

break;

}

if (accounts[numberOfAccount].role == ADMIN) { //Если данные администратора

input = true; //Вошли

admin(accounts,amountOfAccounts, numberOfAccount, patients, amountOfPatients); //Вызываем функционал администартор

break;

}

if (accounts[numberOfAccount].role == ADMIN\_MAIN) { //Если данные главного администратора

input = true; //Вошли

mainAdmin(accounts, amountOfAccounts, numberOfAccount, patients, amountOfPatients); //Вызываем функционал главного админа

break;

}

int activeAccount = numberOfAccount; //Номер аккаунта под которым зашли

return activeAccount; //Возвращаем номер активного аккаунта

}

}

if (input == false) { //Если вход не выполнен

system("cls");

coutRedText(INCORRECT\_LOGIN\_AND\_PASSWORD\_INPUT);

entrance(accounts, amountOfAccounts, patients, amountOfPatients);

}

}

//Функция входа в систему

void entrance(account\* (&accounts), int &amountOfAccounts,dataPatients\*(&patients), int &amountOfPatients) {

cout << LOGIN\_AND\_PASSWORD\_INPUT;

cout << LOGIN\_INPUT;

string validationLogin; cin >> validationLogin; //Вводим логин

if (validationLogin == BACK\_TO\_AUTORIZATION) { //Если вернулись назад

autorization(accounts, amountOfAccounts, patients, amountOfPatients); //Меню авторизации;

}

else {

cout << PASSWORD\_INPUT;

string validationPassword = inputPassword(); //Вводим пароль и делаем ввод звездочками

string hash = generationHashForPassword(validationPassword); //Хешируем пароль

checkLoginAndPasswordForEntrance(accounts, amountOfAccounts, hash, validationLogin, patients, amountOfPatients); //Функция входа

}

}

//Выход из программы

void exitFromProgram(account\* (&accounts), int &amountOfAccounts, dataPatients\*(&patients), int &amountOfPatients) {

system("cls");

cout << CONFIRMATION\_EXIT\_FROM\_PROGRAM;

if (confirmation()) {

system("cls");

cout << EXIT\_FROM\_PROGRAM;

system("pause");

return;

}

else {

autorization(accounts, amountOfAccounts, patients, amountOfPatients);

}

}

//Регистрация

void registration(account\* (&accounts), int& amountOfAccounts, dataPatients\* (&patients), int& amountOfPatients) {

//Создаем новый временный массив и добавляем количество аккаунтов

account\* buf;

buf = new account[amountOfAccounts];

copyAccount(buf, accounts, amountOfAccounts);

amountOfAccounts++;

accounts = new account[amountOfAccounts];

copyAccount(accounts, buf, --amountOfAccounts);

//Заполняем данные для добавления

cout << INPUT\_DATA\_FOR\_REGIST;

cout << INPUT\_LOGIN; string newLogin = inputValidationPasswordOrLogin();

cout << INPUT\_PASSWORD; string newPassword = inputValidationPasswordOrLogin();

cout << INPUT\_REPEAT\_PASSWORD; string repeatNewPassword = inputValidationPasswordOrLogin();

bool regist = true; //Разрешение для регистрации

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].login == newLogin) { //Если логин уже занят

regist = false; //Занят

coutRedText(LOGIN\_IS\_BUSY);

break;

}

}

//Сверяем на корректный ввод пароля

if (newPassword != repeatNewPassword) {

coutRedText( PASSWORDS\_DO\_MOT\_MATCH);

regist = false;

}

else if (regist == true) { //Если данные введены вверно

cout << MENU\_CONFIRMATION\_REGISTRATION;

int confirmation = inputNumber(YES, NO); //Подтверждение действия и проверка на корректный ввод

if (confirmation == YES) { //Если нет , то не сохраняем

amountOfAccounts++; //увеличение количества пациентов в структуре на 1

accounts[amountOfAccounts - 1].login = newLogin;

accounts[amountOfAccounts - 1].role = USER;

accounts[amountOfAccounts - 1].access = NOACCESS;

accounts[amountOfAccounts - 1].hash = generationHashForPassword(newPassword);

accounts[amountOfAccounts - 1].salt = generateSalt(SALT\_SIZE);

savingAccount(accounts, amountOfAccounts, "accounts.txt"); //Сохранение изменений в файл

coutGreenText(YOU\_ARE\_REGISTERED);

}

}

delete[]buf; //Удаляем временный массив

system("pause");

autorization(accounts,amountOfAccounts, patients,amountOfPatients);

}

**Основные функции из файла EditFunctions.cpp**

//Ввод данных изменения

string inputEditString(string INPUT\_DATA, string editData, string SYMBOLS) {

cout << INPUT\_DATA;

editData = inputValidationString(SYMBOLS);

return editData;

}

//Редактирование пациентов

void editDataPatient(dataPatients\* (&patients), int& amountOfPatients) {

printAllPatients(patients, amountOfPatients); //Вывод всех пациентов

cout << "\nВведите номер пациента от 1 до " << amountOfPatients << " данные, которого хотите изменить << ";

int numberOfPatient = inputNumber(BACK, amountOfPatients); //Проверка на корректный ввод номера пациента

system("cls");

if (numberOfPatient != BACK) { //Если не 0 (выход назад)

string editData;

numberOfPatient--;

printPatient(patients, numberOfPatient); //Вывод одного пациента данные которого хотим изменить

menuEditDataPatient(); //Меню выбора редактирования

int choiceEditDataPatient = inputNumber(EXIT\_MENU\_EDIT\_PATIENT, DIAGNOSIS\_EDIT); //Проверка на корректный ввод

switch (choiceEditDataPatient) { //Передаем выбор вида изменения

case SURNAME\_EDIT: { //Редактирование фамилии

surnamePatientEdit(patients, amountOfPatients, numberOfPatient, editData);

break;

}

case NAME\_EDIT: { //Редактирование имени

namePatientEdit(patients, amountOfPatients, numberOfPatient, editData);

break;

}

case PATRONYMIC\_EDIT: { //Редактирование отчества

patronymicPatientEdit(patients, amountOfPatients, numberOfPatient, editData);

break;

}

case GENDER\_EDIT: { //Изменение пола

genderPatientEdit(patients, amountOfPatients, numberOfPatient);

break;

}

case DATA\_OF\_BIRTH\_EDIT: { //Редактирование даты рождения

dataOfBirthPatientEdit(patients, amountOfPatients, numberOfPatient); //Изменеие даты рождения

break;

}

case TOWN\_EDIT: { //Редактирование места жительства

townPatientEdit(patients, amountOfPatients, numberOfPatient, editData);

break;

}

case PHONE\_NUMBER\_EDIT: { //Редактирование номера телефона

phonenumberPatientEdit(patients, amountOfPatients, numberOfPatient, editData);

break;

}

case DIAGNOSIS\_EDIT: { //Редактирование диагноза

diagnosisPatientEdit(patients, amountOfPatients, numberOfPatient, editData);

break;

}

case EXIT\_MENU\_EDIT\_PATIENT: { //Выход из меню редактирования

break;

}

}

}

else {

return;

}

}

//Функция редактирования фамилии данного пациента

void surnamePatientEdit(dataPatients\* patients, int amountOfPatients, int numberOfPatient, string editData) {

editData = inputEditString(INPUT\_SURNAME, editData, LETTERS\_SYMBOLS); //Вводим новые данные

menuConfirmationEdit(SURNAME); //Меню подтверждения

if (confirmation()) { //Если согласны на изменение данных

patients[numberOfPatient].surname = editData; //Заменяем старые данные на новые

savingPatients(patients, amountOfPatients, "patients.txt"); //Сохраняем изменения

coutGreenText(DATA\_BEEN\_CHANGED);

}

else {

coutRedText(DATA\_HAS\_NOT\_BEEN\_CHANGED);

}

}

//Функция изменения пола данного пациента

void genderPatientEdit(dataPatients\* patients, int amountOfPatients, int numberOfPatient) {

menuConfirmationEdit(GENDER); //Меню подтверждения

if (confirmation()) { //Если согласны на изменение пола

if (patients[numberOfPatient].gender == MAN) { //Если пол был мужской

patients[numberOfPatient].gender = WOMAN; //Заменяем на женский

}

else { //Если пол был женский

patients[numberOfPatient].gender = MAN; //Заменяем на мужской

}

savingPatients(patients, amountOfPatients, "patients.txt"); //Сохраняем изменения

coutGreenText(DATA\_BEEN\_CHANGED);

}

else {

coutRedText(DATA\_HAS\_NOT\_BEEN\_CHANGED);

}

}

//Функция редактирования даты рождения

void dataOfBirthPatientEdit(dataPatients\*(&patients), int &amountOfPatients, int numberOfPatient) {

//Ввод даты рождения пациента

int day, month, year;

while (true) { //Выполняем цикл пока не будет правильный ввод

//Вводим день, месяц, год по календарному принципу

cout << DAY; cin >> day;

cout << MONTH; cin >> month;

cout << YEAR; cin >> year;

if (validationInputData(day, month, year)) { //Если данные введены правильно и соответсвуют корректному тип присваиваиваем значения

menuConfirmationEdit(DATA\_OF\_BIRTH); //Меню подтверждения

if (confirmation()) { //Если согласны на изменение данных

patients[numberOfPatient].dateOfBirth.day = day;

patients[numberOfPatient].dateOfBirth.month = month;

patients[numberOfPatient].dateOfBirth.year = year;

savingPatients(patients, amountOfPatients, "patients.txt"); //Сохраняем изменения

coutGreenText(DATA\_BEEN\_CHANGED);

}

else {

coutRedText(DATA\_HAS\_NOT\_BEEN\_CHANGED);

}

break;

}

else { //Если данные введены неправильно , повторный ввод

coutRedText(INCCORECT\_INPUT\_DATE\_OF\_BIRTH);

}

}

}

//Редактирование данных учетной записи

void editAccount(account\* (&accounts), int amountOfAccounts, int activeAccount) {

printAllAccounts(accounts, amountOfAccounts); //Вывод всех учетных записей

cout << "\n\nДля того , чтобы вернуться назад нажмите - '0'\nВведите номер аккаунта от 1 до " << amountOfAccounts << " который хотите изменить << ";

int numberEditAccount = inputNumber(BACK, amountOfAccounts); //Ввод номера учетной записи, которую хотим редактировать

if (numberEditAccount - 1 == activeAccount) { //Если выбрали свой аккаунт, редактировать нельзя

HANDLE hOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOutput, FOREGROUND\_RED);

cout << YOU\_NOT\_EDIT\_ACCOUNT;

SetConsoleTextAttribute(hOutput, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

}

else if (numberEditAccount != BACK) { //Если не выбрали вернуться назад

system("cls");

numberEditAccount--;

cout << DATA\_ACCOUNT\_FOR\_EDIT;

printAccount(accounts, numberEditAccount); //Вывод данных выбранной учетной записи

menuAccountEdit(); //Меню редактирования учетной записи

int choiceAccountEdit = inputNumber(EXIT\_FROM\_EDIT\_ACCOUNT, ROLE\_EDIT);

switch (choiceAccountEdit) {

case EDIT\_LOGIN\_ACCOUNT: { //Редактирование логина

loginEdit(accounts, amountOfAccounts, numberEditAccount);

break;

}

case ACCESS\_EDIT: { //Измение доступа

accessEdit(accounts, amountOfAccounts, numberEditAccount);

break;

}

case ROLE\_EDIT: { //Изменение роли

roleEdit(accounts, amountOfAccounts, numberEditAccount);

break;

}

case EXIT\_FROM\_EDIT\_ACCOUNT: { //Выход из редактирования аккаунта

break;

}

}

}

}

//Функция изменение доступа учетной записи

void accessEdit(account\* accounts, int amountOfAccounts, int numberAccount) {

menuConfirmationEdit(ACCES); //Меню подтверждения

if (confirmation()) { //Если согласны

if (accounts[numberAccount].access == NO\_ACCESS) { //Если было без доступа

accounts[numberAccount].access = ACCESS; //Заменяем на с доступом

}

else { //Если было с доступом

accounts[numberAccount].access = NO\_ACCESS; //Заменяем на без доступа

}

savingAccount(accounts, amountOfAccounts, "accounts.txt"); //Сохраняем изменения

coutGreenText(DATA\_BEEN\_CHANGED);

}

else {

coutRedText(DATA\_HAS\_NOT\_BEEN\_CHANGED);

}

}

//Функция изменения пароля учетной записи

void passwordEdit(account\* accounts, int amountOfAccounts, int numberOfAccount) {

bool edit = false;

cout << INPUT\_PASSWORD<< BACK\_ON\_0;

string password = inputValidationPasswordOrLogin(); //Пароль вводим

if (password != EXIT\_FROM\_INPUT) {

cout << REPEAT\_PASSWORD;

string validationPassword = inputValidationPasswordOrLogin();

if (validationPassword != EXIT\_FROM\_INPUT) {

if (password == validationPassword) {

menuConfirmationEdit(PASSWORD);

if (confirmation()) { //Если согласны на изменения пароля

accounts[numberOfAccount].hash = generationHashForPassword(password); //Генерируем хеш

accounts[numberOfAccount].salt = generateSalt(SALT\_SIZE); //Генерируем соль

savingAccount(accounts, amountOfAccounts, "accounts.txt"); //Сохраняем изменения

edit = true;

coutGreenText(PASSWORD\_SUCCESSFULLY\_CHANGED);

}

}

else {

coutRedText(PASSWORDS\_DO\_NOT\_MATCH);

}

}

}

if (edit == false) {

coutRedText(PASSWORD\_HAS\_NOT\_BEEN\_CHANGED);

}

}

//Функция изменения логина учетной записи

void loginEdit(account\* accounts, int amountOfAccounts, int numberAccount) {

cout << INPUT\_LOGIN<< BACK\_ON\_0;

string login = inputValidationPasswordOrLogin(); //Вводим логин

if (login != EXIT\_FROM\_INPUT) { //Если не выбрали вернуться назад

bool correct = true; //Разрешение для регистрации

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].login == login) { //Если логин уже занят

correct = false; //Занят

coutRedText( LOGIN\_IS\_BUSY);

break;

}

}

if (correct == true) { //Если логин введен верно

menuConfirmationEdit(LOGIN); //Меню подтверждения

if (confirmation()) { //Если согласны на изменения данных

accounts[numberAccount].login = login; //Заменяем старые данные на новые

savingAccount(accounts, amountOfAccounts, "accounts.txt"); //Сохраняем изменения

coutGreenText(DATA\_BEEN\_CHANGED);

}

else {

coutRedText(DATA\_HAS\_NOT\_BEEN\_CHANGED);

}

}

}

}

**Основные функции из файла MainAdmin.cpp**

//Работа с учетными записями

void workWithAccounts(account\* (&accounts), int& amountOfAccounts, int activeAccount, dataPatients\* (&patients), int amountOfPatients) {

menuWorkWithAccounts(); //Меню работы с учетными записями

int choiceWorkWithAccounts = inputNumber(EXIT\_FROM\_WORKING\_WITH\_ACCOUNTS, DELETE\_ACCOUNTS); //Выбор действия и проверка на численный ввод

switch (choiceWorkWithAccounts) { //Передаем выбор действия

case VIEW\_ALL\_ACCOUNTS: { //Просмотр всех учетных записей

printAllAccounts(accounts, amountOfAccounts);

break;

}

case EDIT\_ACCOUNTS: { //Редактирование учетных записей

editAccount(accounts, amountOfAccounts, activeAccount);

break;

}

case DELETE\_ACCOUNTS: { //Удаление учетных записей

deleteAccounts(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients);

break;

}

case EXIT\_FROM\_WORKING\_WITH\_ACCOUNTS: { //Выход из работы с учетными записями

return;

}

}

system("pause");

workWithAccounts(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients);

}

//Функционал главного администратора

void mainAdmin(account\* (&accounts), int& amountOfAccounts, int activeAccount, dataPatients\* (&patients), int& amountOfPatients) {

mainMenuMainAdmin(); //Меню работы с учетными записями

int choiceMainAdmin = inputNumber(EXIT\_FROM\_MAIN\_ADMIN, EDIT\_PASSWORD\_MAIN\_ADMIN); //Выбор действия и проверка на численный ввод

switch (choiceMainAdmin) { //Передаем выбор действия

case WORK\_WITH\_ACCOUNTS\_FOR\_MAIN\_ADMIN: { //Работа с учетными записями

workWithAccounts(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients);

break;

}

case PROCCESSING\_PATIENTS: { //Обработка данных о пациентах

if (amountOfPatients != DATA\_EMPTY) { //Если есь пациенты в массиве

proccesingPatientsData(patients, amountOfPatients);

}

else {

coutBlueText(PATIENTS\_NO);

}

break;

}

case EDIT\_PASSWORD\_MAIN\_ADMIN: { //Редактирование пароля

passwordEdit(accounts, amountOfAccounts, activeAccount);

break;

}

case EXIT\_FROM\_MAIN\_ADMIN: { //Выход из функционала главного администратора

exitFromAccount(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients, mainAdmin);

return;

}

}

system("pause");

mainAdmin(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients);

}

**Основные функции из файла Functions.cpp**

//Открытия файла учетных записей

void openingFileAccounts(account\* (&accounts), int& amountOfAccounts, string fileAccounts) {

ifstream reading(fileAccounts); //Поток для чтения из файла

if (!reading.is\_open()) { //Если файла с учетными записями нету. Регистрируемся в качестве главного админа

cout << REGISTR\_AS\_MAIN\_ADMIN;

cout << INPUT\_LOGIN; string newLogin = inputValidationPasswordOrLogin();

cout << INPUT\_PASSWORD; string newPassword = inputValidationPasswordOrLogin();

cout << INPUT\_REPEAT\_PASSWORD; string repeatNewPassword = inputValidationPasswordOrLogin();

amountOfAccounts++; //увеличение количество учетных записей в структуре на 1

accounts[amountOfAccounts - 1].login = newLogin;

accounts[amountOfAccounts - 1].role = ADMIN\_MAIN;

accounts[amountOfAccounts - 1].access = ACCESS;

accounts[amountOfAccounts - 1].hash = generationHashForPassword(newPassword);

accounts[amountOfAccounts - 1].salt = generateSalt(SALT\_SIZE);

savingAccount(accounts, amountOfAccounts, fileAccounts); //Сохранение изменений в файл

coutYellowText(YOU\_HAVE\_CREATED\_FILE\_AND\_REGISTERED\_MAIN\_ADMIN);

system("pause");

}

else { //Если файл с учетными записями есть

readingFileAccounts(accounts, amountOfAccounts, fileAccounts);//Считываем данные из файла в массив

}

reading.close(); //Закрываем файл

}

//Считывание учетных записей в массив

void readingFileAccounts(account\*(&accounts), int &amountOfAccounts, string fileAccounts){

ifstream reading(fileAccounts);

int numberAccount = 0;

while (!reading.eof()) { //Пока файл не пустой

if (numberAccount < RESERVE\_AMOUNT\_OF\_ACCOUNTS) {

reading >> accounts[numberAccount].login //Считываем данные

>> accounts[numberAccount].hash

>> accounts[numberAccount].salt

>> accounts[numberAccount].role

>> accounts[numberAccount].access;

if (accounts[numberAccount].login.empty()) {

break;

}

numberAccount++; //Считаем количество учетных записей

}

}

amountOfAccounts = numberAccount; //Количество аккаунтов

reading.close();

}

//Сохраняем учетную запись

void savingAccount(account\* accounts, int amountOfAccounts, string fileName) {

ofstream record(fileName, ios::out); //Создаем поток для записи, открываем файл и делаем его пустым

if (record) { //если файл открылся

//Записываем данные в файл

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

record << accounts[numberAccount].login << " "

<< accounts[numberAccount].hash << " "

<< accounts[numberAccount].salt << " "

<< accounts[numberAccount].role << " ";

if (numberAccount < amountOfAccounts - 1) { //Если элемент не последний делаем отступ на след строчку

record << accounts[numberAccount].access << endl;

}

else { //Если элемнт последний делаем отступ на след строчку

record << accounts[numberAccount].access;

}

}

}

else { //Если файл не открылся

coutRedText( ERROR\_OPEN\_FAIL);

}

record.close(); //Закрываем файл

}

//Выход из аккаунта

void exitFromAccount(account\* (&accounts), int &amountOfAccounts, int activeAccount, dataPatients\* (&patients), int &amountOfPatients, void user(account\*(&accounts), int &amountOfAccounts, int activeAccount, dataPatients\* (&patients), int &amountOfPatients)) {

system("cls");

cout << CONFIRMATION\_EXIT\_FROM\_ACCOUNT;

if (confirmation()) { //Если согласны на выход из аккаунта

autorization(accounts, amountOfAccounts, patients, amountOfPatients);

}

else {

user(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients);

}

}

//Удаление аккаунта

void deleteAccounts(account\* (&accounts), int& amountOfAccounts, int activeAccount, dataPatients\*(&patients), int& amountOfPatients) { //Передаем ссылку на массив, чтоб создать копию. Удаление данных

printAllAccounts(accounts, amountOfAccounts); //Вывод всех учетных записей

cout << "\n\nДля того , чтобы вернуться назад нажмите - '0'\nВведите номер аккаунта от 1 до " << amountOfAccounts << " которого хотите удалить << ";

int numberDeleteAccount = inputNumber(BACK, amountOfAccounts); //Ввод номера данного , который хотите удалить

if (numberDeleteAccount == BACK) { //Если вернуться назад

return;

}

else if (numberDeleteAccount == activeAccount + 1) { //Если выбрали свой аккаунт

coutRedText( NOT\_DELETE\_MAIN\_ADMIN);

return;

}

else {

cout << CONFIRMATION\_DELETE\_ACCOUNT;

if (confirmation()) { //Если согласны на ужаление учетной записи

numberDeleteAccount--; //Уменьшаем на 1

for (int numberOfAccount = numberDeleteAccount; numberOfAccount < amountOfAccounts - 1; numberOfAccount++) {

accounts[numberOfAccount] = accounts[numberOfAccount + 1];

}

amountOfAccounts--; //Уменьшаем количество аккаунтов

savingAccount(accounts, amountOfAccounts, "accounts.txt"); //Сохраняем изменение

coutGreenText(ACCOUNT\_DELETE);

}

else {

coutRedText(ACCOUNT\_NOT\_DELETE);

}

}

}

//Копирование аккаунтов

void copyAccount(account\* (&accountsNew), account\* (accountsOld), int amountOfAccounts) { //передаем два массива

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

accountsNew[numberAccount] = accountsOld[numberAccount]; //каждый элемент массива , присваиваем в новый массив

}

}

//Одобрение заявки на активацию аккаунта

void activationAccount(account\* (&accounts), int& amountOfAccounts, int activeAccount) {

int amountOfAccountNoActivation = accountsWithoutAccess(accounts, amountOfAccounts); //Количество аккаунтов без доступа

account\* accountsWithAccess = createArrOfAccountsWithoutAccess(accounts, amountOfAccounts, amountOfAccountNoActivation); //Создание массива аккаунтов без доступа

if (amountOfAccountNoActivation != DATA\_EMPTY) {

printAccountsForConfirmAdd(accounts, amountOfAccounts); //Вывод завок на регистрацию

confirmApplication(accounts, accountsWithAccess, amountOfAccounts, amountOfAccountNoActivation, activeAccount); //Функция выбора одобрения заявок

}

else { //Проверка на наличие заявок

coutBlueText( NO\_APPLICATION\_REQUESTS);

}

}

void confirmApplication(account\* (&accounts), account\* (&accountsWithAccess), int amountOfAccounts, int amountOfAccountNoActivation, int activeAccount) {

cout << ENTER\_APPLICATION\_NUMBER;

int numberApplication = inputNumber(BACK, amountOfAccountNoActivation); //Выбор номера заявки для активации

if (numberApplication == BACK) { //Если вернуться назад

return;

}

else {

coutBlueText(CONFIRMATION\_ACTIVATION);

if (confirmation()) { //Если согласны на активацию аккаунта

numberApplication--; //Уменьшаем на 1

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].login == accountsWithAccess[numberApplication].login) {

accounts[numberAccount].access = ACCESS; //Меняем на доступ

break;

}

}

coutGreenText( ACCOUNT\_ACTIVATED);

savingAccount(accounts, amountOfAccounts, "accounts.txt");

}

else {

coutRedText(ACCOUNT\_NOT\_ACTIVATED);

}

}

}

//Подсчет аккаунтов без доступа

int accountsWithoutAccess(account\* (&accounts), int amountOfAccounts) {

int amountOfAccountNoActivation = DATA\_EMPTY; //Инициалзация переменной счетчика аккаунтов без доступа

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].access == NO\_ACCESS) { //Проверка на доступ аккаунта

amountOfAccountNoActivation++; //Увеличение аккаунтов без доступа

}

}

return amountOfAccountNoActivation; //Возвращаем количество аккаунтов без доступа

}

//Запись аккаунтов без доступа в отдельный массив

account\* createArrOfAccountsWithoutAccess(account\* (&accounts), int amountOfAccounts, int amountOfAccountNoActivation) {

account\* accountsWithAccess = new account[amountOfAccountNoActivation]; //Выделяем память под новый массив

amountOfAccountNoActivation = DATA\_EMPTY; //Инициализация счетчика аккаунтов без доступа

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].access == NO\_ACCESS) { //Проверка на доступ аккаунтв

accountsWithAccess[amountOfAccountNoActivation].login = accounts[numberAccount].login; //Переписываем данные в новый массив

amountOfAccountNoActivation++; //Увеличиваем количество аккаунтов без доступа

}

}

return accountsWithAccess; //Возвращаем указатель на массив аккаунтов без доступа

}

//Вывод акккаунтов не имеющих доступа

void printAccountsForConfirmAdd(account\* (&accounts), int amountOfAccounts) {

system("cls");

cout << BACK\_ON\_0;

coutYellowText(ACTIVATION\_REQUESTS);

cout << setw(17) << "Логин:" << setw(10) << "Роль:" << setw(14) << "Доступ:\n";

int amountOfAccountNoActivation = DATA\_EMPTY; //Инициализация переменной счетчика аккаунтов без доступа

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].access == NO\_ACCESS) { //Проверка на доступ аккаунта и вывод соответ данных

amountOfAccountNoActivation++; //Увеличение аккаунтов без доступа

cout << "Данные №" << amountOfAccountNoActivation << ": ";

printAccount(accounts, numberAccount);

}

}

}

//Блокировка заявки на активацию аккаунта

void blockingAccount(account\* (&accounts), int& amountOfAccounts, int activeAccount) {

int amountOfAccountWithActivation = accountsWithoAccess(accounts, amountOfAccounts); //Количество аккаунтов без доступа

account\* accountsWithAccess = createArrOfAccountsWithAccess(accounts, amountOfAccounts, amountOfAccountWithActivation); //Создание массива аккаунтов без доступа

if (amountOfAccountWithActivation != DATA\_EMPTY) {

printAccountsForConfirmBlocking(accounts, amountOfAccounts); //Вывод завок на регистрацию

confirmBlocking(accounts, accountsWithAccess, amountOfAccounts, amountOfAccountWithActivation, activeAccount); //Функция выбора одобрения заявок

}

else { //Проверка на наличие заявок

coutBlueText(NO\_BLOCK\_REQUESTS);

}

}

void confirmBlocking(account\* (&accounts), account\* (&accountsWithAccess), int amountOfAccounts, int amountOfAccountWithActivation, int activeAccount) {

coutBlueText(ENTER\_FOR\_BLOCKING);

int numberApplication = inputNumber(BACK, amountOfAccountWithActivation); //Выбор номера заявки для активации

if (numberApplication == BACK) { //Если вернуться назад

return;

}

else if (accounts[numberApplication - 1].login == accounts[activeAccount].login) { //Если вы хотите заблокировать свой аккаунтт

coutRedText( YOU\_CANNOT\_BLOCK\_YOUR\_ACCOUNT);

return;

}

else if (accounts[numberApplication - 1].role == ADMIN\_MAIN) { //Если хотите заблокировать главного админа

coutRedText( YOU\_CANNOT\_DISABLE\_MAIN\_ADMIN);

return;

}

else {

coutBlueText(CONFIRMATION\_BLOCKING);

if (confirmation()) { //Если согласны на активацию аккаунта

numberApplication--; //Уменьшаем на 1

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].login == accountsWithAccess[numberApplication].login) {

accounts[numberAccount].access = NO\_ACCESS; //Меняем на доступ

break;

}

}

coutGreenText(ACCOUNT\_BLOCKING);

savingAccount(accounts, amountOfAccounts, "accounts.txt");

}

else {

coutRedText( ACCOUNT\_NOT\_BLOCKED);

}

}

}

//Подсчет аккаунтов с доступа

int accountsWithoAccess(account\* (&accounts), int amountOfAccounts) {

int amountOfAccountWithActivation = DATA\_EMPTY; //Инициалзация переменной счетчика аккаунтов без доступа

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].access == ACCESS) { //Проверка на доступ аккаунта

amountOfAccountWithActivation++; //Увеличение аккаунтов без доступа

}

}

return amountOfAccountWithActivation; //Возвращаем количество аккаунтов без доступа

}

//Запись аккаунтов с доступов в отдельный массив

account\* createArrOfAccountsWithAccess(account\* (&accounts), int amountOfAccounts, int amountOfAccountWithActivation) {

account\* accountsWithAccess = new account[amountOfAccountWithActivation]; //Выделяем память под новый массив

amountOfAccountWithActivation = DATA\_EMPTY; //Инициализация счетчика аккаунтов без доступа

for (int numberAccount = 0; numberAccount < amountOfAccounts; numberAccount++) {

if (accounts[numberAccount].access == ACCESS) { //Проверка на доступ аккаунтв

accountsWithAccess[amountOfAccountWithActivation].login = accounts[numberAccount].login; //Переписываем данные в новый массив

amountOfAccountWithActivation++; //Увеличиваем количество аккаунтов без доступа

}

}

return accountsWithAccess; //Возвращаем указатель на массив аккаунтов без доступа

}

//Чтение данных пациентов в структуру

void openingFilePatients(dataPatients\* (&patients), int& amountOfPatients, string fileName) {

ifstream reading(fileName); //поток для чтения из файла

int numberPatients = 0;

while (!reading.eof()) { //Пока файл не пустой

if (numberPatients < RESERVE\_AMOUNT\_OF\_PATIENT) {

reading >> patients[numberPatients].surname

>> patients[numberPatients].name

>> patients[numberPatients].patronymic

>> patients[numberPatients].gender

>> patients[numberPatients].dateOfBirth.day

>> patients[numberPatients].dateOfBirth.month

>> patients[numberPatients].dateOfBirth.year

>> patients[numberPatients].town

>> patients[numberPatients].phoneNumber

>> patients[numberPatients].diagnosis;

if (patients[numberPatients].surname.empty()) {

break;

}

numberPatients++; //Считаем количество пациентов

}

}

amountOfPatients = numberPatients; //Присваиваем количество пациентов

reading.close(); //Закрываем файл

}

//Добавление пациента

void addPatient(dataPatients\* (&patients), int& amountOfPatients) {

//Создаем новый временный массив

dataPatients\* buf;

buf = new dataPatients[amountOfPatients];

copy(buf, patients, amountOfPatients);

//Добавляем количество пациентов

amountOfPatients++;

patients = new dataPatients[amountOfPatients];

copy(patients, buf, --amountOfPatients);

//Заполняем данные для добавления

system("cls");

cout << INPUT\_DATA;

cout << INPUT\_ADD\_SURNAME; patients[amountOfPatients].surname = inputValidationString(LETTERS\_SYMBOLS);

cout << INPUT\_ADD\_NAME; patients[amountOfPatients].name = inputValidationString(LETTERS\_SYMBOLS);

cout << INPUT\_ADD\_PATRONYMIC; patients[amountOfPatients].patronymic = inputValidationString(LETTERS\_SYMBOLS);

cout << INPUT\_ADD\_GENDER; patients[amountOfPatients].gender = inputValidationString(LETTERS\_SYMBOLS);

cout << INPUT\_ADD\_DATE\_BIRTH; inputDateOfBirth(patients, amountOfPatients); cin.ignore();

cout << INPUT\_ADD\_TOWN; patients[amountOfPatients].town = inputValidationString(LETTERS\_SYMBOLS);

cout << INPUT\_ADD\_PHONE; patients[amountOfPatients].phoneNumber = inputValidationString(NUMBER\_SYMBOLS);

cout << INPUT\_ADD\_DIAGNOSIS; patients[amountOfPatients].diagnosis = inputValidationString(LETTERS\_SYMBOLS);

cout << CONFIRMATION\_ADD;

if (confirmation()) { //Если согласны на добавление пациента

amountOfPatients++; //Увеличиваем количество пациентов

savingPatients(patients, amountOfPatients, "patients.txt"); //Сохраняем изменения

coutGreenText(DATA\_HAS\_ADD);

}

else {

coutRedText(DATA\_NOT\_HAS\_ADD);

}

delete[]buf; //Удаляем временный массив

}

//Ввод даты рождения

void inputDateOfBirth(dataPatients\*(&patients), int amountOfPatients) {

//Ввод даты рождения пациента

int day, month, year;

while (true) { //Выполняем цикл пока не будет правильный ввод

//Вводим день, месяц, год по календарному принципу

cout << DAY; cin >> day;

cout << MONTH; cin >> month;

cout << YEAR; cin >> year;

if (validationInputData(day, month, year)) { //Если данные введены правильно и соответсвуют корректному тип присваиваиваем значения

patients[amountOfPatients].dateOfBirth.day = day;

patients[amountOfPatients].dateOfBirth.month = month;

patients[amountOfPatients].dateOfBirth.year = year;

break;

}

else { //Если данные введены неправильно , повторный ввод

coutRedText( INCCORECT\_INPUT\_DATE\_OF\_BIRTH);

}

}

}

//Добавление пациента , увеличиваем размер структуры

void copy(dataPatients\* (&patientstNew), dataPatients\* (patientsOld), int amountOfPatients) {

for (int numberPatient = 0; numberPatient < amountOfPatients; numberPatient++) {

patientstNew[numberPatient] = patientsOld[numberPatient]; //каждый элемент массива , присваиваем в новый массив

}

}

//Сохрание данных пациентов в файл

void savingPatients(dataPatients \* patients, int amountOfPatients, string fileName) {

ofstream record(fileName, ios::out); //Создаем поток для записи, открываем файл и делаем его пустым

if (record) { //если файл открылся

//Записываем данные в файл

for (int numberPatient = 0; numberPatient < amountOfPatients; numberPatient++) {

record << patients[numberPatient].surname << " "

<< patients[numberPatient].name << " "

<< patients[numberPatient].patronymic << " "

<< patients[numberPatient].gender << " "

<< patients[numberPatient].dateOfBirth.day << " "

<< patients[numberPatient].dateOfBirth.month << " "

<< patients[numberPatient].dateOfBirth.year << " "

<< patients[numberPatient].town << " "

<< patients[numberPatient].phoneNumber << " ";

if (numberPatient < amountOfPatients - 1) { //Если элемент не последний делаем отступ на след строчку

record << patients[numberPatient].diagnosis << endl;

}

else { //Если элемнт последний делаем отступ на след строчку

record << patients[numberPatient].diagnosis;

}

}

}

else { //Если файл не открылся

coutRedText(ERROR\_OPEN\_FAIL);

}

record.close(); //Закрываем файл

}

//Удаление пациента

void deletePatient(dataPatients\* (&patients), int& amountOfPatients) {

printAllPatients(patients, amountOfPatients); //Вывод всех пациентов

cout << "\n\nДля того , чтобы вернуться назад нажмите - '0'\nВведите номер пациента от 1 до " << amountOfPatients << " которого хотите удалить << ";

int numberDeletePatient = inputNumber(BACK, amountOfPatients); //Ввод номера данного , который хотите удалить

if (numberDeletePatient == BACK) { //Если выбрали вернуться назад

return;

}

else {

cout << CONFIRMATION\_DELETE\_PATIENT;

if (confirmation()) { //Если подтверддили действие

numberDeletePatient--; //Уменьшаем номер пациента

for (int numberOfPatient = numberDeletePatient; numberOfPatient < amountOfPatients - 1; numberOfPatient++) {

patients[numberOfPatient] = patients[numberOfPatient + 1];

}

amountOfPatients--; //Уменьшаем количетво пациентов

savingPatients(patients, amountOfPatients, "patients.txt"); //Сохраняем изменение

coutGreenText(PATIENT\_DELETE);

}

else {

coutRedText(PATIENT\_NOT\_DELETE);

}

}

}

**Основные функции из файла SearchSort.cpp**

//Главная функция меню поиска

void search(dataPatients\* (&patients), int &amountOfPatients) {

string searchString;

bool isCountData = false;

menuSearch(); //Вызов меню поиска

int choiceSearch = inputNumber(BACK\_FROM\_SEARCH, SEARCH\_FOR\_TOWN); //Выбор действия в меню и проверка на корректность ввода

switch (choiceSearch) { //Выбор какое действие выполнить

case SEARCH\_FOR\_SURNAME: { //Поиск по фамилии(1)

searchString = inputAndOutputSearchString(SURNAME, searchString);

searchForSurname(patients, amountOfPatients, isCountData, searchString);

break;

}

case SEARCH\_FOR\_NAME: { //Поиск по имени(2)

searchString = inputAndOutputSearchString(NAME, searchString);

searchForName(patients, amountOfPatients, isCountData, searchString);

break;

}

case SEARCH\_FOR\_DIAGNOSIS: { //Поиск по диагнозу(3)

searchString = inputAndOutputSearchString(DIAGNOSIS, searchString);

searchForDiagnosis(patients, amountOfPatients, isCountData, searchString);

break;

}

case SEARCH\_FOR\_TOWN: { //Поиск по городу(5)

searchString = inputAndOutputSearchString(TOWN, searchString);

searchForTown(patients, amountOfPatients, isCountData, searchString);

break;

}

case BACK\_FROM\_SEARCH: //Назад (0)

return;

}

system("pause");

search(patients, amountOfPatients);

}

//Ввод искомых данных

string inputAndOutputSearchString(string SURNAME, string searchString) {

system("cls");

cout << "Введите данные " << SURNAME << " которые хотите найти : \n << ";

searchString = inputValidationString(LETTERS\_SYMBOLS);

cout << setw(100) << FIND\_DATA\_AFTER\_SEARCH;

return searchString;

}

//Поиск по фамилии

void searchForSurname(dataPatients\* patients, int amountOfPatients, int countData, string searchString) { //Поиск по фамилии

for (int numberOfPatientsSearch = 0; numberOfPatientsSearch < amountOfPatients; numberOfPatientsSearch++) {

string str = patients[numberOfPatientsSearch].surname; //Присваиваем str строку в которой будет искать substr

size\_t substr = str.find(searchString); //Возвращаемый тип для возрата размера. Ищет в str строке первое вхождение другой строки substr

if (substr != string::npos) { //Пока не конец строки

countData = true; //Если подстрока имеется

printPatient(patients, numberOfPatientsSearch); //Вывод найденных данных в консоль

}

}

reSearch(patients, amountOfPatients, countData, searchString, SURNAME, searchForSurname); //Если данные не найдены, повторный поиск

}

//Повторный поиск

if (countData == false) { //Если данные не найдены

coutBlueText( DATA\_NOT\_FOUND);

int continueOrBack = inputNumber(BACK\_TO\_SEARCH\_MENU, CONTINUE); //Выбираем продолжить поиск или вернуться в меню и проверяем на корректный ввод

if (continueOrBack == CONTINUE) { //Если выбрали продожить

searchString = inputAndOutputSearchString(SURNAME, searchString);

searchForSurname(patients, amountOfPatients, countData, searchString); //выбираем выриант поиска

}

}

}

//Главная функция сортировки

void sort(dataPatients\* (&patients), int &amountOfPatients) {

menuSort(); //Вызов меню сортировки

int choiceSort = inputNumber(BACK\_FROM\_SORT, SORT\_FOR\_TOWN); //Выбор действия в меню и проверка на корректность ввода

system("cls");

switch (choiceSort) { //Выбор какое действие выполнить

case SORT\_FOR\_SURNAME: { //Сортировка по фамилии(1)

sortForSurnameOrNameOrTown(patients, amountOfPatients, choiceSort, SURNAME, isSortForSurnameAscending, isSortForSurnameDescending);

break;

}

case SORT\_FOR\_NAME: { //Сортировка по имени(2)

sortForSurnameOrNameOrTown(patients, amountOfPatients, choiceSort, NAME, isSortForNameAscending, isSortForNameDescending);

break;

}

case SORT\_FOR\_AGE: { //Сортировка по возрасту(3)

sortForAge(patients, amountOfPatients, choiceSort);

break;

}

case SORT\_FOR\_TOWN: { //Сортировка по месту жительства(5)

sortForSurnameOrNameOrTown(patients, amountOfPatients, choiceSort, TOWN, isSortForTownAscending, isSortForTownDescending);

break;

}

case BACK\_FROM\_SORT: //Назад (0)

return;

}

system("pause");

sort(patients, amountOfPatients);

}

//Сортировка по фамилии,имени, месту жиельства

void sortForSurnameOrNameOrTown(dataPatients\* patients, int amountOfPatients, int choiceSort, string SURNAME, bool isSortForSurnameAscending(dataPatients patients, dataPatients patientsPlus), bool isSortForSurnameDescending(dataPatients patients, dataPatients patientsPlus)) {

cout << "Сортировка данных по " << SURNAME << ":\n1.От А до Я\n2.От Я до А\n0.Назад\n << ";

int sortSelection = inputNumber(BACK, DESCENDING); //Выбор вида сортировки и проверка на корректный ввод

system("cls");

if (sortSelection == ASCENDING) { //Если от А до Я

sort(patients, patients + amountOfPatients, isSortForSurnameAscending); //Сортировка от А до Я

cout << setw(80) << "Пациенты отсортированы по " << SURNAME << "(A - Я)\n";

printAllPatients(patients, amountOfPatients); //Вывод отсортированных данных пациентов

}

else if (sortSelection == DESCENDING) { //Если от Я до А

sort(patients, patients + amountOfPatients, isSortForSurnameDescending); //Сортировка от Я до А

cout << setw(80) << "Пациенты отсортированы по " << SURNAME << "(Я - А)\n";

printAllPatients(patients, amountOfPatients); //Вывод отсортированных данных пациентов

}

}

//Сортировка по возрасту

void sortForAge(dataPatients\* patients, int amountOfPatients, int choiceSort) {

cout << MENU\_SORT\_FOR\_AGE;

int sortSelection = inputNumber(BACK, DESCENDING); //Выбор вида сортировки и проверка на корректный ввод

system("cls"); //Очистка консоли

if (sortSelection == ASCENDING) { //Если выбор по возрастанию

sort(patients, patients + amountOfPatients, isSortForDayAscending); // Функции

sort(patients, patients + amountOfPatients, isSortForMonthAscending); // сортировки

sort(patients, patients + amountOfPatients, isSortForYearAscending); //Возраста по возрастанию

cout << setw(80) << SORT\_FOR\_AGE\_ASCENDING;

printAllPatients(patients, amountOfPatients); //Вывод всех отсортированных данных в консоль

}

else if (sortSelection == DESCENDING) { //Если по выбор по убыванию

sort(patients, patients + amountOfPatients, isSortForDayDescending); //Функции

sort(patients, patients + amountOfPatients, isSortForMonthDescending);//Сортирвки

sort(patients, patients + amountOfPatients, isSortForYearDescending); //Возраста по убыванию

cout << setw(80) << SORT\_FOR\_AGE\_DESCENDING;

printAllPatients(patients, amountOfPatients); //Вывод всех отсортированных данных в консоль

}

//Главная функция фильтрации

void filtration(dataPatients\* (&patients), int &amountOfPatients) {

menuFiltration(); //Вызов меню фильтрации

int choiceFiltr = inputNumber(BACK\_FROM\_FILTR, FILTR\_FOR\_AGE\_AND\_DIAGNOSIS); //Выбор вида фильтрации и корректный ввод

system("cls");

switch (choiceFiltr) { //Выбор какое действие выполнить

case FILTR\_FOR\_GENDER: { //Фильтрация по полу

filtrationForGender(patients, amountOfPatients);

break;

}

case FILTR\_FOR\_AGE\_AND\_DIAGNOSIS: { //Фильтрация по возрасту и диагнозу

filtrationForAgeAndDiagnosis(patients, amountOfPatients);

break;

}

case BACK\_FROM\_FILTR: //Назад (0)

return;

}

}

//Фильтрация по полу

void filtrationForGender(dataPatients\* patients, int amountOfPatients) {

menuFiltratonForGender(); //Меню фильтрации по полу

int choiceFiltrGender = inputNumber(BACK, WOMAN\_GENDER); //Выбираем пол для фильтрации

if (choiceFiltrGender == MAN\_GENDER) { //Если мужской пол

filtrForGender(patients, amountOfPatients, choiceFiltrGender, MAN\_GENDER, MAN); //Выводим мужчин

}

else if (choiceFiltrGender == WOMAN\_GENDER) { //Если женский пол

filtrForGender(patients, amountOfPatients, choiceFiltrGender, WOMAN\_GENDER, WOMAN); //Выводим женщиг

}

}

void filtrForGender(dataPatients\* patients, int amountOfPatients, int choiceFiltrGender, int MAN\_GENDER, string MAN) {

bool correct = false; //Для проверки есть ли пациенты с таким полом

cout << setw(60) << "Пациенты " << MAN << " пола:\n";

for (int numberPatient = 0; numberPatient < amountOfPatients; numberPatient++) {

if (patients[numberPatient].gender == MAN) {

correct = true;

printPatient(patients, numberPatient);

}

}

if (correct == false) { //Пациентов с таким гендерным полом нет

coutBlueText(NO\_PATIENTS\_GENDER);

}

}

//Фильтрация по возрасту и диагнозу

void filtrationForAgeAndDiagnosis(dataPatients\* patients, int amountOfPatients) {

//Устанавливаем нанешнюю дату

SYSTEMTIME st;

GetLocalTime(&st);

cout << st.wDay << " " << st.wMonth << " " << st.wYear << endl;

cout << WHO\_AGE\_AND\_DIAGNOSIS;

cout << "Старше << "; //Вводим возраст , который хотим найти

int filtrAge = inputNumber(MIN\_AGE, MAX\_AGE);

cout << "Диагноз << "; //Вводим диагноз , который хотим найти

string filtrDiagnosis = inputValidationString(LETTERS\_SYMBOLS);

cout << "\nПациенты старше " << filtrAge << " лет и с диагнозом " << filtrDiagnosis << ":\n";

filtrForAgeAndDiagnosis(patients, amountOfPatients, filtrAge, filtrDiagnosis, st);

}

void filtrForAgeAndDiagnosis(dataPatients\* patients, int amountOfPatients, int filtrAge, string filtrDiagnosis, SYSTEMTIME st) {

int patientAge, patientMonth, patientDay;

bool isCountData = false;

//Высчитываем полное количество лет у пациентов

for (int numberPatient = 0; numberPatient < amountOfPatients; numberPatient++) {

patientAge = st.wYear - patients[numberPatient].dateOfBirth.year;

patientMonth = st.wMonth - patients[numberPatient].dateOfBirth.month;

patientDay = st.wDay - patients[numberPatient].dateOfBirth.day;

if (patientMonth < 0 || patientDay < 0) patientAge = patientAge - 1;

if (patientMonth < 0 && patientDay <= 0) patientMonth = 12 + patientMonth - 1;

if (patientMonth < 0 && patientDay > 0) patientMonth = 12 + patientMonth;

if (patientMonth >= 0 && patientDay < 0) patientDay = patients[numberPatient].dateOfBirth.day + patientDay;

if (filtrAge <= patientAge && filtrDiagnosis == patients[numberPatient].diagnosis) {

isCountData = true;

printPatient(patients, numberPatient); //Вывод одного пациента в консоль

}

}

if (isCountData == false) {

coutRedText(SEARCH\_NOT\_FOUND);

}

}

**Основные функции из файла User.cpp**

//Функционал пользователя

void user(account\*(&accounts), int &amountOfAccounts, int activeAccount, dataPatients\*(&patients), int &amountOfPatients) {

mainUserMenu(); //Вызов главного меню пользователя

int choiceUser = inputNumber(EXIT\_FROM\_USER, EDIT\_PASSWORD); //Выбор действия пользователя и проверка на численный ввод

switch (choiceUser) { //Передаем выбор действия от пользователя

case VIEW\_ALL\_PATIENTS: { //Просмотр всех пациентов

if (amountOfPatients != DATA\_EMPTY) { //Если есть пациенты в массиве

printAllPatients(patients, amountOfPatients);

}

else { //Если нет пациентов в массиве

coutBlueText(NO\_PATIENTS);

}

break;

}

case VIEW\_NONRESIDENT: { //Иногородние пациенты

if (amountOfPatients != DATA\_EMPTY) { //Если есть пациенты в массиве

printNonresidentPatients(patients, amountOfPatients);

}

else { //Если нет пациентов в массиве

coutBlueText(NO\_PATIENTS);

}

break;

}

case SEARCH: { //Поиск

if (amountOfPatients != DATA\_EMPTY) { //Если есть пациенты в массиве

search(patients, amountOfPatients);

}

else { //Если нет пациентов в массиве

coutBlueText(NO\_PATIENTS);

}

break;

}

case SORT: { //Сортировка

if (amountOfPatients != DATA\_EMPTY) { //Если есть пациенты в массиве

sort(patients, amountOfPatients);

}

else { //Если нет пациентов в массиве

coutBlueText(NO\_PATIENTS);

}

break;

}

case FILTRATION: { //Фильтрация

if (amountOfPatients != DATA\_EMPTY) { //Если есть пациенты в массиве

filtration(patients, amountOfPatients);

}

else { //Если нет пациентов в массиве

coutBlueText(NO\_PATIENTS);

}

break;

}

case EDIT\_PASSWORD: { //Изменить пароль

passwordEdit(accounts, amountOfAccounts, activeAccount);

break;

}

case EXIT\_FROM\_USER: { //Выход из аккаунта админа

exitFromAccount(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients, user);

return;

}

}

system("pause");

user(accounts, amountOfAccounts, activeAccount, patients, amountOfPatients);