МИНОБРНАУКИ РОССИИ

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

**КУРСОВАЯ РАБОТА**по дисциплине «Программирование»  
**Тема:** Разработка электронной картотеки

|  |  |  |
| --- | --- | --- |
| Студенты гр. 0322: |  | Воловикова А.А. |
|  |  | Павлова А.Н. |
| Преподаватель |  | Кузьмин С.А. |

Санкт-Петербург

2021

ЗАДАНИЕ  
НА КУРСОВУЮ РАБОТУ

**Студенты:** Павлова А.Н, Воловикова А.А.

**Группа:** 0322

**Тема работы:** Разработка электронной картотеки

Исходные данные:

Разработать программу, позволяющую выполнять различные операции над базой данных, представленной в виде линейного списка (тема базы данных и набор операций есть в своём варианте задания).

Курсовая работа «собирается» студентом из функций, объединённых с помощью меню в головной программе, выполняющей обработку связанного линейного списка.

В виде отдельных пользовательских функций оформляются части программы, реализующие операции:

* создание списка (выделение памяти, создание и заполнение вводимыми с клавиатуры данными элементов списка);
* сохранение введённой информации в заданном пользователем файле;
* восстановление списка (заполнение его информацией, считываемой из файла, поиск элемента по признаку (признак - одно из полей структуры));
* сортировка найденных элементов и вывод информации о них на экран;
* корректировка полей записи выбранного элемента (идентификация элемента по номеру в выводимом на экран перечне (по номеру указателя на элемент));
* удаление выбранного элемента (одного из найденных по признаку);
* вставка нового элемента (после/перед выбранным).

Курсовая работа оформляется в виде пояснительной записки, в которой отражены все полученные результаты разработки.

Хранение данных программы (базы данных, картотеки, справочника, информационной системы и т.п.) необходимо осуществить в форме линейного списка. Для организации линейного списка запрещается использовать готовые контейнерные типы данных. Всю структуру и связи линейного списка нужно прописать в программе вручную.

Ввод исходных данных осуществляется через внешний файл. Вывод полученных новых данных должно осуществляться в новый файл. Ввод других необходимых данных для выполнения программы может осуществляться как через отдельный файл, так и напрямую с консоли. При этом, необходимо вывести на консоль весь ход выполнения программы (комментарии по действиям программы, копии вводимых и выводимых данных).

При разработке программы необходимо учитывать и корректно обрабатывать исключительные ситуации, которые могут возникнуть при открытии на чтение или запись несуществующего файла, а также на возможные некорректные значения некоторых важных данных.

Для выполнения некоторых заданий может понадобиться осуществить доступ к отдельному элементу списка. В этом случае доступ должен осуществляться с помощью указателей.

Каждая операция из меню программы должна осуществляться в отдельных пользовательских функциях.

Для упрощения проектирования программы рекомендуется представить проводимые алгоритмические операции в графическом виде.

**Вариант № 5.** В поликлинике работают врачи (ФИО) различных специальностей, имеющие разную квалификацию. Каждый врач осуществляет приём по своему заданному графику приёма пациентов. Каждый пациент (ФИО) может обращаться в поликлинику несколько раз к разным специалистам. Необходимо организовать запись пациентов на приём к врачам поликлиники по дням недели, каждый пациент в один день может быть записан не более чем к D врачам, а также отказаться от имеющейся записи.

При выполнении записи необходимо иметь информацию о:

* занятости конкретного врача в течение недели;
* занятости врачей конкретной специальности;
* когда и куда пациент уже записан.

**Содержание пояснительной записки**

«Содержание», «Введение», «Внешние форматы хранения данных», «Внутренние форматы хранения данных», «Описание функций», «Описание интерфейса пользователя», «Описание алгоритма работы программы», «Пример работы программы», «Заключение», «Список использованных источников» и «Приложение».

Предполагаемый объем пояснительной записки:

Не менее 30 страниц

|  |  |
| --- | --- |
| Дата выдачи задания: | 20.09.21 |
| Дата сдачи курсовой работы: | 27.12.21 |
| Дата защиты курсовой работы: | 27.12.21 |

|  |  |  |
| --- | --- | --- |
| Студенты |  | Воловикова А.А. |
|  |  | Павлова А.Н. |
| Преподаватель |  | Кузьмин С.А. |

АННОТАЦИЯ

В данной курсовой работе описана реализация программы, выполняющая функцию электронной регистратуры для медицинского учреждения. Программа будет взаимодействовать с пользователем с помощью интерактивного меню. Программа сохраняет данные о врачах, пациентах и записях на приём. Между запусками программы данные сохраняются в текстовых файлах.

В конечном результате пользователь получит программу, функционально отвечающую требованиям данной работы.

ANNOTATION

This course work describes the implementation of a program that performs the function of an electronic registry for a medical institution. The program will interact with the user using an interactive menu. The program saves data about doctors, patients and receptions. Between program launches, data is saved in text files.

In the end result, the user will receive a program that functionally meets the requirements of this work.

СОДЕРЖАНИЕ

[Введение 7](#_Toc62415987)

[Внешние форматы хранения данных 8](#_Toc62415988)

[Внутренние форматы хранения данных 10](#_Toc62415989)

[Описание функций 16](#_Toc62415990)

[Описание интерфейса пользователя 33](#_Toc62415991)

[Описание алгоритма работы программы 38](#_Toc62415992)

[Пример работы программы 85](#_Toc62415993)

[Заключение 116](#_Toc62415994)

[Список используемых источников 117](#_Toc62415995)

[Приложение 1. Листинг программного кода 118](#_Toc62415996)

# Введение

Цель работы:

Приобретение навыков разработки и отладки многомодульных программ на языке C++. Разработка программы, позволяющей создать, обработать, вывести и удалить базу данных на основе линейных списков.

В ходе выполнения работы мы воспользуемся всеми знаниями, полученными при выполнении лабораторных работ.

Содержание пояснительной записки к курсовой работе соответствует стандартным требованиям к пояснительным документам для программного продукта.

Задачи работы включают выполнения следующих этапов:

1. Создание списка (выделение памяти, создание и заполнение вводимыми с клавиатуры данными элементов списка);
2. Сохранение введённой информации в заданном пользователем файле;
3. Восстановление списка (заполнение его информацией, считываемой из файла, поиск элемента по признаку (признак одно из полей структуры));
4. Корректировка полей записи выбранного элемента (идентификация элемента по номеру в выводимом на экран перечне (по номеру указателя на элемент));
5. Удаление выбранного элемента (одного из найденных по признаку);
6. Вставка нового элемента (после/перед выбранным).

Программа разрабатывалась на языке программирования C++ [1] в среде разработки Visual Studio [2].

# 1. Внешние форматы хранения данных

Между запусками программы, данные хранятся в текстовых файлах (.txt). В каждой строке в файле присутствуют все данные по одному из элементов списка. Каждое значение соответствующего поля записи заканчиваетсся символом «;» (чтобы отделять его от значения другого поля).

Файловая структура:

| staff.txt (данные врачей);

| patients.txt (данные пациентов);

| receptions.txt (данные о записях на приём).

Файловая структура *staff.txt* (рис. 1):

* ID;
* ФИО;
* специальность;
* ID категории (0 – вторая, 1 – первая, 2 – высшая);
* продолжительность приёма пациента (в минутах);
* график приёма в формате:
  + <время начала приёма (часы:минуты)>-<время окончания приёма (часы:минуты)>:<день недели>;. Первый день недели - воскресенье (0). Последний день недели – суббота (6).

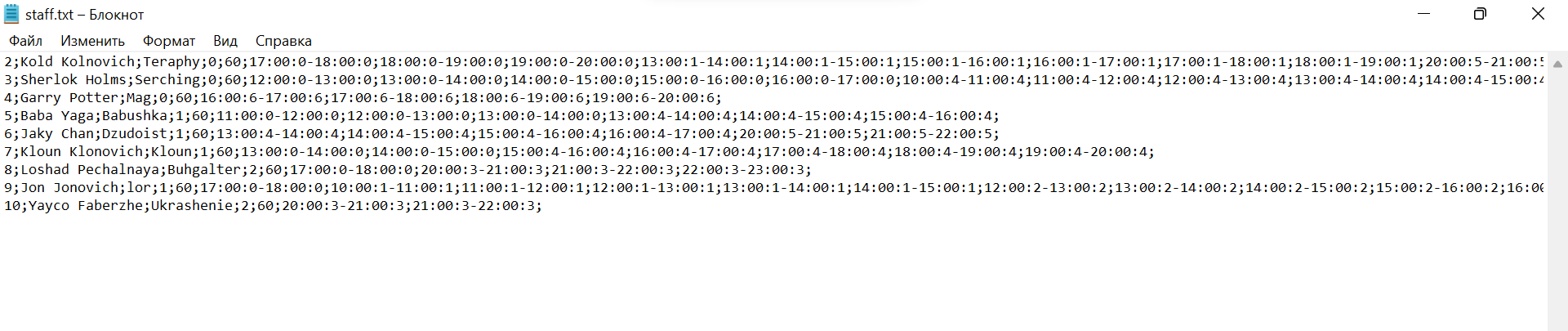


Рисунок 1 – Файловая структура staff.txt

Файловая структура *patients.txt* (рис. 2):

* ID;
* ФИО.

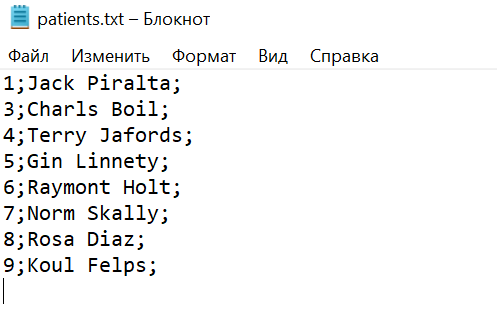


Рисунок 2 – Файловая структура patients.txt

Файловая структура *receptions.txt* (рис. 3):

* ID записи;
* ID врача;
* ID пациента;
* продолжительность приёма (в минутах);
* время начала приёма. Записывается в формате: «часы:минуты:день недели»;
* заключение врача по результатам приёма (строковое значение). В момент создания записи – пустая строка.

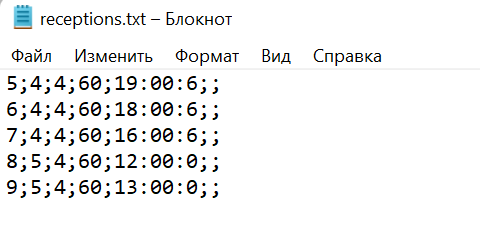


Рисунок 3 – Файловая структура reception.txt

# 2. Внутренние форматы хранения данных

В данной курсовой работе мы сделали упор на использование типов данных *struct* [1].

std::tm

Структура стандартной библиотеки [3]. Содержит календарную дату и время с разбивкой на компоненты. Включает в себя девять членов типа *int* (в любом порядке), а именно:

struct tm

{

int tm\_sec; - секунды после минуты - [0, 60], включая дополнительную секунду

int tm\_min; - минуты после часа - [0, 59]

int tm\_hour; - часы с полуночи - [0, 23]

int tm\_mday; - день месяца - [1, 31]

int tm\_mon; - месяцев с января - [0, 11]

int tm\_year; - лет с 1900 г.

int tm\_wday; - дней с воскресенья - [0, 6]

int tm\_yday; - дни с 1 января - [0, 365]

int tm\_isdst; - флаг летнего времени

};

WorkDay

Структура данных для реализации графика работы врачей.

struct WorkDay {

bool is\_day\_off = false;

tm \*start\_at = nullptr;

tm \*end\_at = nullptr;

};

Описание полей:

* bool is\_day\_off – принимает значение *true* для выходного дня;
* tm \*start\_at – указатель на структуру *std::tm*. Время начала приёма;
* tm \*end\_at – указатель на структуру std::tm. Время окончания приёма.

Doctor

Структура данных хранит информацию о врачах (в виде линейного односвязного списка [4]):

struct Doctor {

int id = NULL;

string full\_name;

string specialty;

int category\_id;

Doctor \*next = nullptr;

WorkDay \*workdays;

int duration = NULL;

};

Описание полей:

* int id – уникальный идентификатор;
* string full\_name – ФИО;
* string specialty – специальность;
* int category\_id – идентификатор категории (0 – вторая, 1 – первая, 2 – высшая);
* Doctor \*next – указатель на следующий элемент списка;
* WorkDay \*workdays – указатель на массив элементов WorkDay, график работы;
* int duration – продолжительность приёма.

DoctorList

Структура данных реализует навигацию по элементам списка Doctor [4].

struct DoctorList {

Doctor \*root = nullptr;

Doctor \*curr = nullptr;

};

Описание полей:

* Doctor \*root – указатель на корневой элемент списка;
* Doctor \*curr – указатель на текущий элемент списка.

Patient

Структура данных хранит информацию о пациентах (в виде линейного односвязного списка [4]):

struct Patient {

int id = NULL;

string full\_name;

Patient \*next = nullptr;

};

Описание полей:

* int id – уникальный идентификатор;
* string full\_name – ФИО;
* Patient \*next – указатель на следующий элемент списка.

PatientList

Структура данных реализует навигацию по элементам списка Patient [4]:

struct PatientList {

Patient \*root = nullptr;

Patient \*curr = nullptr;

};

Описание полей:

* Patient \*root – указатель на корневой элемент списка;
* Patient \*curr – указатель на текущий элемент списка.

Reception

Структура данных хранит информацию о записях на приём (в виде линейного односвязного списка [4]):

struct Reception {

int id = NULL;

staff::Doctor \*doctor = nullptr;

patient::Patient \*patient = nullptr;

tm \*visit\_at = nullptr;

int duration = NULL;

string conclusion;

Reception \*next = nullptr;

};

Описание полей:

* int id – уникальный идентификатор;
* staff::Doctor \*doctor – указатель на врача;
* patient::Patient \*patient – указатель на пациента;
* tm \*visit\_at – указатель на структуру std::tm. Время визита;
* int duration – продолжительность приёма;
* string conclusion – заключение врача по результатам приёма;
* Reception \*next – указатель на следующий элемент списка.

ReceptionList

Структура данных реализует навигацию по элементам списка Reception [4]:

struct ReceptionList {

Reception \*root = nullptr;

Reception \*curr = nullptr;

};

Описание полей:

* Reception \*root – указатель на корневой элемент списка;
* Reception \*curr – указатель на текущий элемент списка.

На рис. 4 показан общий рисунок по всем структурам данных.

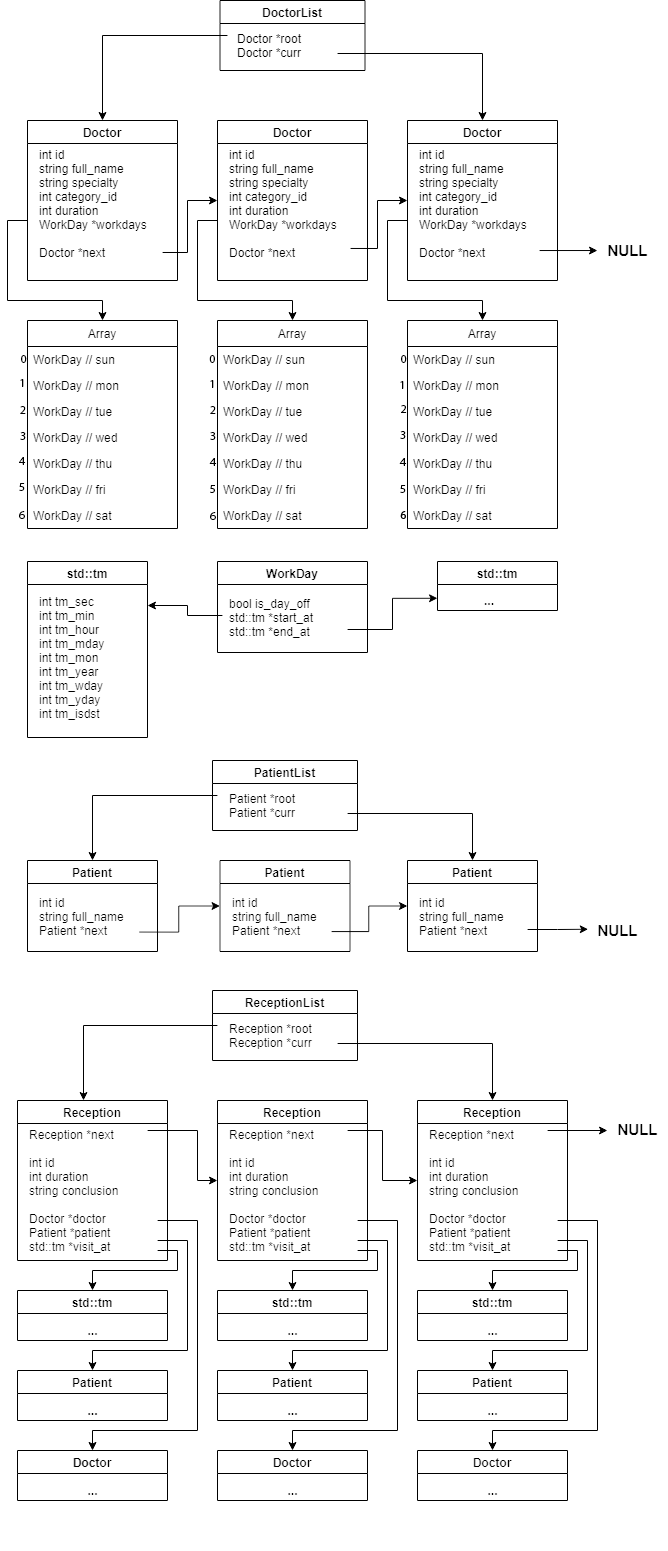


Рисунок 4 – Диаграмма структуры данных программы

# 3. Описание функций

Модуль utils.h

**clearScreen**

|  |  |
| --- | --- |
| Назначение | Очищает консоль |
| Параметры | нет |
| Возвращаемое значение | void |

**readAnswerFromConsole**

|  |  |
| --- | --- |
| Назначение | Читает значение y/n из консоли |
| Параметры | const char title[] - заголовок, выводится в консоли |
| Возвращаемое значение | Bool |

**readNumberFromConsole**

|  |  |
| --- | --- |
| Назначение | Читает значение числа из консоли |
| Параметры | const char title[] - заголовок, выводится в консоли |
| Возвращаемое значение | int |

**readWordFromConsole**

|  |  |
| --- | --- |
| Назначение | Читает строку без пробелов из консоли |
| Параметры | const char title[] - заголовок, выводится в консоли |
| Возвращаемое значение | std::string |

**readLineFromConsole**

|  |  |
| --- | --- |
| Назначение | Читает строку из консоли |
| Параметры | const char title[] - заголовок, выводится в консоли |
| Возвращаемое значение | std::string |

**getWeekDayName**

|  |  |
| --- | --- |
| Назначение | Возвращает название дня недели по индексу |
| Параметры | const char title[] - заголовок, выводится в консоли  int index - от 0 до 6, где 0 – воскресенье  bool abbreviation = false - вернуть сокращение? |
| Возвращаемое значение | std::string |

**getCurrDateTime**

|  |  |
| --- | --- |
| Назначение | Возвращает указатель на значение даты и времени текущего дня |
| Параметры | Нет |
| Возвращаемое значение | std::tm \* |

**readTimeFromConsole**

|  |  |
| --- | --- |
| Назначение | Считывает значение времени из консоли |
| Параметры | const char fieldName[] - наименование поля для вывода в консоль |
| Возвращаемое значение | std::tm \* |

**readLineFromConsoleTo**

|  |  |
| --- | --- |
| Назначение | Читает строку из консоли и проверяет корректность значения |
| Параметры | const char fieldName[] - наименование поля для вывода в консоль  std::string &field - ссылка на поле, куда запишется значение |
| Возвращаемое значение | bool |

**readNumberFromConsoleTo**

|  |  |
| --- | --- |
| Назначение | Читает число из консоли и проверяет корректность значения |
| Параметры | const char fieldName[] - наименование поля для вывода в консоль  int &field - ссылка на поле, куда запишется значение  int min - минимальное допустимое значение  int max - максимальное допустимое значение |
| Возвращаемое значение | bool |

**strToLower**

|  |  |
| --- | --- |
| Назначение | Приводит строку к нижнему регистру |
| Параметры | std::string &s - ссылка на изменяемую строку |
| Возвращаемое значение | std::string |

**isMenuMessage**

|  |  |
| --- | --- |
| Назначение | Проверяет наличие сообщения для вывода в меню |
| Параметры | нет |
| Возвращаемое значение | bool |

**setMenuMessage**

|  |  |
| --- | --- |
| Назначение | Задает сообще-ние для вывода в меню |
| Параметры | const char title[] - строка, которая будет зафиксирована для вывода в блоке результата |
| Возвращаемое значение | void |

**clearMenuMessage**

|  |  |
| --- | --- |
| Назначение | Очищает переменную содержащее сообщение |
| Параметры | нет |
| Возвращаемое значение | void |

**printMenuMessage**

|  |  |
| --- | --- |
| Назначение | Выводит сообщение в меню |
| Параметры | нет |
| Возвращаемое значение | void |

**parseStorageStringToTime**

|  |  |
| --- | --- |
| Назначение | Форматирует подстроку из файла хранилища во время начала и окончания приёма |
| Параметры | std::string &storageStr - подстрока из файла-хранилища  std::tm &start - ссылка на Doctor->start\_at  std::tm &end - ссылка на Doctor->end\_at |
| Возвращаемое значение | void |

**parseStorageStringDateTime**

|  |  |
| --- | --- |
| Назначение | Форматирует подстроку из файла хранилища в дату и время визита |
| Параметры | std::string &storageStr - подстрока из файла-хранилища  std::tm &dt - ссылка на Reception->visit\_at |
| Возвращаемое значение | void |

**reprTimePair**

|  |  |
| --- | --- |
| Назначение | Возвращает строковое представление времени начала и окончания приёма |
| Параметры | std::tm \*start - указатель на время начала приёма врача  std::tm \*end - указатель на время окончания приёма врача |
| Возвращаемое значение | std::string |

**reprDateTime**

|  |  |
| --- | --- |
| Назначение | Возвращает строковое представление даты и времени визита |
| Параметры | std::tm \*dt - указатель на дату и время визита по записи на приём |
| Возвращаемое значение | std::string |

**isCorrectTimePair**

|  |  |
| --- | --- |
| Назначение | Проверяет что пара времени начала и окончания приёма корректны |
| Параметры | std::tm \*start - указатель на время начала приёма  std::tm \*end - указатель на время конца приёма  int duration - продолжительность приёма в минутах |
| Возвращаемое значение | bool |

**isDTGreaterThan**

|  |  |
| --- | --- |
| Назначение | Проверяет что пара время **a** больше времени **b** |
| Параметры | std::tm \*a - указатель на дату и время  std::tm \*b - указатель на дату и время |
| Возвращаемое значение | bool |

**isSameDay**

|  |  |
| --- | --- |
| Назначение | Проверяет что пара времени **a** и **b** относятся к одному дню |
| Параметры | std::tm \*a - указатель на дату и время  std::tm \*b - указатель на дату и время |
| Возвращаемое значение | bool |

Модуль staff.h

**addItem**

|  |  |
| --- | --- |
| Назначение | Добавление элемента Doctor в список |
| Параметры | нет |
| Возвращаемое значение | void |

**editItem**

|  |  |
| --- | --- |
| Назначение | Редактирование элемента Doctor |
| Параметры | нет |
| Возвращаемое значение | void |

**deleteItem**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Doctor |
| Параметры | нет |
| Возвращаемое значение | void |

**deleteItemWeKnowID**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Doctor |
| Параметры | Int staffID - ID врача |
| Возвращаемое значение | void |

**findItems**

|  |  |
| --- | --- |
| Назначение | Выполняет поиск по элементам Doctor |
| Параметры | нет |
| Возвращаемое значение | void |

**printList**

|  |  |
| --- | --- |
| Назначение | Вывод в консоль всех элементов Doctor из списка |
| Параметры | нет |
| Возвращаемое значение | void |

**setSort**

|  |  |
| --- | --- |
| Назначение | Установить порядок вывода элементов Doctor для функции printList |
| Параметры | int sortType - id типа сортировки |
| Возвращаемое значение | void |

**loadData**

|  |  |
| --- | --- |
| Назначение | Загружает данные из файла хранилища и инициализирует список DoctorList |
| Параметры | Нет |
| Возвращаемое значение | void |

**saveData**

|  |  |
| --- | --- |
| Назначение | Сохраняет данные из списка DoctorList в файл хранилище |
| Параметры | Нет |
| Возвращаемое значение | void |

**clearMemory**

|  |  |
| --- | --- |
| Назначение | Удаляет все элементы Doctor из динамической памяти |
| Параметры | Нет |
| Возвращаемое значение | void |

**deleteItemByID**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Doctor по ID |
| Параметры | int id - ID элемента Doctor |
| Возвращаемое значение | bool |

**sortList**

|  |  |
| --- | --- |
| Назначение | Сортирует элементы Doctor в списке согласно установленному типу сортировки |
| Параметры | нет |
| Возвращаемое значение | void |

**isNeedSwapBySortType**

|  |  |
| --- | --- |
| Назначение | Определяет необходимость перестановки элементов Doctor согласно типу сортировки |
| Параметры | Doctor \*curr - указатель на текущий элемент Doctor  Doctor \*next - указатель на следующий элемент Doctor |
| Возвращаемое значение | bool |

**getLastID**

|  |  |
| --- | --- |
| Назначение | Возвращает последний ID из списка DoctorList |
| Параметры | Нет |
| Возвращаемое значение | Int |

**getLastNode**

|  |  |
| --- | --- |
| Назначение | Возвращает последний элемент из списка DoctorList |
| Параметры | Нет |
| Возвращаемое значение | Doctor \* |

**findItemByID**

|  |  |
| --- | --- |
| Назначение | Ищет элемент Doctor по ID в списке |
| Параметры | Нет |
| Возвращаемое значение | bool |

**getItemByID**

|  |  |
| --- | --- |
| Назначение | Возвращает элемент Doctor по ID |
| Параметры | int id - ID элемента Doctor |
| Возвращаемое значение | Doctor \* |

**getListRepresentation**

|  |  |
| --- | --- |
| Назначение | Возвращает строковое представление всех элементов Doctor из списка |
| Параметры | нет |
| Возвращаемое значение | std::string |

**getItemRepresentation**

|  |  |
| --- | --- |
| Назначение | Возвращает строковое представление элемента Doctor |
| Параметры | Doctor \*node - указатель на элемент Doctor |
| Возвращаемое значение | std::string |

**isNotOnVacation**

|  |  |
| --- | --- |
| Назначение | Определяет находится ли врач в отпуске |
| Параметры | Doctor \*node node - указатель на элемент Doctor |
| Возвращаемое значение | bool |

Модуль patients.h

**addItem**

|  |  |
| --- | --- |
| Назначение | Добавление элемента Patient в список |
| Параметры | нет |
| Возвращаемое значение | void |

**editItem**

|  |  |
| --- | --- |
| Назначение | Редактирование элемента Patient |
| Параметры | нет |
| Возвращаемое значение | void |

**deleteItem**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Patient |
| Параметры | нет |
| Возвращаемое значение | void |

**findItems**

|  |  |
| --- | --- |
| Назначение | Выполняет поиск по элементам Patient |
| Параметры | нет |
| Возвращаемое значение | void |

**printList**

|  |  |
| --- | --- |
| Назначение | Вывод в консоль всех элементов Patient из списка |
| Параметры | нет |
| Возвращаемое значение | void |

**setSort**

|  |  |
| --- | --- |
| Назначение | Установить порядок вывода элементов Patient для функции printList |
| Параметры | int sortType - id типа сортировки |
| Возвращаемое значение | void |

**loadData**

|  |  |
| --- | --- |
| Назначение | Загружает данные из файла хранилища и инициализирует список PatientList |
| Параметры | Нет |
| Возвращаемое значение | void |

**saveData**

|  |  |
| --- | --- |
| Назначение | Сохраняет данные из списка PatientList в файл хранилище |
| Параметры | Нет |
| Возвращаемое значение | void |

**clearMemory**

|  |  |
| --- | --- |
| Назначение | Удаляет все элементы Patient из динамической памяти |
| Параметры | Нет |
| Возвращаемое значение | void |

**deleteItemByID**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Patient по ID |
| Параметры | int id - ID элемента Patient |
| Возвращаемое значение | bool |

**sortList**

|  |  |
| --- | --- |
| Назначение | Сортирует элементы Patient в списке согласно установленному типу сортировки |
| Параметры | нет |
| Возвращаемое значение | void |

**isNeedSwapBySortType**

|  |  |
| --- | --- |
| Назначение | Определяет необходимость перестановки элементов Patient согласно типу сортировки |
| Параметры | Patient \*curr - указатель на текущий элемент Patient  Patient \*next - указатель на следующий элемент Patient |
| Возвращаемое значение | bool |

**getLastID**

|  |  |
| --- | --- |
| Назначение | Возвращает последний ID из списка PatientList |
| Параметры | Нет |
| Возвращаемое значение | Int |

**getLastNode**

|  |  |
| --- | --- |
| Назначение | Возвращает последний элемент из списка PatientList |
| Параметры | Нет |
| Возвращаемое значение | Patient \* |

**findItemByID**

|  |  |
| --- | --- |
| Назначение | Ищет элемент Patient по ID в списке |
| Параметры | Нет |
| Возвращаемое значение | bool |

**getItemByID**

|  |  |
| --- | --- |
| Назначение | Возвращает элемент Patient по ID |
| Параметры | int id - ID элемента Patient |
| Возвращаемое значение | Patient \* |

**getListRepresentation**

|  |  |
| --- | --- |
| Назначение | Возвращает строковое представление всех элементов Patient из списка |
| Параметры | нет |
| Возвращаемое значение | std::string |

**getItemRepresentation**

|  |  |
| --- | --- |
| Назначение | Возвращает строковое представление элемента Patient |
| Параметры | Patient \*node - указатель на элемент Patient |
| Возвращаемое значение | std::string |

**deleteItemKnowID**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Patient |
| Параметры | Int PatID - ID пациента |
| Возвращаемое значение | void |

Модуль receptions.h

**addItem**

|  |  |
| --- | --- |
| Назначение | Добавление элемента Reception в список |
| Параметры | нет |
| Возвращаемое значение | void |

**editItem**

|  |  |
| --- | --- |
| Назначение | Редактирование элемента Reception |
| Параметры | нет |
| Возвращаемое значение | void |

**setConclusion**

|  |  |
| --- | --- |
| Назначение | Задать заключение для элемента Reception |
| Параметры | нет |
| Возвращаемое значение | void |

**deleteItem**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Reception |
| Параметры | нет |
| Возвращаемое значение | int |

**findItems**

|  |  |
| --- | --- |
| Назначение | Выполняет поиск по элементам Reception |
| Параметры | нет |
| Возвращаемое значение | void |

**printList**

|  |  |
| --- | --- |
| Назначение | Вывод в консоль элементов Reception из списка согласно типу вывода |
| Параметры | int printType - id типа вывода на экран |
| Возвращаемое значение | void |

**setSort**

|  |  |
| --- | --- |
| Назначение | Установить порядок вывода элементов Reception для функции printList |
| Параметры | int sortType - id типа сортировки |
| Возвращаемое значение | void |

**loadData**

|  |  |
| --- | --- |
| Назначение | Загружает данные из файла хранилища и инициализирует список ReceptionList |
| Параметры | Нет |
| Возвращаемое значение | void |

**saveData**

|  |  |
| --- | --- |
| Назначение | Сохраняет данные из списка ReceptionList в файл хранилище |
| Параметры | Нет |
| Возвращаемое значение | void |

**clearMemory**

|  |  |
| --- | --- |
| Назначение | Удаляет все элементы Reception из динамической памяти |
| Параметры | Нет |
| Возвращаемое значение | void |

**deleteItemByID**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Reception по ID |
| Параметры | int id - ID элемента Reception |
| Возвращаемое значение | bool |

**deleteItemByDoctor**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Reception при удалении Doctor |
| Параметры | нет |
| Возвращаемое значение | void |

**deleteItemByPatient**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Reception при удалении Patient |
| Параметры | нет |
| Возвращаемое значение | void |

**sortList**

|  |  |
| --- | --- |
| Назначение | Сортирует элементы Reception в списке согласно установленному типу сортировки |
| Параметры | нет |
| Возвращаемое значение | void |

**isNeedSwapBySortType**

|  |  |
| --- | --- |
| Назначение | Определяет необходимость перестановки элементов Reception согласно типу сортировки |
| Параметры | Reception \*curr - указатель на текущий элемент Reception  Reception \*next - указатель на следующий элемент Reception |
| Возвращаемое значение | bool |

**getLastID**

|  |  |
| --- | --- |
| Назначение | Возвращает последний ID из списка  ReceptionList |
| Параметры | Нет |
| Возвращаемое значение | Int |

**getLastNode**

|  |  |
| --- | --- |
| Назначение | Возвращает последний элемент из списка ReceptionList |
| Параметры | Нет |
| Возвращаемое значение | Patient \* |

**findItemByID**

|  |  |
| --- | --- |
| Назначение | Ищет элемент Reception по ID в списке |
| Параметры | Нет |
| Возвращаемое значение | bool |

**getItemByID**

|  |  |
| --- | --- |
| Назначение | Возвращает элемент Reception по ID |
| Параметры | int id - ID элемента Reception |
| Возвращаемое значение | Patient \* |

**getListRepresentation**

|  |  |
| --- | --- |
| Назначение | Возвращает строковое представление элементов Reception из списка согласно типу вывода |
| Параметры | int printType - ID типа вывода на экран |
| Возвращаемое значение | std::string |

**getItemRepresentation**

|  |  |
| --- | --- |
| Назначение | Возвращает строковое представление элемента Reception |
| Параметры | Reception \*node - указатель на элемент Reception |
| Возвращаемое значение | std::string |

**getAvailableDateTime**

|  |  |
| --- | --- |
| Назначение | Возвращает ближайшее доступное время визита для выбранного дня недели |
| Параметры | Reception \*node - указатель на элемент Reception  std::tm \*day - указатель на день, для которого рассчитываем доступное время  staff::WorkDay \* wd - указатель на рабочий день недели врача |
| Возвращаемое значение | std:: tm \* |

**readAvailableDateFromConsole**

|  |  |
| --- | --- |
| Назначение | Читает из консоли время визита |
| Параметры | Reception &node - ссылка на элемент в который буде записано время визита |
| Возвращаемое значение | bool |

**deleteItemByPatientID**

|  |  |
| --- | --- |
| Назначение | Удаление элемента Receptions по Patient |
| Параметры | Int ID - ID пациента |
| Возвращаемое значение | int |

Модуль medical.h

**loopMenu**

|  |  |
| --- | --- |
| Назначение | Вывод меню в цикле до выхода из программы |
| Параметры | нет |
| Возвращаемое значение | void |

**printPage**

|  |  |
| --- | --- |
| Назначение | Вывод содержания текущей страницы в консоль |
| Параметры | нет |
| Возвращаемое значение | void |

**displayMenu**

|  |  |
| --- | --- |
| Назначение | Вывод пунктов меню |
| Параметры | нет |
| Возвращаемое значение | void |

**pushMenuItem**

|  |  |
| --- | --- |
| Назначение | Добавление пункта меню в коллекцию |
| Параметры | const char \*name - название пункта меню  int code - ID пункта меню  std::string \*menu - указатель на объект меню |
| Возвращаемое значение | void |

**readUserMenuChoice**

|  |  |
| --- | --- |
| Назначение | Чтение пункта меню из консоли |
| Параметры | нет |
| Возвращаемое значение | void |

**doUserChoiceAction**

|  |  |
| --- | --- |
| Назначение | Выполнение действия меню в соответствии с выбранным пунктом |
| Параметры | нет |
| Возвращаемое значение | void |

**loadData**

|  |  |
| --- | --- |
| Назначение | Загрузка данных из текстовых файлов |
| Параметры | нет |
| Возвращаемое значение | void |

**clearMemory**

|  |  |
| --- | --- |
| Назначение | Удаляет все элементы из динамической памяти |
| Параметры | Нет |
| Возвращаемое значение | void |

# 4. ОПИСАНИЕ интерфейса пользователя

При запуске программы пользователю будет доступно меню для навигации и выполнения операций (рис. 5).

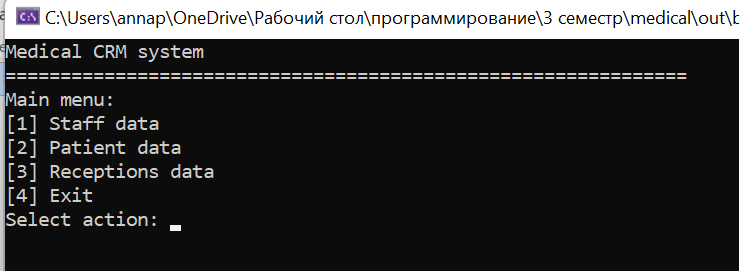


Рисунок 5 – Главное меню (Main menu)

Меню включает в себя следующие пункты:

1. Данные сотрудников: «Staff data» (рис. 6);

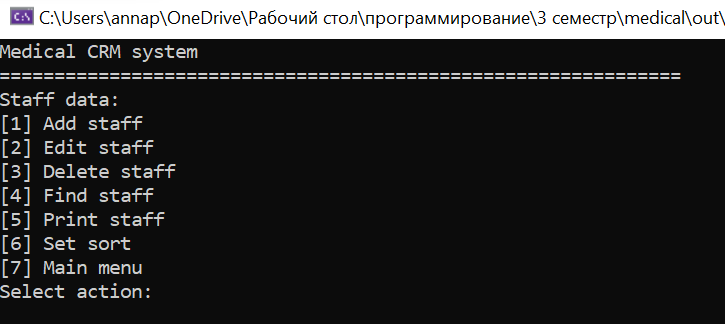


Рисунок 6 – Данные сотрудников (Staff data)

* 1. Добавить сотрудника: «Add staff»;
     1. Ввод ФИО: «Enter full name»;
     2. Ввод специальности: «Enter specialty»;
     3. Выбор категории из доступных: «Enter category ID»;
     4. Ввод продолжительности времени приёма: «Enter reception duration»;
     5. Ввод графика работы по дням недели: «Enter workdays»;
  2. Редактировать сотрудника: «Edit staff»;
     1. Ввод ID сотрудника: «Enter staff ID»;
     2. Опциональное редактирование полей с помощью операций: 1.1.1‑1.1.5;
  3. Удалить сотрудника: «Delete staff»;
     1. Ввод ID сотрудника: «Enter staff ID»;
  4. Найти сотрудника: «Find staff»;
     1. Ввод подстроки: «Enter search string»;
  5. Вывести список сотрудников: «Print staff»;
  6. Задать сортировку: «Set sort» (рис. 7);

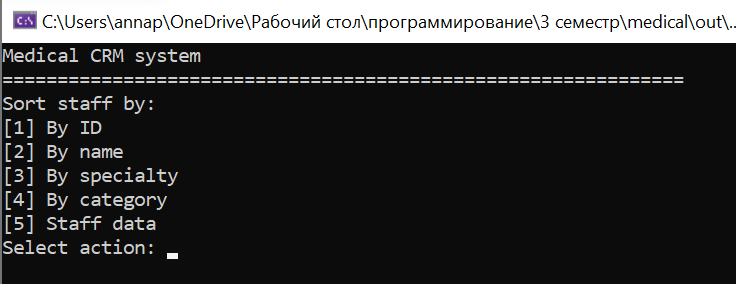


Рисунок 7 – Задать сортировку (Set sort)

* + 1. По ID: «By ID»;
    2. По ФИО: «By name»;
    3. По специальности: «By specialty»;
    4. По категории: «By category»;
    5. Вернуться в меню «Данные сотрудников»: «Staff data»;
  1. Вернуться в главное меню: «Main menu»;

1. Данные пациентов: «Patients data» (рис. 8);

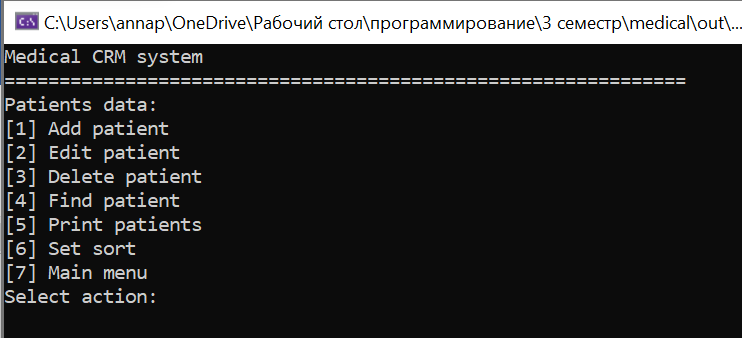


Рисунок 8 – Данные пациентов (Patients data)

* 1. Добавить пациента: «Add patient»;
     1. Ввод ФИО: «Enter full name»;
  2. Редактировать пациента: «Edit patient»;
     1. Ввод ID пациента: «Enter patient ID»;
     2. Опциональное редактирование поля ФИО с помощью операции: 2.1.1;
  3. Удалить пациента: «Delete patient»;
     1. Ввод ID пациента: «Enter patient ID»;
  4. Найти пациента: «Find patient»;
     1. Ввод подстроки: «Enter search string»;
  5. Вывести список пациентов: «Print patients»;
  6. Задать сортировку: «Set sort» (рис. 9);

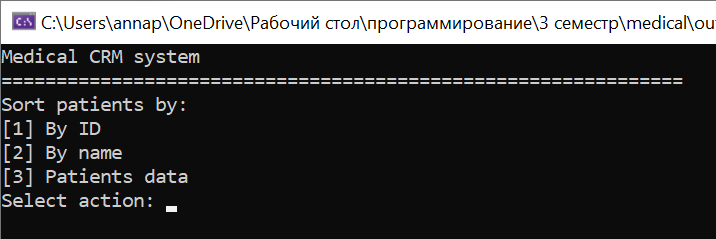


Рисунок 9 – Задать сортировку (Set sort)

* + 1. По ID: «By ID»;
    2. По ФИО: «By name»;
    3. Вернуться в меню «Данные пациентов»: «Patients data»;
  1. Вернуться в главное меню: «Main menu»;

1. Данные о записях на приём: «Receptions data» (рис. 10);

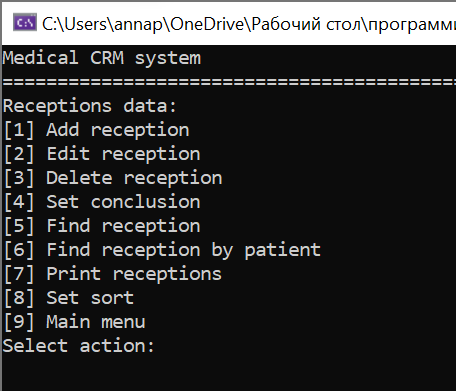


Рисунок 10 – Данные о записях на приём (Receptions data)

* 1. Добавить запись на приём: «Add reception»;
     1. Ввести место для вставки «Enter position to insert»
     2. Ввести ID врача: «Enter doctor ID»;
     3. Ввести ID пациента: «Enter patient ID»;
     4. Выбрать доступное время визита: «Enter visit time»;
  2. Редактировать запись на приём: «Edit reception»;
     1. Ввести ID записи на приём: «Enter reception ID»;
     2. Опционально отредактировать поля с помощью операций из пунктов: 3.1.1-3.1.3;
  3. Удалить запись на приём: «Delete reception»;
     1. Ввести ID записи на приём: «Enter reception ID»;
  4. Установить заключение по приёму: «Set conclusion»;
     1. Ввести ID записи на приём: «Enter reception ID»;
     2. Ввести заключение по приёму: «Enter conclusion»;
  5. Найти запись на приём: «Find reception»;
     1. Ввод подстроки: «Enter search string»;
  6. Вывести список записей: «Print receptions» (рис. 11);

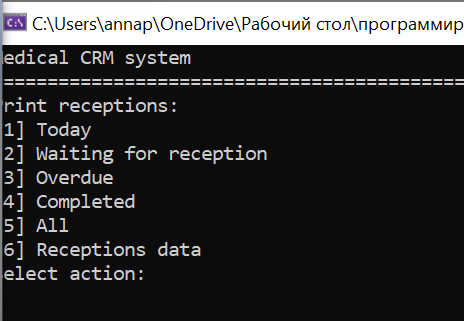


Рисунок 11 – Вывести список записей (Print receptions)

* + 1. Назначенных на сегодня: «Today»;
    2. Ожидающих выполнения: «Waiting for reception»;
    3. Просроченных: «Overdue»;
    4. Выполненных: «Completed»;
    5. Все: «All»;
    6. Вернуться в меню «Данные о записях на приём»: «Receptions data»;
  1. Задать сортировку: «Set sort» (рис. 12);

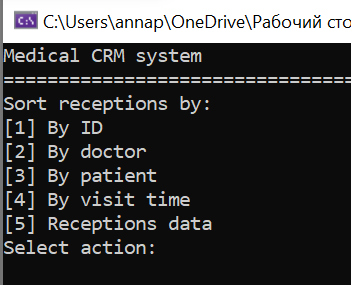


Рисунок 12 – Задать сортировку (Set sort);

* + 1. По ID: «By ID»;
    2. По врачам: «By doctor»;
    3. По пациентам: «By patient»;
    4. По времени визита: «By visit time»;
    5. Вернуться в меню «Данные о записях на приём»: «Receptions data»;
  1. Вернуться в главное меню: «Main menu»;

1. Выход: «Exit».

# ОПИСАНИЕ алгоритма работы программы

При запуске программы пользователю отобразится главное меню. Для осуществления навигации, пользователь должен указывать целочисленное значение пункта меню. Меню выводится циклично до выхода из программы.

main.cpp

**main**

mdc::loadData();

mdc::loopMenu();

mdc::clearMemory()

return 0;

**medical.cpp**

**loadData**

staff::loadData();

patient::loadData();

reception::loadData();

**void clearMemory**

staff::clearMemory();

patient::clearMemory();

reception::clearMemory();

**loopMenu**

ПОКА (curMenuCode != MC\_EXIT)

Начало цикла

printPage();

readUserMenuChoice();

doUserChoiceAction();

Конец цикла

**printPage**

ЕСЛИ (utl::isMenuMessage())

Начало

utl::printMenuMessage();

Конец

displayMenu();

**displayMenu**

string menu[MENU\_SIZE\_LIMIT];

Цикл menuCode по menuCodes

menuCode = MC\_UNDECLARED;

ЕСЛИ (curMenuCode)

Начало

равно MC\_MAIN:

pushMenuItem("Staff data", MC\_STAFF\_DATA, menu);

pushMenuItem("Patient data", MC\_PATIENT\_DATA, menu);

pushMenuItem("Receptions data", MC\_RECEPTION\_DATA, menu);

равно MC\_STAFF\_DATA:

pushMenuItem("Add staff", MC\_STAFF\_DATA\_ADD, menu);

pushMenuItem("Edit staff", MC\_STAFF\_DATA\_EDIT, menu);

pushMenuItem("Delete staff", MC\_STAFF\_DATA\_DELETE, menu);

pushMenuItem("Find staff", MC\_STAFF\_DATA\_FIND, menu);

pushMenuItem("Print staff", MC\_STAFF\_DATA\_PRINT, menu);

pushMenuItem("Set sort", MC\_STAFF\_DATA\_SET\_SORT, menu);

равно MC\_STAFF\_DATA\_SET\_SORT:

pushMenuItem("By ID", MC\_STAFF\_DATA\_SET\_SORT\_BY\_ID, menu);

pushMenuItem("By name", MC\_STAFF\_DATA\_SET\_SORT\_BY\_NAME, menu);

pushMenuItem("By specialty", MC\_STAFF\_DATA\_SET\_SORT\_BY\_SPECIALTY, menu);

pushMenuItem("By category", MC\_STAFF\_DATA\_SET\_SORT\_BY\_CATEGORY, menu);

равно MC\_PATIENT\_DATA:

pushMenuItem("Add patient", MC\_PATIENT\_DATA\_ADD, menu);

pushMenuItem("Edit patient", MC\_PATIENT\_DATA\_EDIT, menu);

pushMenuItem("Delete patient", MC\_PATIENT\_DATA\_DELETE, menu);

pushMenuItem("Find patient", MC\_PATIENT\_DATA\_FIND, menu);

pushMenuItem("Print patients", MC\_PATIENT\_DATA\_PRINT, menu);

pushMenuItem("Set sort", MC\_PATIENT\_DATA\_SET\_SORT, menu);

равно MC\_PATIENT\_DATA\_SET\_SORT:

pushMenuItem("By ID", MC\_PATIENT\_DATA\_SET\_SORT\_BY\_ID, menu);

pushMenuItem("By name", MC\_PATIENT\_DATA\_SET\_SORT\_BY\_NAME, menu);

равно MC\_RECEPTION\_DATA:

pushMenuItem("Add reception", MC\_RECEPTION\_DATA\_ADD, menu);

pushMenuItem("Edit reception", MC\_RECEPTION\_DATA\_EDIT, menu);

pushMenuItem("Delete reception", MC\_RECEPTION\_DATA\_DELETE, menu);

pushMenuItem("Set conclusion", MC\_RECEPTION\_DATA\_SET\_CONCLUSION, menu);

pushMenuItem("Find reception", MC\_RECEPTION\_DATA\_FIND, menu);

pushMenuItem("Print receptions", MC\_RECEPTION\_DATA\_PRINT, menu);

pushMenuItem("Set sort", MC\_RECEPTION\_DATA\_SET\_SORT, menu);

равно MC\_RECEPTION\_DATA\_PRINT:

pushMenuItem("Today", MC\_RECEPTION\_DATA\_PRINT\_TODAY, menu);

pushMenuItem("Waiting for reception", MC\_RECEPTION\_DATA\_PRINT\_WAITING, menu);

pushMenuItem("Overdue", MC\_RECEPTION\_DATA\_PRINT\_OVERDUE, menu);

pushMenuItem("Completed", MC\_RECEPTION\_DATA\_PRINT\_COMPLETED, menu);

pushMenuItem("All", MC\_RECEPTION\_DATA\_PRINT\_ALL, menu);

равно MC\_RECEPTION\_DATA\_SET\_SORT:

pushMenuItem("By ID", MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_ID, menu);

pushMenuItem("By doctor", MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_DOCTOR, menu);

pushMenuItem("By patient", MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_PATIENT, menu);

pushMenuItem("By visit time", MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_VISIT\_TIME, menu);

Конец

ЕСЛИ (curMenuCode)

Начало

равно MC\_MAIN:

pushMenuItem("Exit", MC\_EXIT, menu);

равно MC\_STAFF\_DATA\_SET\_SORT:

pushMenuItem("Staff data", MC\_STAFF\_DATA, menu);

равно MC\_PATIENT\_DATA\_SET\_SORT:

pushMenuItem("Patients data", MC\_PATIENT\_DATA, menu);

равно MC\_RECEPTION\_DATA\_PRINT или MC\_RECEPTION\_DATA\_SET\_SORT:

pushMenuItem("Receptions data", MC\_RECEPTION\_DATA, menu);

Конец

ИНАЧЕ:

pushMenuItem("Main menu", MC\_MAIN, menu);

Конец

Цикл по i от 0 до MENU\_SIZE\_LIMIT с шагом 1

Начало

string &menuItem = menu[i];

ЕСЛИ (menuItem.empty())

Прервать цикл

Вывод menuItem

Конец

**pushMenuItem**

Цикл по i от 0 до MENU\_SIZE\_LIMIT с шагом 1

Начало

ЕСЛИ (menu[i].empty())

Начало

menu[i] = name;

menuCodes[i] = code;

Прервать цикл

Конец

Конец

**readUserMenuChoice**

int choice = utl::readNumberFromConsole("Select action");

ЕСЛИ (choice не равно NULL)

Начало

ЕСЛИ (choice меньше или равно 0 или choice больше MENU\_SIZE\_LIMIT или menuCodes[choice - 1] равно MC\_UNDECLARED)

utl::setMenuMessage("Invalid choice!");

ИНАЧЕ

curMenuCode = menuCodes[choice - 1];

Конец

ИНАЧЕ

utl::setMenuMessage("Input error for 4 attempts. Operation is interrupted!");

**doUserChoiceAction**

ЕСЛИ (curMenuCode)

Начало

равно MC\_STAFF\_DATA\_ADD:

staff::addItem();

curMenuCode = MC\_STAFF\_DATA;

равно MC\_STAFF\_DATA\_EDIT:

staff::editItem();

curMenuCode = MC\_STAFF\_DATA;

равно MC\_STAFF\_DATA\_DELETE:

staff::deleteItem();

reception::deleteItemByDoctor();

curMenuCode = MC\_STAFF\_DATA;

равно MC\_STAFF\_DATA\_FIND:

staff::findItems();

curMenuCode = MC\_STAFF\_DATA;

равно MC\_STAFF\_DATA\_PRINT:

staff::printList();

curMenuCode = MC\_STAFF\_DATA;

равно MC\_STAFF\_DATA\_SET\_SORT\_BY\_ID:

staff::setSort(staff::LIST\_SORT\_TYPE\_ID);

curMenuCode = MC\_STAFF\_DATA;

равно MC\_STAFF\_DATA\_SET\_SORT\_BY\_NAME:

staff::setSort(staff::LIST\_SORT\_TYPE\_NAME);

curMenuCode = MC\_STAFF\_DATA;

равно MC\_STAFF\_DATA\_SET\_SORT\_BY\_SPECIALTY:

staff::setSort(staff::LIST\_SORT\_TYPE\_SPECIALTY);

curMenuCode = MC\_STAFF\_DATA;

равно MC\_STAFF\_DATA\_SET\_SORT\_BY\_CATEGORY:

staff::setSort(staff::LIST\_SORT\_TYPE\_CATEGORY);

curMenuCode = MC\_STAFF\_DATA;

равно MC\_PATIENT\_DATA\_ADD:

patient::addItem();

curMenuCode = MC\_PATIENT\_DATA;

равно MC\_PATIENT\_DATA\_EDIT:

patient::editItem();

curMenuCode = MC\_PATIENT\_DATA;

равно MC\_PATIENT\_DATA\_DELETE:

patient::deleteItem();

reception::deleteItemByPatient();

curMenuCode = MC\_PATIENT\_DATA;

равно MC\_PATIENT\_DATA\_FIND:

patient::findItems();

curMenuCode = MC\_PATIENT\_DATA;

равно MC\_PATIENT\_DATA\_PRINT:

patient::printList();

curMenuCode = MC\_PATIENT\_DATA;

равно MC\_PATIENT\_DATA\_SET\_SORT\_BY\_ID:

patient::setSort(patient::LIST\_SORT\_TYPE\_ID);

curMenuCode = MC\_PATIENT\_DATA;

равно MC\_PATIENT\_DATA\_SET\_SORT\_BY\_NAME:

patient::setSort(patient::LIST\_SORT\_TYPE\_NAME);

curMenuCode = MC\_PATIENT\_DATA;

равно MC\_RECEPTION\_DATA\_ADD:

reception::addItem();

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_EDIT:

reception::editItem();

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_SET\_CONCLUSION:

reception::setConclusion();

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_DELETE:

reception::deleteItem();

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_FIND:

reception::findItems();

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_PRINT\_TODAY:

reception::printList(reception::PRINT\_TYPE\_TODAY);

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_PRINT\_WAITING:

reception::printList(reception::PRINT\_TYPE\_WAITING);

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_PRINT\_OVERDUE:

reception::printList(reception::PRINT\_TYPE\_OVERDUE);

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_PRINT\_COMPLETED:

reception::printList(reception::PRINT\_TYPE\_COMPLETED);

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_PRINT\_ALL:

reception::printList(reception::PRINT\_TYPE\_ALL);

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_ID:

reception::setSort(reception::LIST\_SORT\_TYPE\_ID);

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_DOCTOR:

reception::setSort(reception::LIST\_SORT\_TYPE\_DOCTOR);

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_PATIENT:

reception::setSort(reception::LIST\_SORT\_TYPE\_PATIENT);

curMenuCode = MC\_RECEPTION\_DATA;

равно MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_VISIT\_TIME:

reception::setSort(reception::LIST\_SORT\_TYPE\_VISIT\_TIME);

curMenuCode = MC\_RECEPTION\_DATA;

Конец

**staff.cpp**

DoctorList \*nodeList;

int sortListIndex = LIST\_SORT\_TYPE\_ID;

**addItem:**

auto \*node = new Doctor;

int insert\_position = utl::readNumberFromConsole("Enter position to insert");

bool validFullName = utl::readLineFromConsoleTo("full name", node->full\_name);

bool validSpecialty = utl::readLineFromConsoleTo("specialty", node->specialty);

Вывод доступных категорий

bool validCategoryID = utl::readNumberFromConsoleTo("category ID", node->category\_id, 0, 2);

bool validDuration = utl::readNumberFromConsoleTo("reception duration", node->duration, MIN\_DURATION\_MIN, MAX\_DURATION\_MIN);

ЕСЛИ (validFullName и validSpecialty и validCategoryID и validDuration)

Начало

node->workdays = new WorkDay[7];

auto current\_wd = node->workdays;

Цикл по i от 0 до 7 с шагом 1

Начало цикла

bool workday = utl::readAnswerFromConsole("Is workday?");

auto \*wd = new WorkDay;

wd->is\_day\_off = не workday;

Конец цикла

ЕСЛИ (workday)

Начало

wd->start\_at = utl::readTimeFromConsole("start time");

wd->end\_at = utl::readTimeFromConsole("end time");

ЕСЛИ (не utl::isCorrectTimePair(wd->start\_at, wd->end\_at, node->duration))

Начало

delete wd->start\_at;

delete wd->end\_at;

wd->is\_day\_off = true;

Конец

node->workdays[i] = \*wd;

Конец

Doctor\* tmp = nodeList->root;

node->id = getLastID() + 1;

ЕСЛИ (tmp == nullptr)

Начало

nodeList->root = node;

nodeList->curr = node;

saveData();

utl::setMenuMessage("Element successfully created!");

Конец

ИНАЧЕ

Начало

Doctor\* tmp = nodeList->root;

Doctor\* serchid = nodeList->root;

int schet = 0;

ПОКА (tmp != nullptr)

Начало цикла

ЕСЛИ (tmp->id == insert\_position)

Начало

schet++;

node->next = tmp->next;

tmp->next = node;

break;

Конец

tmp = tmp->next;

Конец цикла

ЕСЛИ (schet==0)

Начало

delete node;

utl::setMenuMessage("Error adding element!");

Конец

ИНАЧЕ

Начало

nodeList->curr = node;

saveData();

utl::setMenuMessage("Element successfully created!");

Конец

Конец

ИНАЧЕ

Начало

utl::setMenuMessage("Error adding element!");

delete node;

Конец

**editItem**

ЕСЛИ (findItemByID())

Начало

Doctor \*node = nodeList->curr;

string buffer;

int bufferNum = NULL;

bool changed = false;

ЕСЛИ (utl::readAnswerFromConsole("Change [full name]?"))

Начало

ЕСЛИ (utl::readLineFromConsoleTo("full name", buffer))

Начало

node->full\_name = buffer;

changed = true;

Конец

buffer.clear();

Конец

ЕСЛИ (utl::readAnswerFromConsole("Change [specialty]?"))

Начало

ЕСЛИ (utl::readLineFromConsoleTo("specialty", buffer))

Начало

node->specialty = buffer;

changed = true;

Конец

buffer.clear();

Конец

ЕСЛИ (utl::readAnswerFromConsole("Change [category]?"))

Начало

Вывод доступных категорий;

ЕСЛИ (utl::readNumberFromConsoleTo("category ID", bufferNum, 0, 2))

Начало

node->category\_id = bufferNum;

changed = true;

Конец

bufferNum = NULL;

Конец

ЕСЛИ (utl::readAnswerFromConsole("Change [reception duration]?"))

Начало

ЕСЛИ (utl::readNumberFromConsoleTo("reception duration", bufferNum, MIN\_DURATION\_MIN, MAX\_DURATION\_MIN))

Начало

node->duration = bufferNum;

changed = true;

Конец

bufferNum = NULL;

Конец

ЕСЛИ (utl::readAnswerFromConsole("Change [workdays]?"))

Начало

changed = true;

Цикл по i от 0 до 7 с шагом 1

Начало цикла

Вывод utl::getWeekDayName(i)

bool workday = utl::readAnswerFromConsole("Is workday?");

auto \*wd = &(node->workdays[i]);

wd->is\_day\_off = не workday;

ЕСЛИ (workday)

Начало

wd->start\_at = utl::readTimeFromConsole("start time");

wd->end\_at = utl::readTimeFromConsole("end time");

ЕСЛИ (не utl::isCorrectTimePair(wd->start\_at, wd->end\_at, node->duration))

Начало

delete wd->start\_at;

delete wd->end\_at;

wd->is\_day\_off = true;

Конец

Конец

node->workdays[i] = \*wd;

Конец цикла

ЕСЛИ (changed)

Начало

saveData();

utl::setMenuMessage("Element successfully updated!");

Конец

ИНАЧЕ

utl::setMenuMessage("Nothing to change!");

Конец

ИНАЧЕ

utl::setMenuMessage("Element does not found!");

Конец

**deleteItem**

int staffID = utl::readNumberFromConsole("Enter staff ID");

ЕСЛИ (staffID не равно NULL)

Начало

ЕСЛИ (deleteItemByID(staffID))

utl::setMenuMessage("Doctor successfully deleted!");

ИНАЧЕ

utl::setMenuMessage("Doctor was not found!");

Конец

ИНАЧЕ

utl::setMenuMessage("Incorrect ID!");

**deleteItemWeKnowID**

utl::clearScreen();

ЕСЛИ (staffID != NULL)

Начало

ЕСЛИ (deleteItemByID(staffID))

Начало

utl::setMenuMessage("Doctor successfully deleted!");

Конец

ИНАЧЕ

utl::setMenuMessage("Doctor was not found!");

Конец

ИНАЧЕ

utl::setMenuMessage("Incorrect ID!");

**findItems**

string search = utl::readLineFromConsole("Enter search string");

search = utl::strToLower(search);

Doctor \*node = nodeList->root;

string buffer;

ПОКА (node не равно nullptr)

Начало

string representation = getItemRepresentation(node);

string representationLower = utl::strToLower(representation);

string::size\_type matchPos = representationLower.find(search);

ЕСЛИ (matchPos не равно string::npos)

buffer.append(representation);

node = node->next;

Конец

ЕСЛИ (buffer.empty())

utl::setMenuMessage("Matches not found!");

ИНАЧЕ

utl::setMenuMessage(buffer.c\_str());

**printList**

string buffer = getListRepresentation();

utl::setMenuMessage(buffer.c\_str());

**setSort**

sortListIndex = sortType;

sortList();

string message = "Set sorting by " + LIST\_SORT\_TYPES[sortListIndex];

utl::setMenuMessage(message.c\_str());

**loadData**

nodeList = new DoctorList;

ifstream storage(STORAGE\_FILEPATH);

ЕСЛИ (storage)

Начало

string line;

Doctor \*node, \*prev = nullptr;

ПОКА (getline(storage, line))

Начало цикла

ЕСЛИ (не line.empty())

Начало

node = new Doctor;

auto wdList = new WorkDay[7];

short cell\_counter = 0;

string::size\_type lastSeparatorPosition = 0;

string::size\_type currentSeparatorPosition = line.find(utl::STORAGE\_SEPARATOR,

lastSeparatorPosition);

ПОКА (currentSeparatorPosition не равно string::npos

string value = line.substr(lastSeparatorPosition, currentSeparatorPosition - lastSeparatorPosition);)

Начало цикла

ЕСЛИ (cell\_counter)

Начало

равно 0:

node->id = stoi(value);

равно 1:

node->full\_name = value;

равно 2:

node->specialty = value;

равно 3:

node->category\_id = stoi(value);

равно 4:

node->duration = stoi(value);

Конец

ИНАЧЕ

Начало

auto \*wd = new WorkDay;

wd->is\_day\_off = value.empty();

ЕСЛИ (не wd->is\_day\_off)

Начало

wd->start\_at = new tm;

wd->end\_at = new tm;

utl::parseStorageStringToTime(value, \*wd->start\_at, \*wd->end\_at);

Конец

int wdPos = cell\_counter - 5;

wdList[wdPos] = \*wd;

Конец

cell\_counter++;

lastSeparatorPosition = currentSeparatorPosition + sizeof(utl::STORAGE\_SEPARATOR);

currentSeparatorPosition = line.find(utl::STORAGE\_SEPARATOR, lastSeparatorPosition);

node->workdays = wdList;

Конец цикла

ЕСЛИ prev не равно nullptr

prev->next = node;

prev = node;

ЕСЛИ nodeList->root равно nullptr

nodeList->root = node;

nodeList->curr = node;

Конец

Конец цикла

Конец

**saveData**

ofstream storage(STORAGE\_FILEPATH);

Doctor \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало цикла

string line;

line.append(to\_string(node->id) + utl::STORAGE\_SEPARATOR);

line.append(node->full\_name + utl::STORAGE\_SEPARATOR);

line.append(node->specialty + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->category\_id) + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->duration) + utl::STORAGE\_SEPARATOR);

Цикл по i от 0 до 7 с шагом 1

Начало цикла

WorkDay \*wd = &(node->workdays[i]);

ЕСЛИ (не wd->is\_day\_off)

line.append(utl::reprTimePair(wd->start\_at, wd->end\_at));

line += utl::STORAGE\_SEPARATOR;

Конец цикла

storage << line << endl;

node = node->next;

Конец цикла

**clearMemory**

Doctor \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало цикла

Doctor \*buffer = node->next;

delete[] node->workdays;

delete node;

node = buffer;

Конец цикла

delete nodeList;

**deleteItemByID**

Doctor \*node = nodeList->root;

Doctor \*prev = nullptr;

int schet = 0;

ПОКА (node не равно nullptr)

Начало цикла

ЕСЛИ (node->id равно id)

Начало

schet += 1;

ЕСЛИ (prev равно nullptr)

Начало

nodeList->root = node->next;

delete node->workdays;

delete node;

node = nodeList->root;

Конец

ИНАЧЕ

Начало

prev->next = node->next;

delete[] node->workdays;

delete node;

node = prev->next;

Конец

Конец

prev = node;

node = node->next;

Конец цикла

saveData();

ЕСЛИ (schet == 0)

вернуть false;

ИНАЧЕ

вернуть true;

**sortList**

bool unsorted = true;

Doctor \*curr, \*next, \*prev, \*buffer;

ПОКА (unsorted)

Начало цикла

unsorted = false;

curr = nodeList->root;

prev = nullptr;

ПОКА (curr)

Начало

next = curr->next;

ЕСЛИ (isNeedSwapBySortType(curr, next))

Начало

swap(curr, next);

ЕСЛИ (prev не равно nullptr)

prev->next = curr;

ИНАЧЕ

nodeList->root = curr;

buffer = curr->next;

curr->next = next;

next->next = buffer;

unsorted = true;

Конец

nodeList->curr = curr;

prev = curr;

curr = next;

Конец

Конец цикла

**isNeedSwapBySortType**

ЕСЛИ (next равно nullptr)

Вернуть false;

ЕСЛИ (sortListIndex)

Начало

равно LIST\_SORT\_TYPE\_ID:

Вернуть curr->id больше next->id;

равно LIST\_SORT\_TYPE\_NAME:

Вернуть curr->full\_name больше next->full\_name;

равно LIST\_SORT\_TYPE\_SPECIALTY:

Вернуть curr->specialty больше next->specialty;

равно LIST\_SORT\_TYPE\_CATEGORY:

Вернуть DOCTORS\_CATEGORIES[curr->category\_id] больше DOCTORS\_CATEGORIES[next->category\_id];

Конец

ИНАЧЕ:

Вернуть false;

**getLastID**

int last\_id = 0;

Doctor \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало

ЕСЛИ (last\_id меньше node->id)

Начало

last\_id = node->id;

Конец

node = node->next;

Конец

Вернуть last\_id;

**getLastNode**

Doctor \*prev = nullptr;

Doctor \*curr = nodeList->root;

ПОКА (curr не равно nullptr)

Начало

prev = curr;

curr = curr->next;

Конец

Вернуть prev;

**findItemByID**

int staffID = utl::readNumberFromConsole("Enter staff ID");

Doctor \*node = getItemByID(staffID);

ЕСЛИ (node не равно nullptr)

Начало

nodeList->curr = node;

Вернуть true;

Конец

Вернуть false;

**getItemByID**

Doctor \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало

ЕСЛИ (node->id равно id)

Вернуть node;

node = node->next;

Конец

Вернуть nullptr;

**getListRepresentation**

Doctor \*node = nodeList->root;

string buffer;

buffer.append("ID\tDATA\n");

int counter = 0;

ПОКА (node не равно nullptr)

Начало

string line = getItemRepresentation(node);

buffer.append(line);

counter++;

node = node->next;

Конец

ЕСЛИ (counter равно 0)

Начало

buffer.clear();

buffer.append("Nothing to show...");

Конец

Вернуть buffer;

**getItemRepresentation**

string buffer;

buffer.append(to\_string(node->id) + '\t');

buffer.append(node->full\_name);

buffer.append(" (" + node->specialty + '/' + DOCTORS\_CATEGORIES[node->category\_id] + ")\n");

buffer.append("\treception duration: " + to\_string(node->duration) + " minutes\n");

Цикл по i от 0 до 7 с шагом 1

Начало цикла

WorkDay \*wd = &(node->workdays[i]);

ЕСЛИ (не wd->is\_day\_off)

Начало

buffer += "\t> ";

buffer += utl::getWeekDayName(i, true);

buffer += " [";

buffer += utl::reprTimePair(wd->start\_at, wd->end\_at);

buffer += "] <\n";

Конец

Конец цикла

buffer += '\n';

Вернуть buffer;

**isNotOnVacation**

Цикл по i от 0 до 7 с шагом 1

Начало цикла

WorkDay \*wd = &(node->workdays[i]);

ЕСЛИ (не wd->is\_day\_off)

Вернуть true;

Конец цикла

Вернуть false;

**patient.cpp**

PatientList \*nodeList;

int sortListIndex = LIST\_SORT\_TYPE\_ID;

**addItem**

auto \*node = new Patient;

ЕСЛИ (utl::readLineFromConsoleTo("full name", node->full\_name))

Начало

Patient \*lastNode = getLastNode();

ЕСЛИ (lastNode равно nullptr)

Начало

node->id = 1;

nodeList->root = node;

Конец

ИНАЧЕ

Начало

node->id = getLastID() + 1;

lastNode->next = node;

Конец

nodeList->curr = node;

saveData();

utl::setMenuMessage("Element successfully created!");

Конец

ИНАЧЕ

Начало

delete node

utl::setMenuMessage("Error adding element!");

Конец

**editItem**

ЕСЛИ (findItemByID())

Начало

Patient \*node = nodeList->curr;

string buffer;

bool changed = false;

ЕСЛИ (utl::readAnswerFromConsole("Change [full name?]"))

Начало

ЕСЛИ (utl::readLineFromConsoleTo("full name", buffer))

Начало

node->full\_name = buffer;

changed = true;

Конец

buffer.clear();

Конец

ЕСЛИ (changed)

Начало

saveData();

utl::setMenuMessage("Element successfully updated!");

Конец

ИНАЧЕ

utl::setMenuMessage("Nothing to change!");

Конец

ИНАЧЕ

utl::setMenuMessage("Error updating element!");

**deleteItem**

int staffID = utl::readNumberFromConsole("Enter patient ID");

ЕСЛИ (staffID не равно NULL)

Начало

ЕСЛИ (deleteItemByID(staffID))

utl::setMenuMessage("Patient successfully deleted!");

ИНАЧЕ

utl::setMenuMessage("Patient was not found!");

Конец

ИНАЧЕ

utl::setMenuMessage("Incorrect ID!");

**deleteItemKnowID**

utl::clearScreen();

ЕСЛИ (PatID != NULL)

Начало

ЕСЛИ (deleteItemByID(PatID))

Начало

utl::setMenuMessage("Patient successfully deleted!");

Конец

ИНАЧЕ

utl::setMenuMessage("Patient was not found!");

Конец

ИНАЧЕ

utl::setMenuMessage("Incorrect ID!");

**findItems**

string search = utl::readLineFromConsole("Enter search string");

search = utl::strToLower(search);

Patient \*node = nodeList->root;

string buffer;

ПОКА (node не равно nullptr)

Начало цикла

string repr = getItemRepresentation(node);

string reprLower = utl::strToLower(repr);

string::size\_type matchPos = reprLower.find(search);

ЕСЛИ (matchPos не равно string::npos)

buffer.append(repr);

node = node->next;

Конец цикла

ЕСЛИ (buffer.empty())

utl::setMenuMessage("Matches not found!");

ИНАЧЕ

utl::setMenuMessage(buffer.c\_str());

**printList**

string buffer = getListRepresentation();

utl::setMenuMessage(buffer.c\_str());

**setSort**

sortListIndex = sortType;

sortList();

string message = "Set sorting by " + LIST\_SORT\_TYPES[sortListIndex];

utl::setMenuMessage(message.c\_str());

**loadData**

nodeList = new PatientList;

ifstream storage(STORAGE\_FILEPATH);

ЕСЛИ (storage)

Начало

string line;

Patient \*node;

Patient \*prev = nullptr;

ПОКА (getline(storage, line))

Начало цикла

ЕСЛИ (!line.empty())

Начало

node = new Patient;

short cell\_counter = 0;

string::size\_type lastSepPos = 0;

string::size\_type currSepPos = line.find(utl::STORAGE\_SEPARATOR, lastSepPos);

ПОКА (currSepPos не равно string::npos

string value = line.substr(lastSepPos, currSepPos - lastSepPos);)

Начало цикла

ЕСЛИ (cell\_counter)

Начало

равно 0:

node->id = stoi(value);

равно 1:

node->full\_name = value;

Конец

cell\_counter++;

lastSepPos = currSepPos + sizeof(utl::STORAGE\_SEPARATOR);

currSepPos = line.find(utl::STORAGE\_SEPARATOR, lastSepPos);

Конец цкила

ЕСЛИ (prev не равно nullptr)

prev->next = node;

ИНАЧЕ

nodeList->root = node;

prev = node;

nodeList->curr = node;

Конец

Конец цикла

Конец

**saveData**

ofstream storage(STORAGE\_FILEPATH);

Patient \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало цикла

string line;

line.append(to\_string(node->id) + utl::STORAGE\_SEPARATOR);

line.append(node->full\_name + utl::STORAGE\_SEPARATOR);

storage << line << endl;

node = node->next;

Конец цикла

**clearMemory**

Patient \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало цикла

Patient \*buffer = node->next;

delete node;

node = buffer;

Конец цикла

delete nodeList;

**deleteItemByID**

Patient \*node = nodeList->root;

Patient \*prev = nullptr;

ПОКА (node не равно nullptr)

Начало цикла

ЕСЛИ (node->id равно id)

Начало

ЕСЛИ (prev равно nullptr)

nodeList->root = node->next;

ИНАЧЕ

prev->next = node->next;

delete node;

saveData();

Вернуть true;

Конец

prev = node;

node = node->next;

Конец цикла

Вернуть false;

**sortList**

bool unsorted = true;

Patient \*curr, \*next, \*prev, \*buffer;

ПОКА (unsorted)

Начало цикла

unsorted = false;

curr = nodeList->root;

prev = nullptr;

ПОКА (curr)

Начало

next = curr->next;

ЕСЛИ (isNeedSwapBySortType(curr, next))

Начало

swap(curr, next);

ЕСЛИ (prev не равно nullptr)

prev->next = curr;

ИНАЧЕ

nodeList->root = curr;

buffer = curr->next;

curr->next = next;

next->next = buffer;

unsorted = true;

Конец

nodeList->curr = curr;

prev = curr;

curr = next;

Конец

Конец цикла

**isNeedSwapBySortType**

ЕСЛИ (next равно nullptr)

Вернуть false;

ЕСЛИ (sortListIndex)

Начало

равно LIST\_SORT\_TYPE\_ID:

Вернуть curr->id больше next->id;

равно LIST\_SORT\_TYPE\_NAME:

Вернуть curr->full\_name больше next->full\_name;

ИНАЧЕ:

Вернуть false;

Конец

**getLastID**

int last\_id = 0;

Patient \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало цикла

ЕСЛИ (last\_id меньше node->id)

last\_id = node->id;

node = node->next;

Конец цикла

Вернуть last\_id;

**getLastNode**

Patient \*prev = nullptr;

Patient \*curr = nodeList->root;

ПОКА (curr не равно nullptr)

Начало

prev = curr;

curr = curr->next;

Конец

Вернуть prev;

**findItemByID**

int staffID = utl::readNumberFromConsole("Enter patient ID");

Patient \*node = getItemByID(staffID);

ЕСЛИ (node не равно nullptr)

Начало

nodeList->curr = node;

Вернуть true;

Конец

Вернуть false;

**getItemByID**

Patient \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало

ЕСЛИ (node->id равно id)

Вернуть node;

node = node->next;

Конец

Вернуть nullptr;

**getListRepresentation**

Patient \*node = nodeList->root;

string buffer;

int counter = 0;

ПОКА (node не равно nullptr)

Начало

string line = getItemRepresentation(node);

buffer.append(line);

counter++;

node = node->next;

Конец

ЕСЛИ (counter равно 0)

Начало

buffer.clear();

buffer.append("Nothing to show...");

Конец

Вернуть buffer;

**getItemRepresentation**

string buffer;

buffer.append(to\_string(node->id) + '\t');

buffer.append(node->full\_name);

buffer += '\n';

Вернуть buffer;

**reception.cpp**

ReceptionList \*nodeList;

int sortListIndex = LIST\_SORT\_TYPE\_ID;

**addItem**

auto \*node = new Reception;

int doctor\_id = utl::readNumberFromConsole("Enter doctor ID");

node->doctor = staff::getItemByID(doctor\_id);

bool validDoctor = node->doctor не равно nullptr и staff::isNotOnVacation(node->doctor);

int patient\_id = utl::readNumberFromConsole("Enter patient ID");

node->patient = patient::getItemByID(patient\_id);

bool validPatient = node->patient не равно nullptr;

ЕСЛИ (validDoctor и validPatient и readAvailableDateFromConsole(\*node))

Начало

Reception \*lastNode = getLastNode();

ЕСЛИ (lastNode равно nullptr)

Начало

node->id = 1;

nodeList->root = node;

Конец

ИНАЧЕ

Начало

node->id = getLastID() + 1;

lastNode->next = node;

Конец

nodeList->curr = node;

saveData();

utl::setMenuMessage("Element successfully created!");

Конец

ИНАЧЕ

Начало

delete node;

utl::setMenuMessage("Error adding element!");

Конец

**editItem**

ЕСЛИ (findItemByID())

Начало

Reception \*node = nodeList->curr;

bool changed = false;

ЕСЛИ (node->conclusion.empty())

Начало

ЕСЛИ (utl::readAnswerFromConsole("Change [doctor]?"))

Начало

int doctor\_id = utl::readNumberFromConsole("Enter doctor ID");

staff::Doctor \*doctor = staff::getItemByID(doctor\_id);

ЕСЛИ (node->doctor не равно nullptr и staff::isNotOnVacation(node->doctor))

Начало

node->doctor = doctor;

changed = true;

Конец

Конец

ЕСЛИ (utl::readAnswerFromConsole("Change [patient]?"))

Начало

int patient\_id = utl::readNumberFromConsole("Enter patient ID");

patient::Patient \*patient = patient::getItemByID(patient\_id);

ЕСЛИ (node->patient не равно nullptr)

Начало

node->patient = patient;

changed = true;

Конец

Конец

ЕСЛИ (utl::readAnswerFromConsole("Change [visit time]?"))

Начало

ЕСЛИ (readAvailableDateFromConsole(\*node))

changed = true;

Конец

ЕСЛИ (changed)

Начало

saveData();

utl::setMenuMessage("Element successfully updated!");

Конец

ИНАЧЕ

utl::setMenuMessage("Nothing to change!");

Конец

ИНАЧЕ

utl::setMenuMessage("You can not edit element with conclusion!");

Конец

ИНАЧЕ

utl::setMenuMessage("Element does not found!");

**setConclusion**

ЕСЛИ (findItemByID())

Начало

Reception \*node = nodeList->curr;

string buffer;

bool changed = false;

ЕСЛИ (node->conclusion.empty()

changed = utl::readLineFromConsoleTo("conclusion", node->conclusion);

ИНАЧЕ ЕСЛИ (utl::readAnswerFromConsole(“Change [conclusion]?”))

Начало

ЕСЛИ (utl::readLineFromConsoleTo(“conclusion”, buffer))

Начало

node->conclusion = buffer;

changed = true;

Конец

Конец

ЕСЛИ (changed)

Начало

saveData();

utl::setMenuMessage("Conclusion successfully updated!");

Конец

ИНАЧЕ

utl::setMenuMessage("Nothing to change!");

Конец

ИНАЧЕ

utl::setMenuMessage("Element does not found!");

**deleteItem**

int staffID = utl::readNumberFromConsole("Enter staff ID");

ЕСЛИ (staffID равно NULL)

utl::setMenuMessage("Incorrect ID!");

ЕСЛИ (deleteItemByID(staffID))

utl::setMenuMessage("Doctor successfully deleted!");

ИНАЧЕ

utl::setMenuMessage("Doctor was not found!");

**findItems**

string search = utl::readLineFromConsole("Enter search string");

search = utl::strToLower(search);

Reception \*node = nodeList->root;

string buffer;

ПОКА (node не равно nullptr)

Начало

string repr = getItemRepresentation(node);

string reprLower = utl::strToLower(repr);

string::size\_type matchPos = reprLower.find(search);

ЕСЛИ (matchPos не равно string::npos)

buffer.append(repr);

node = node->next;

Конец

ЕСЛИ (buffer.empty())

utl::setMenuMessage("Matches not found!");

ИНАЧЕ

utl::setMenuMessage(buffer.c\_str());

**printList**

string buffer = getListRepresentation(printType);

utl::setMenuMessage(buffer.c\_str());

**setSort**

sortListIndex = sortType;

sortList();

string message = "Set sorting by " + LIST\_SORT\_TYPES[sortListIndex];

utl::setMenuMessage(message.c\_str());

**loadData**

nodeList = new ReceptionList;

ifstream storage(STORAGE\_FILEPATH);

ЕСЛИ (storage)

Начало

string line;

Reception \*node;

Reception \*prev = nullptr;

ПОКА (getline(storage, line))

Начало

ЕСЛИ (не line.empty())

Начало

node = new Reception;

short cell\_counter = 0;

string::size\_type lastSepPos = 0;

string::size\_type currSepPos = line.find(utl::STORAGE\_SEPARATOR, lastSepPos);

ПОКА (currSepPos не равно string::npos

string value = line.substr(lastSepPos, currSepPos - lastSepPos);)

Начало цикла

ЕСЛИ (cell\_counter)

Начало

равно 0:

node->id = stoi(value);

равно 1:

node->doctor = staff::getItemByID(stoi(value));

равно 2:

node->patient = patient::getItemByID(stoi(value));

равно 3:

node->duration = stoi(value);

равно 4:

node->visit\_at = new tm;

utl::parseStorageStringDateTime(value, \*node->visit\_at);

mktime(node->visit\_at);

равно 5:

node->conclusion = value;

Конец

cell\_counter++;

lastSepPos = currSepPos + sizeof(utl::STORAGE\_SEPARATOR);

currSepPos = line.find(utl::STORAGE\_SEPARATOR, lastSepPos);

Конец цикла

ЕСЛИ (node->doctor равно nullptr || node->patient равно nullptr)

continue;

ЕСЛИ (prev не равно nullptr)

prev->next = node;

ИНАЧЕ

nodeList->root = node;

prev = node;

nodeList->curr = node;

Конец

Конец

Конец

**saveData**

ofstream storage(STORAGE\_FILEPATH);

Reception \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало

string line;

line.append(to\_string(node->id) + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->doctor->id) + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->patient->id) + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->duration) + utl::STORAGE\_SEPARATOR);

line.append(utl::reprDateTime(node->visit\_at) + utl::STORAGE\_SEPARATOR);

line.append(node->conclusion + utl::STORAGE\_SEPARATOR);

storage << line << endl;

node = node->next;

Конец

**clearMemory**

Reception \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало

Reception \*buffer = node->next;

delete node->visit\_at;

delete node;

node = buffer;

Конец

delete nodeList;

**deleteItemByID**

Reception \*node = nodeList->root;

Reception \*prev = nullptr;

ПОКА (node не равно nullptr)

Начало цикла

ЕСЛИ (node->id равно id)

Начало

ЕСЛИ (prev равно nullptr)

nodeList->root = node->next;

ИНАЧЕ

prev->next = node->next;

delete node->visit\_at;

delete node;

saveData();

Вернуть true;

Конец

prev = node;

node = node->next;

Конец цикла

Вернуть false;

**deleteItemByDoctor**

Reception \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало цикла

ЕСЛИ (node->doctor->id меньше или равно 0)

Начало

deleteItemByID(node->id);

Прервать;

Конец

node = node->next;

Конец цикла

**deleteItemByPatient**

Reception \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало цикла

ЕСЛИ (node->patient->id меньше или равно 0)

Начало

deleteItemByID(node->id);

Прервать;

Конец

node = node->next;

Конец цикла

**sortList**

bool unsorted = true;

Reception \*curr, \*next, \*prev, \*buffer;

ПОКА (unsorted)

Начало цикла

unsorted = false;

curr = nodeList->root;

prev = nullptr;

ПОКА (curr)

Начало цикла

next = curr->next;

ЕСЛИ (isNeedSwapBySortType(curr, next))

Начало

- Меняем значения местами.

swap(curr, next);

ЕСЛИ (prev не равно nullptr)

prev->next = curr;

ИНАЧЕ

nodeList->root = curr;

buffer = curr->next;

curr->next = next;

next->next = buffer;

unsorted = true;

Конец

nodeList->curr = curr;

prev = curr;

curr = next;

Конец цикла

Конец цикла

**isNeedSwapBySortTypе**

ЕСЛИ (next равно nullptr)

Вернуть false;

ЕСЛИ (sortListIndex)

Начало

равно LIST\_SORT\_TYPE\_ID:

Вернуть curr->id больше next->id;

равно LIST\_SORT\_TYPE\_DOCTOR:

Вернуть curr->doctor->full\_name больше next->doctor->full\_name;

равно LIST\_SORT\_TYPE\_PATIENT:

Вернуть curr->patient->full\_name больше next->patient->full\_name;

равно LIST\_SORT\_TYPE\_VISIT\_TIME:

Вернуть utl::isDTGreaterThan(curr->visit\_at, next->visit\_at);

ИНАЧЕ:

Вернуть false;

Конец

**getLastID**

int last\_id = 0;

Reception \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало цикла

ЕСЛИ (last\_id < node->id)

last\_id = node->id;

node = node->next;

Конец цикла

Вернуть last\_id;

**getLastNode**

Reception \*prev = nullptr;

Reception \*curr = nodeList->root;

ПОКА (curr не равно nullptr)

Начало

prev = curr;

curr = curr->next;

Конец

Вернуть prev;

**findItemByID**

int staffID = utl::readNumberFromConsole("Enter reception ID");

Reception \*node = getItemByID(staffID);

ЕСЛИ (node не равно nullptr)

Начало

nodeList->curr = node;

Вернуть true;

Конец

Вернуть false;

**getItemByID**

Reception \*node = nodeList->root;

ПОКА (node не равно nullptr)

Начало

ЕСЛИ (node->id равно id)

Вернуть node;

node = node->next;

Конец

Вернуть nullptr;

**getListRepresentation**

Reception \*node = nodeList->root;

string buffer;

time\_t today\_time{};

time(&today\_time);

tm \* today = localtime(&today\_time);

int counter = 0;

bool canPrint;

ПОКА (node не равно nullptr)

Начало цикла

ЕСЛИ (printType)

Начало

равно PRINT\_TYPE\_TODAY:

canPrint = node->conclusion.empty() и utl::isSameDay(today, node->visit\_at);

break;

равно PRINT\_TYPE\_WAITING:

canPrint = node->conclusion.empty() и utl::isDTGreaterThan(node->visit\_at, today);

break;

равно PRINT\_TYPE\_OVERDUE:

canPrint = node->conclusion.empty() и utl::isDTGreaterThan(today, node->visit\_at);

break;

равно PRINT\_TYPE\_COMPLETED:

canPrint = не node->conclusion.empty();

break;

равно PRINT\_TYPE\_ALL:

canPrint = true;

break;

ИНАЧЕ:

canPrint = false;

break;

Конец

ЕСЛИ (canPrint)

Начало

string line = getItemRepresentation(node);

buffer.append(line);

counter++;

Конец

node = node->next;

Конец цикла

ЕСЛИ (counter равно 0)

buffer.clear();

Вернуть buffer;

**getItemRepresentation**

string buffer;

buffer.append(to\_string(node->id) + '\t');

buffer.append('[' + utl::getWeekDayName(node->visit\_at->tm\_wday, true) + "] ");

buffer.append(utl::reprDateTime(node->visit\_at) + '\t');

buffer.append(node->doctor->full\_name + '(' + node->doctor->specialty + ") / " + node->patient->full\_name);

ЕСЛИ (не node->conclusion.empty())

buffer.append("\n\tConclusion: " + node->conclusion + '\n');

buffer += '\n';

Вернуть buffer;

tm \*getAvailableDateTime(Reception \*node, tm \*day, staff::WorkDay \* wd)

staff::Doctor \*doctor = node->doctor;

patient::Patient \*patient = node->patient;

tm \*available = new tm;

\*available = \*day;

available->tm\_hour = wd->start\_at->tm\_hour;

available->tm\_min = wd->start\_at->tm\_min;

int receptCnt = 0;

Reception \*curr = nodeList->root;

ПОКА (curr не равно nullptr и receptCnt меньше MAX\_RECEPTION\_PER\_DAY)

Начало

tm \*visit\_at = curr->visit\_at;

ЕСЛИ (curr->conclusion.empty() и utl::isSameDay(day, visit\_at))

Начало

bool isSamePatient = curr->patient->id равно patient->id;

ЕСЛИ (curr->doctor->id равно doctor->id)

Начало

ЕСЛИ (isSamePatient)

Начало

delete available;

Вернуть nullptr;

Конец

bool validHour = visit\_at->tm\_hour больше или равно available->tm\_hour;

bool validMinutes = visit\_at->tm\_min больше или равно available->tm\_min;

ЕСЛИ (validHour и validMinutes)

Начало

available->tm\_hour = visit\_at->tm\_hour;

available->tm\_min = visit\_at->tm\_min;

available->tm\_min += curr->duration;

Конец

Конец

ИНАЧЕ ЕСЛИ (isSamePatient)

receptCnt++;

Конец

curr = curr->next;

Конец

mktime(available);

ЕСЛИ (receptCnt меньше или равно MAX\_RECEPTION\_PER\_DAY и utl::isCorrectTimePair(wd->end\_at, available, doctor->duration))

Вернуть available;

ИНАЧЕ

delete available;

Вернуть nullptr;

**readAvailableDateFromConsole**

time\_t buffer{};

time(&buffer);

int counter = 0;

tm \*day = localtime(&buffer);

tm visitList[14];

цикл по i от 0 до 14; i++

Начало цикла

day->tm\_mday += 1;

mktime(day);

staff::WorkDay \*wd = &(node.doctor->workdays[day->tm\_wday]);

ЕСЛИ (wd->is\_day\_off)

continue;

tm \*available = getAvailableDateTime(&node, day, wd);

ЕСЛИ (available не равно nullptr)

Начало

visitList[counter] = \*available;

Вывод available;

counter++;

Конец

Конец цикла

int choice;

ЕСЛИ (counter больше 0)

Начало

ЕСЛИ (utl::readNumberFromConsoleTo("visit time", choice, 0, counter - 1))

Начало

tm \*dt = new tm;

\*dt = visitList[choice];

node.visit\_at = dt;

node.duration = node.doctor->duration;

Вернуть true;

Конец

Конец

Вернуть false;

utils.cpp

string operationMessage;

**clearScreen**

system("cls");

**readAnswerFromConsole**

string answer = readWordFromConsole(title);

Вернуть tolower(answer[0]) равно 'y';

**readNumberFromConsole**

int tries = 0;

int buffer;

string input;

ПОКА (tries меньше MAX\_INPUT\_TRIES )

Начало цикла

cin >> input;

Попробовать

buffer = stoi(input);

Вернуть buffer;

Поймать invalid\_argument &err

tries++;

Поймать out\_of\_range &err

tries++;

Конец цикла

Вернуть NULL;

**readWordFromConsole**

string line;

cin >> line;

Вернуть line;

**readLineFromConsole**

int tries = 0;

string line;

Выполнить

Начало цикла

getline(cin, line);

tries++;

Конец цикла

ПОКА (line.empty() или tries >= MAX\_INPUT\_TRIES)

Вернуть line;

**getWeekDayName**

string name;

ЕСЛИ (abbreviation )

name = WEEK\_DAYS\_SHORT[index];

ИНАЧЕ

name = WEEK\_DAYS[index];

Вернуть name;

**getCurrDateTime**

time\_t curr\_time{};

time(&curr\_time);

tm \*curr\_tm = localtime(&curr\_time);

Вернуть curr\_tm;

**readTimeFromConsole**

tm \*buffer = new tm;

string value;

stringstream ss;

ss << readWordFromConsole(fieldName);

ss >> std::get\_time(buffer, TIME\_FORMAT);

ЕСЛИ (ss.fail() )

Начало

delete buffer;

Вернуть nullptr;

Конец

tm \*curr\_tm = new tm;

\*curr\_tm = \*getCurrDateTime();

curr\_tm->tm\_hour = buffer->tm\_hour;

curr\_tm->tm\_min = buffer->tm\_min;

delete buffer;

Вернуть curr\_tm;

**readLineFromConsoleTo**

char buffer[150];

string detail, label;

label += "Enter ";

label += fieldName;

string value = utl::readLineFromConsole(label.c\_str());

string::size\_type pos = value.find(STORAGE\_SEPARATOR);

ЕСЛИ (value.empty() )

detail = "is empty";

ИНАЧЕ ЕСЛИ (pos не равно string::npos )

Начало

detail = "contain restrict char [";

detail += STORAGE\_SEPARATOR;

detail += ']';

Конец

ЕСЛИ (detail.empty() )

Начало

field = value;

Вернуть true;

Конец

sprintf(buffer, FIELD\_ERROR\_TEMPLATE, fieldName, detail.c\_str());

utl::setMenuMessage(buffer);

Вернуть false;

**readNumberFromConsoleTo**

char buffer[150];

string detail, label;

label += "Enter ";

label += fieldName;

int number = utl::readNumberFromConsole(label.c\_str());

ЕСЛИ (number меньше min )

detail = "less than " + to\_string(min);

ИНАЧЕ ЕСЛИ (number больше max )

detail = "greater than " + to\_string(max);

ЕСЛИ (detail.empty() )

Начало

field = number;

Вернуть true;

Конец

sprintf(buffer, FIELD\_ERROR\_TEMPLATE, fieldName, detail.c\_str());

utl::setMenuMessage(buffer);

Вернуть false;

**strToLower**

string buffer;

Для всех char c из s

Начало

buffer += tolower(c);

Конец

Вернуть buffer;

**isMenuMessage**

Вернуть не operationMessage.empty();

**setMenuMessage**

operationMessage += title;

**clearMenuMessage**

operationMessage.clear();

**printMenuMessage**

Вывод operationMessage

clearMenuMessage();

**parseStorageStringToTime**

stringstream startStr, endStr;

startStr << storageStr.substr(0, 5);

endStr << storageStr.substr(6);

tm \*curr\_tm = getCurrDateTime();

start = \*curr\_tm;

end = \*curr\_tm;

startStr >> get\_time(&start, TIME\_FORMAT);

endStr >> get\_time(&end, TIME\_FORMAT);

**parseStorageStringDateTime**

stringstream buffer;

dt = \*getCurrDateTime();

buffer << storageStr;

buffer >> get\_time(&dt, DATETIME\_FORMAT);

**reprTimePair**

stringstream repr;

repr << put\_time(start, TIME\_FORMAT) << '-' << put\_time(end, TIME\_FORMAT);

Вернуть repr.str();

**reprDateTime**

stringstream repr;

repr << put\_time(dt, DATETIME\_FORMAT);

Вернуть repr.str();

**isCorrectTimePair**

ЕСЛИ (start равно nullptr или end равно nullptr )

Вернуть false;

time\_t startTime = mktime(start);

time\_t endTime = mktime(end);

Вернуть difftime(endTime, startTime) больше или равно (duration \* 60);

**isDTGreaterThan**

ЕСЛИ (a равно nullptr или b равно nullptr )

Вернуть false;

time\_t a\_time = mktime(a);

time\_t b\_time = mktime(b);

Вернуть difftime(a\_time, b\_time) больше 0;

**bool isSameDay(tm \*a, tm \*b)**

Вернуть (a->tm\_year равно b->tm\_year и a->tm\_mon равно b->tm\_mon и a->tm\_mday равно b->tm\_mday);

# ПРИМЕР работы программы

Тестирование работоспособности программы производилось в операционной системе Windows 11.

Для удобства тестирование программы мы уже имеем некоторые данные в наших файлах-хранилищах (рис. 13, 14, 15).

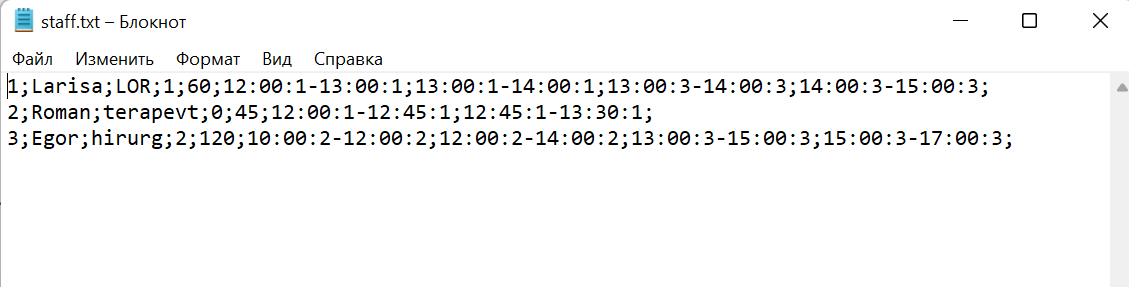


Рисунок 13 - Содержание файла staff.txt на момент начала тестирования

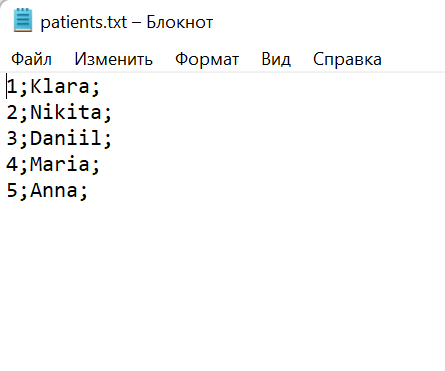


Рисунок 14 - Содержание файла pacient.txt на момент начала тестирования

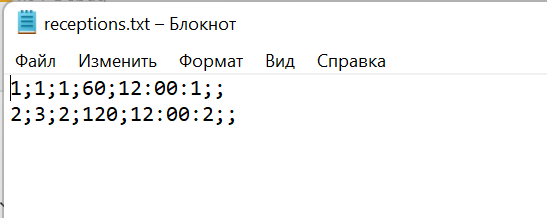


Рисунок 15 - Содержание файла receptions.txt на момент начала тестирования

При запуске программы появляется общее меню с выбором модуля для дальнейшей работы (рис. 16). Перейдём в меню: «Staff data». Для этого выберем пункт № 1 в главном меню.

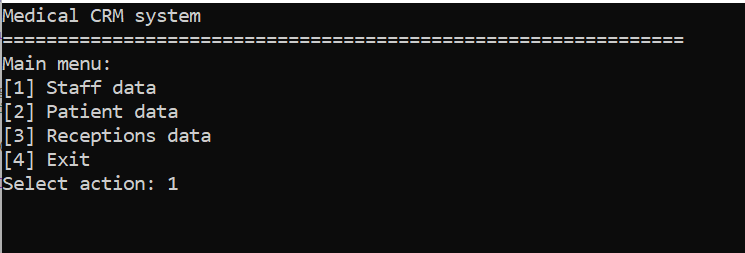


Рисунок 16 - Переход в меню «Staff data»

Данные сотрудников: «Staff data»

Для начала выведем список существующих врачей, для этого выберем пункт номер 5 (рис. 17).

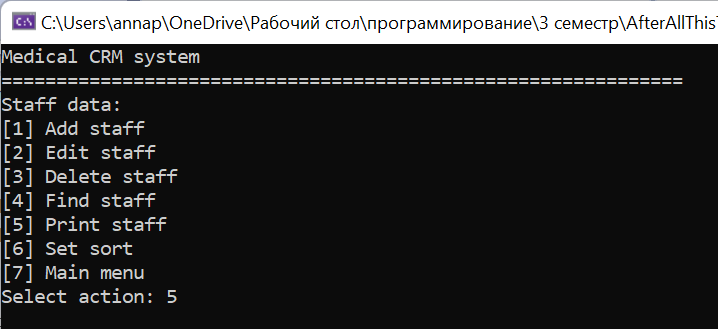


Рисунок 17 – Вывод содержимого в файле staff.txt на экран

На экране появляется список врачей (рис. 18).

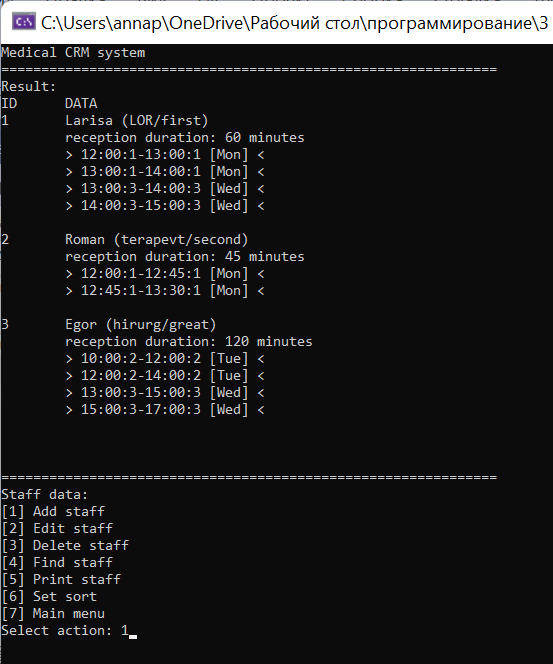


Рисунок 18 – Вывод всего списка врачей (Add staff)

Выбираем операцию добавления нового сотрудника: «Add staff». Для этого выберем пункт № 1 (рис. 18). Далее нам потребуется поочередно внести данные для нового сотрудника (рис. 19).

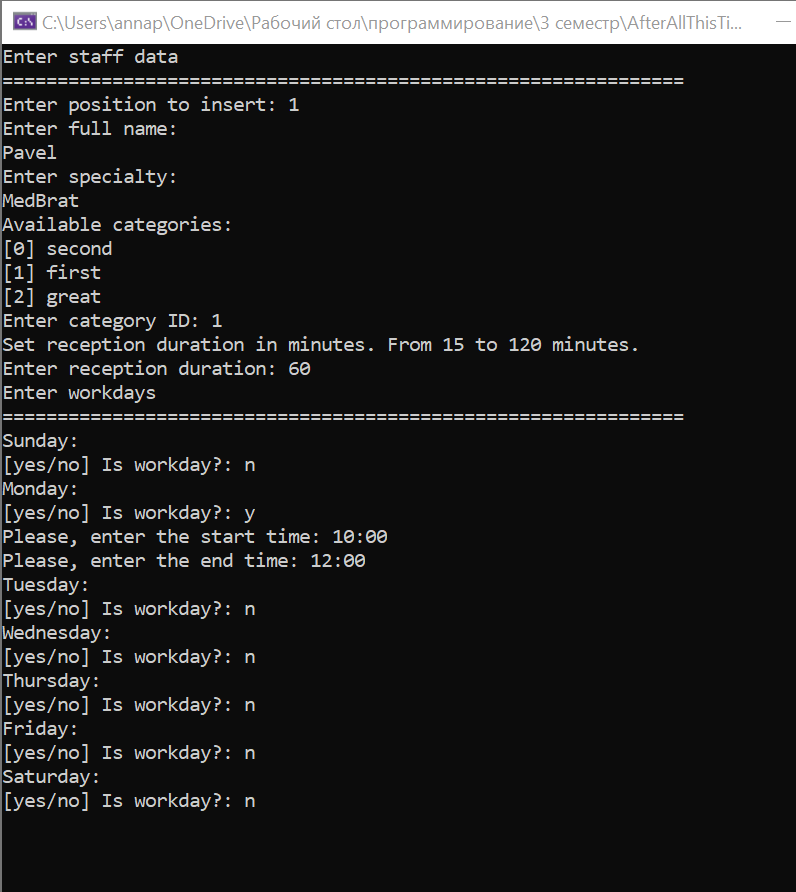


Рисунок 19 – Ввод данных по новому врачу

Новые данные:

1)Позиция для вставки «Position to insert»: после 1 элемента;

2)Имя «full name»: Pavel;

3)Специальность «specialty»: MedBrat;

4)Категория врача «Category»: first;

5)Длительность приёма «Duration»: 60 минут;

6)Рабочие дни «Workdays»: по понедельникам с 10:00 до 12:00.

После заполнения данных мы автоматически возвращаемся в главное меню, где видим сообщение о том, что элемент был успешно создан. Выведем список элементов, чтобы убедиться, что операция выполнена. Выберем пункт № 5 (рис. 20).

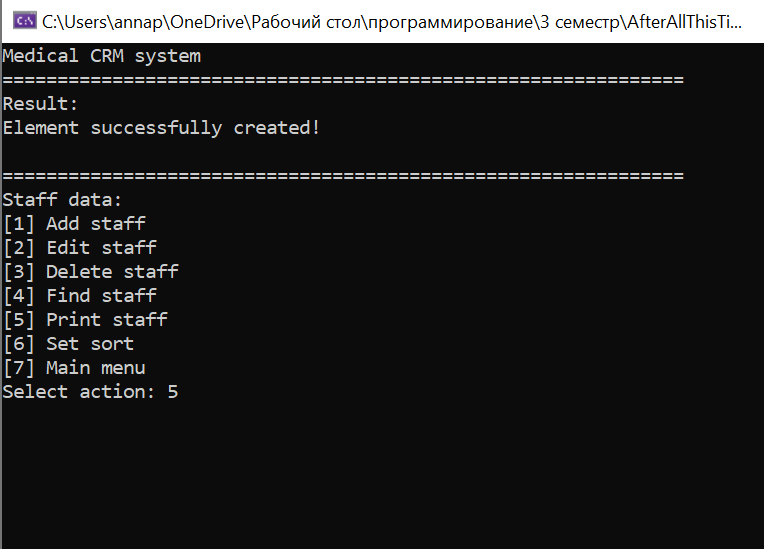


Рисунок 20 – Пример сообщения после создания врача

В консоли отображается полученный список всех врачей. Видно, что нашему новому врачу был присвоен ID под № 4. Отредактируем нашего врача. Для этого выберем пункт № 2 (рис. 21).

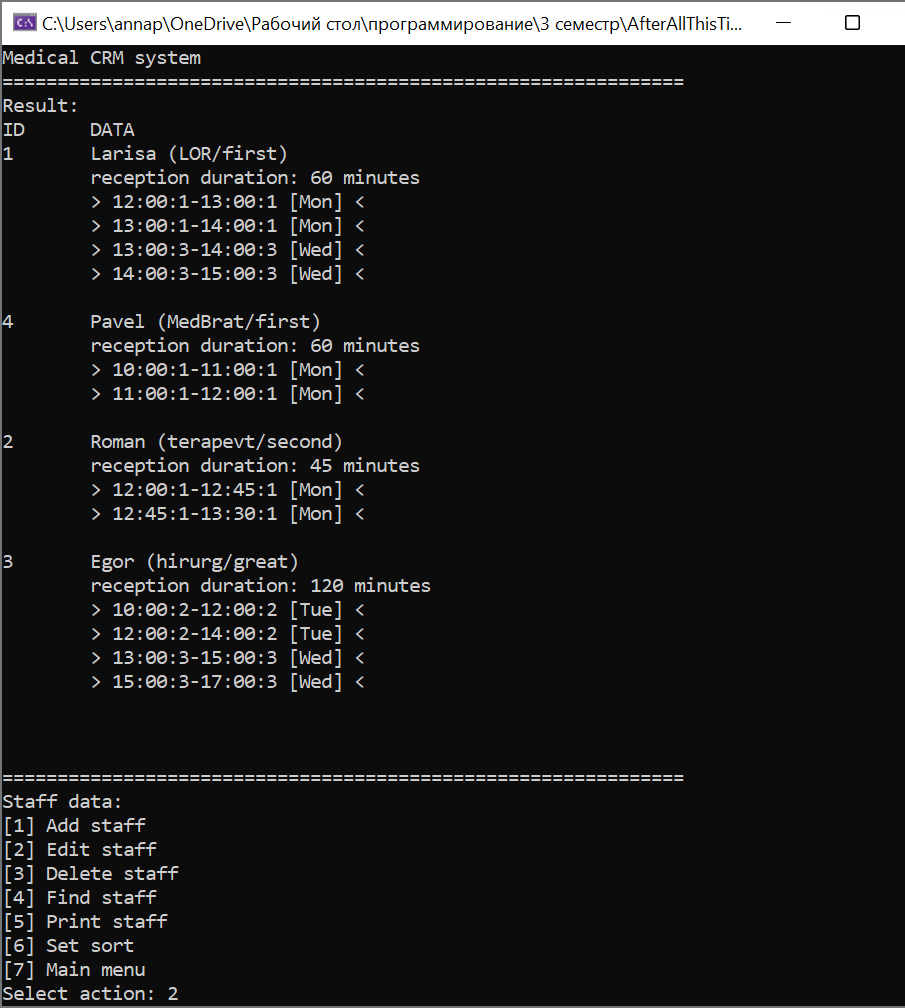


Рисунок 21 – Список врачей с новым элементом

Чтобы отредактировать врача потребуется ввести его ID. Вводим ID под № 4. Дальше изменим нашему врачу специальность и категорию(рис. 22).

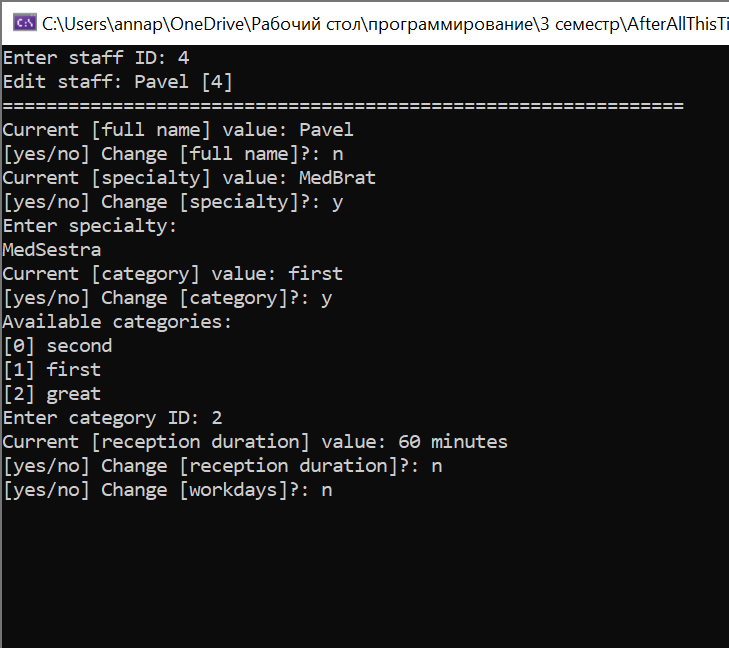


Рисунок 22 – Редактирование врача

Редактируемые данные:

1)Специальность «Specilty»: MedSestra;

2)Категория врача «Category»: great.

Отобразим список врачей (рис. 23) и увидим, что данные изменились.

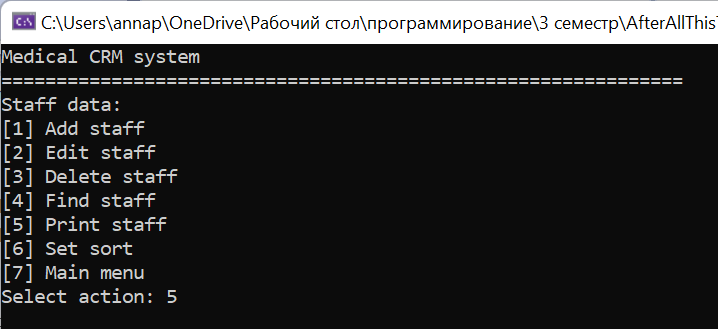


Рисунок 23 – Вывод списка врачей

Видим список врачей на рис. 24. Теперь попробуем изменить сортировку врачей. Выберем пункт № 6 (рис. 24).

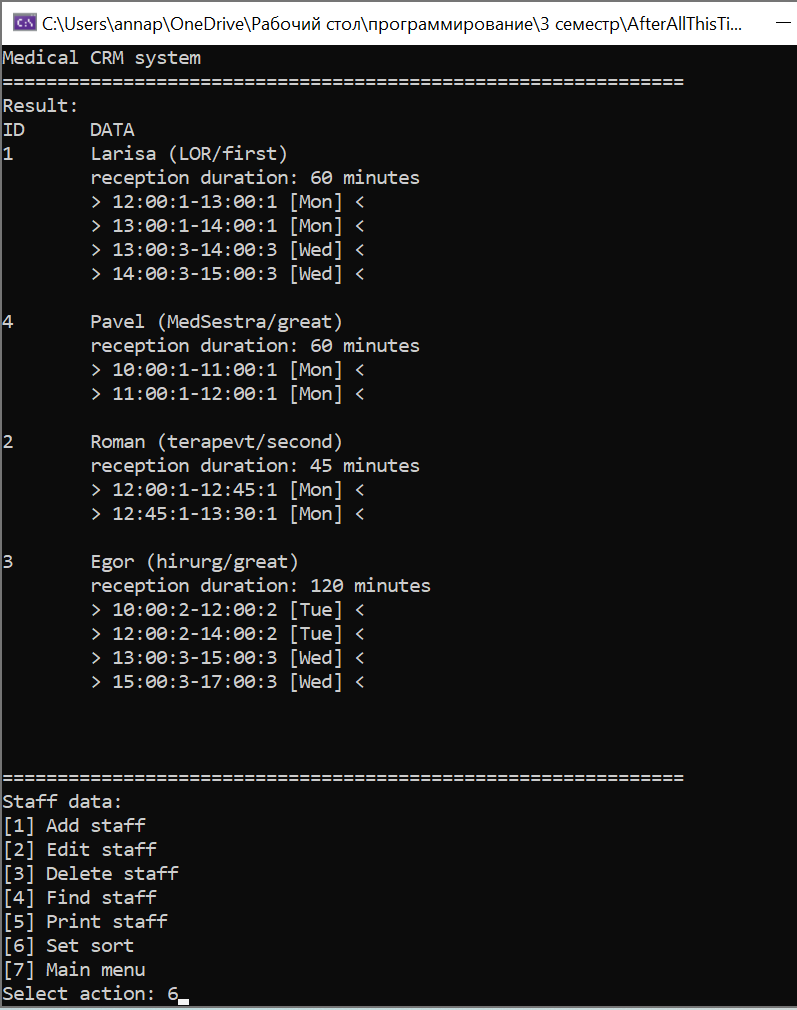


Рисунок 24 – Список врачей с изменённым элементом

Выберем сортировку по ФИО, пункт № 2 (рис. 25).

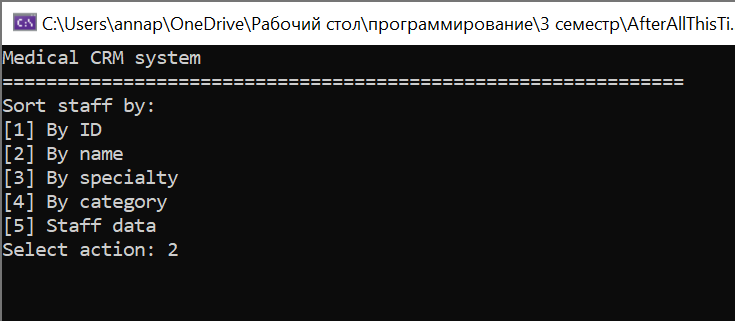


Рисунок 25 – Выбор сортировки врачей по ФИО

Список врачей теперь отсортированы по ФИО (рис. 26).

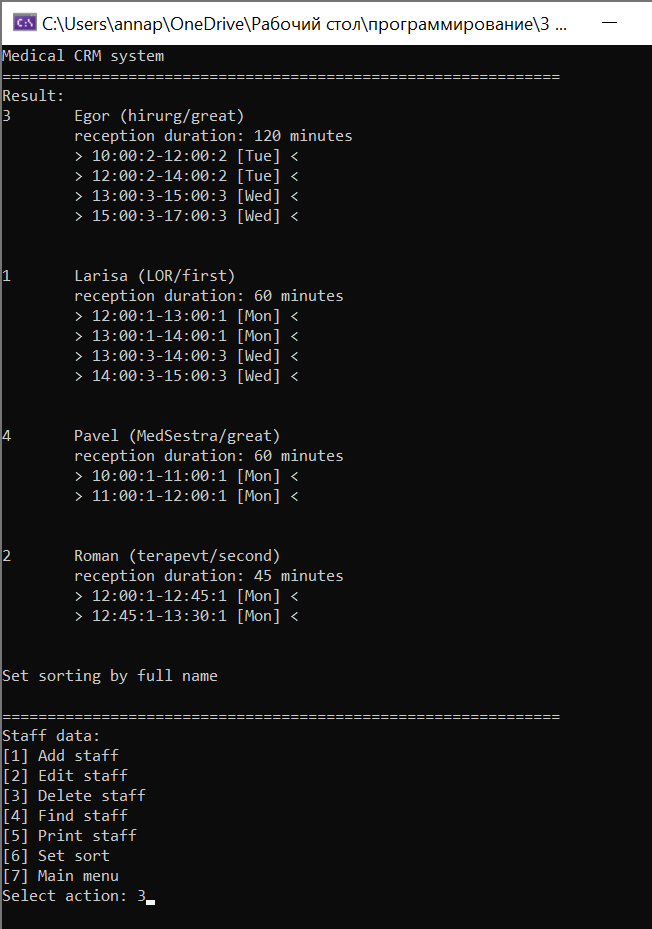


Рисунок 26 – Список врачей отсортированный по ФИО

Проверим операцию удаления. Выберем пункт № 3 удаление. Вводим ID врача под № 2 (рис. 27).

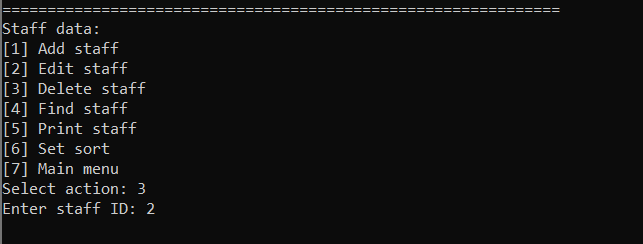


Рисунок 27 – Удаление врача с ID № 2

Видим сообщение, что врач удалён (рис. 28).

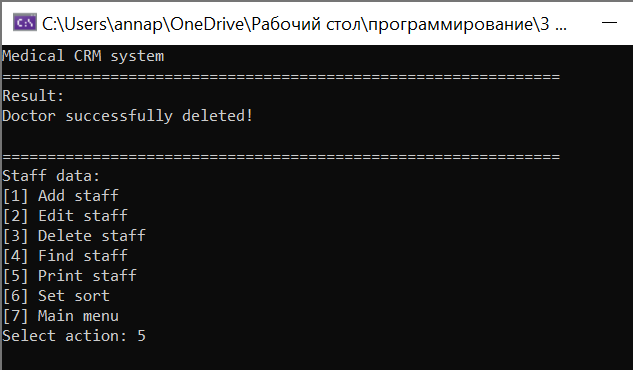


Рисунок 28 – Сообщение успешного удаления

Снова выведем список врачей. Выберем пункт меню № 5 (рис. 28).

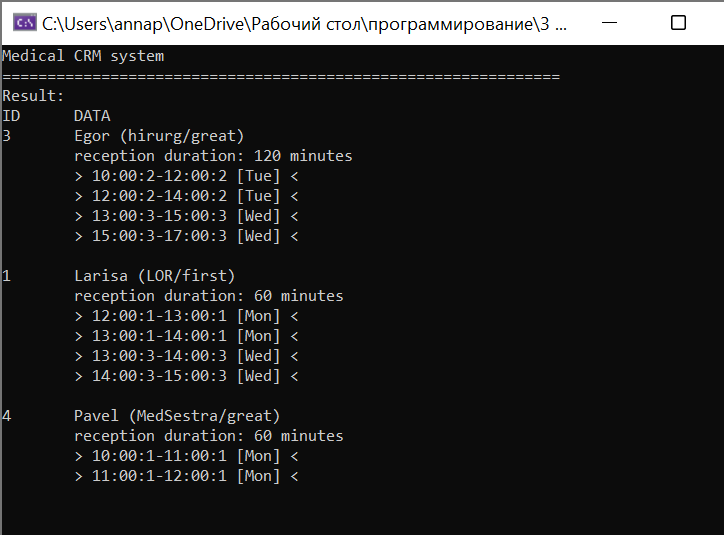


Рисунок 29 – Вывод списка врачей

Как видно из рис. 29, запись с ID № 2 отсутствует в списке врачей.

Выберем функцию поиска врачей (рис. 30) и введем Egor.

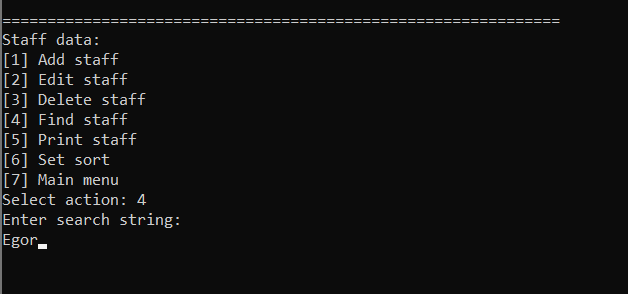


Рисунок 30 – Поиск врача

Как видим из рис. 31, программа нашла нужного врача и вывела всю информацию о нём.

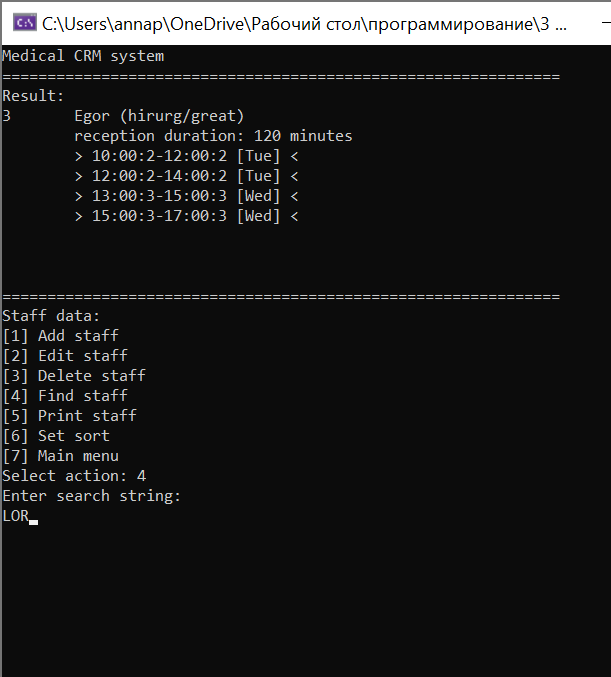


Рисунок 31 - Результат поиска

Попробуем найти врача по специальности. Для этого выберем пункт меню 4 и введем поисковую подстроку (Рис. 31).

Как мы можем видеть на рис. 32, программа отработала успешно.

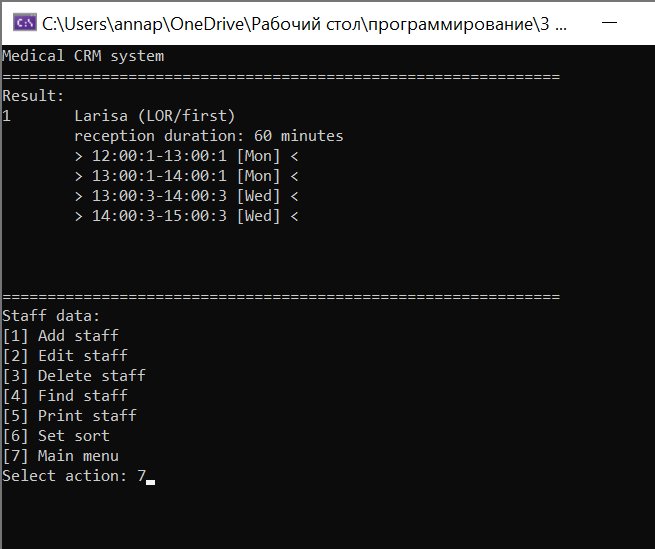


Рисунок 32 – Результат поиска

Вернёмся в главное меню (рис. 32).

Перейдём в меню **данные пациента** (Patient data). Для этого выберем пункт меню под № 2 (рис. 33).

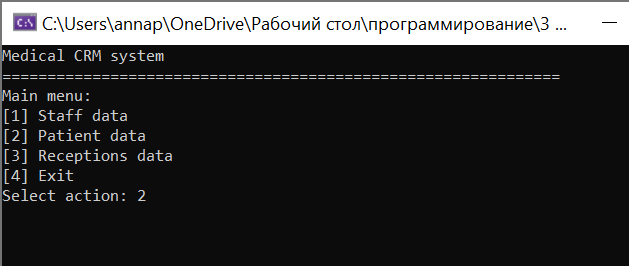


Рисунок 33 – Переход в меню «Patient data»

Выведем содержимое файла patients.txt на экран, для этого выберем пункт меню номер 5 (рис. 34).

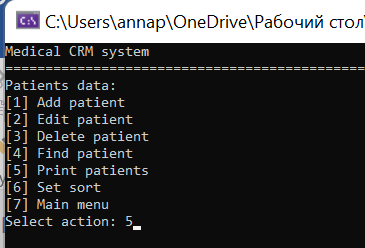


Рисунок 34 – Вывод имеющихся пациентов на экран

Видим список пациентов на экране(рис. 35). Добавим нового пациента. Выберем пункт меню под № 1 (рис. 35).

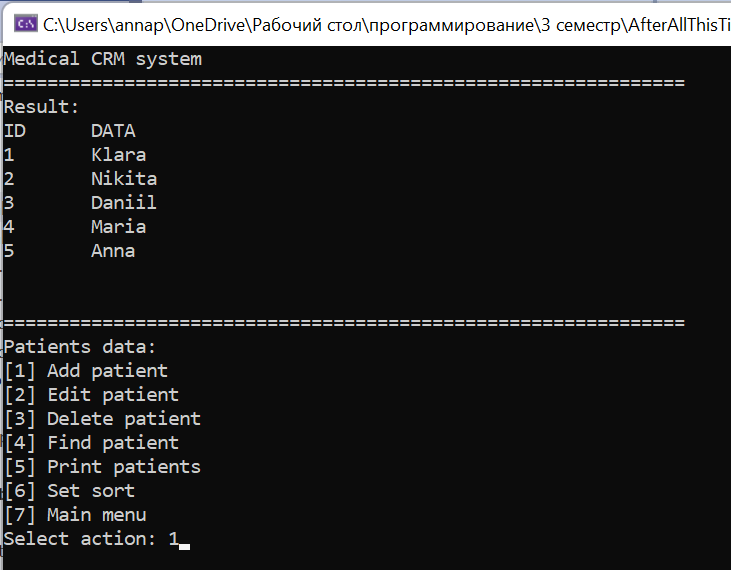
****

Рисунок 35 – Вывод всех пациентов

Вводим имя нового пациента «Yulia» и нажимаем клавишу «Enter» (рис. 36).

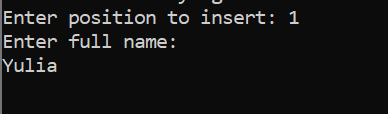


Рисунок 36 – Добавление нового пациента

Автоматически возвращаемся в меню, где видим сообщение об успешном добавление нового пациента (рис. 37).

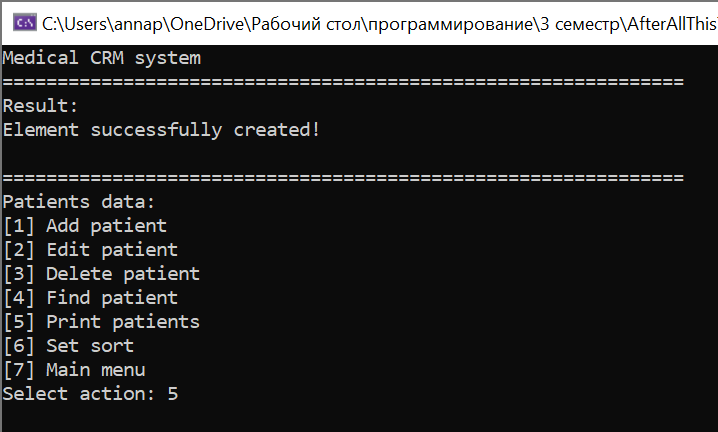


Рисунок 37 – Сообщение об успешном создании пациента

Выведем список всех пациентов. Выберем пункт № 5 (рис. 37). Новому элементу присвоился ID с № 6 (рис. 38).

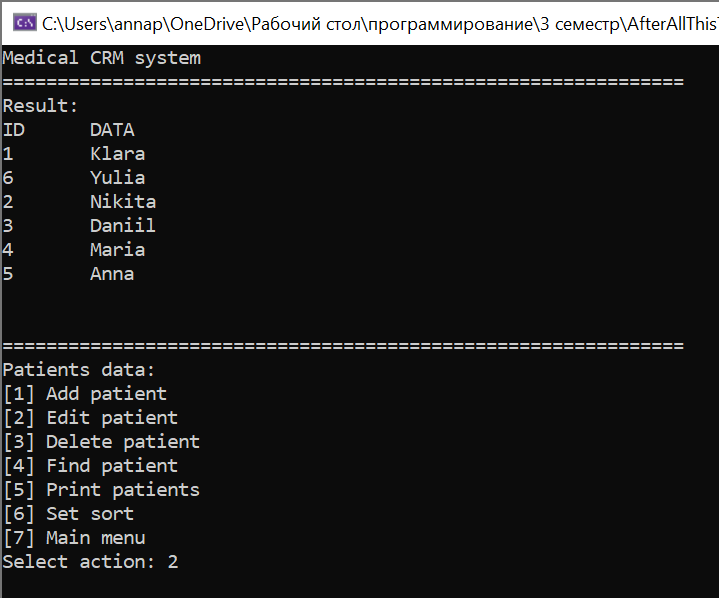


Рисунок 38 – Список пациентов с новым элементом

Отредактируем нового пациента. Выберем пункт меню под № 2 (рис. 38). Введём ID № 6 и изменим имя на «Eva» (рис. 39).

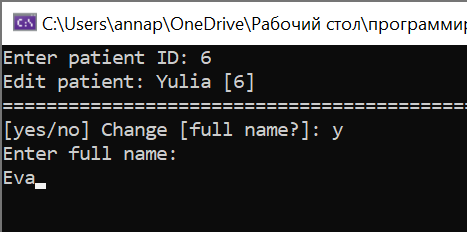


Рисунок 39 – Редактирование пациента

Видим сообщение об успешном обновление пациента (рис. 40).

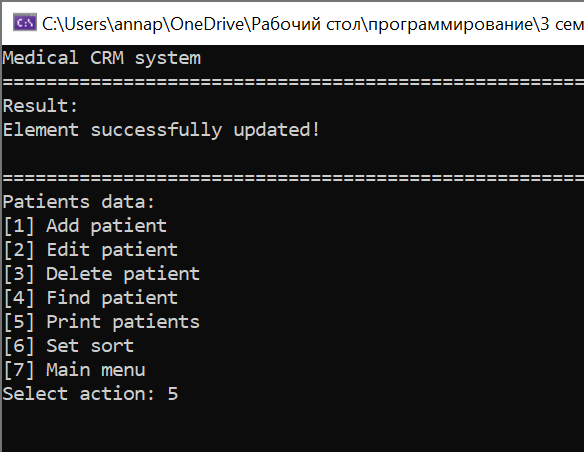


Рисунок 40 – Сообщение об успешном обновлении пациента

Выведем список элементов (рис. 40) и убедимся, что данные обновились (рис. 41).

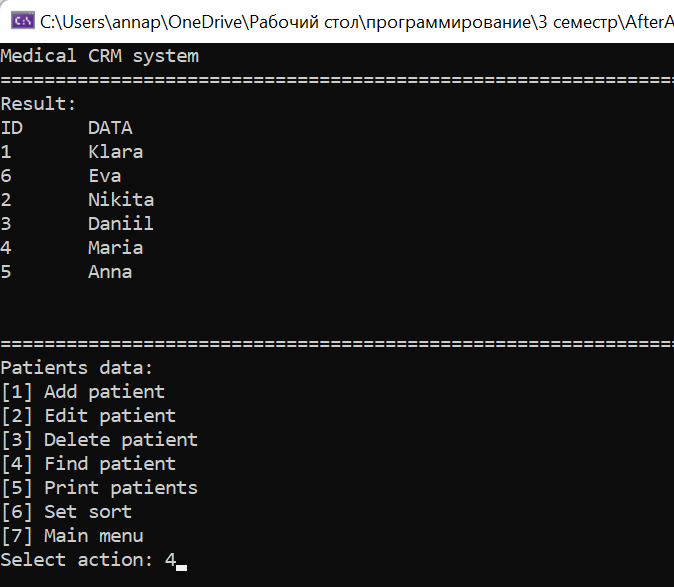


Рисунок 41 – Список пациентов с отредактированным элементом

Проверим операцию поиска. Выберем пункт меню под № 4 и введём поисковую подстроку (рис. 42).

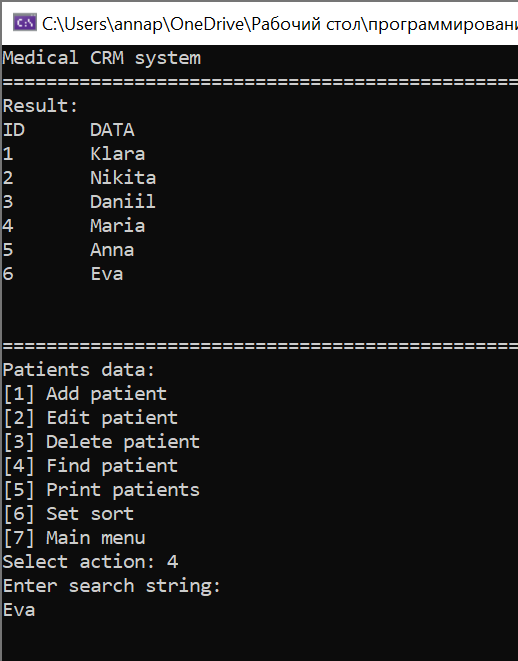


Рисунок 42 – Поиск пациентов

Результат поиска отобразит список найденных совпадений (рис. 43).

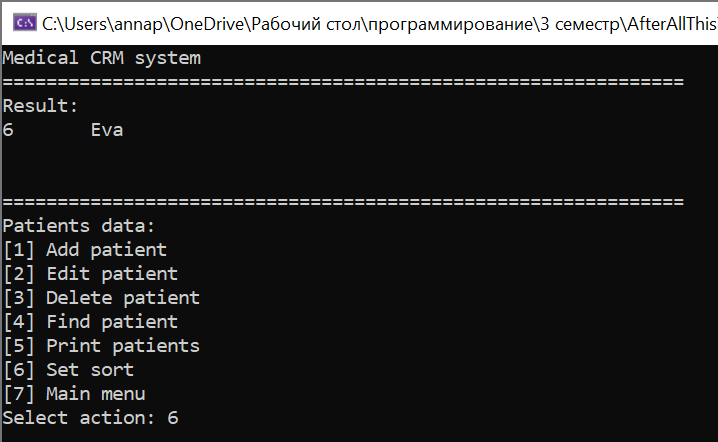


Рисунок 43 – Результат поиска

Сделаем сортировку (рис. 44).

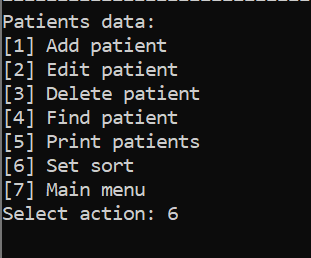


Рисунок 44 – Выбор пункта меню сортировка

Выбираем сортировку по ФИО (пункт 2 в подменю) (рис. 45).

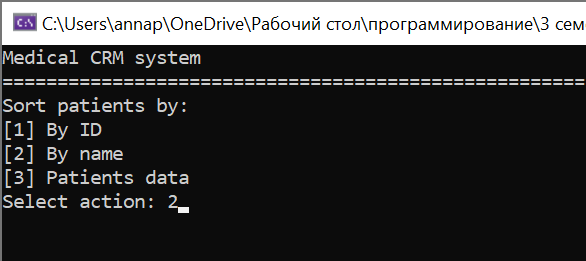


Рисунок 45 – Выбор сортировки

Видим на рис. 46 результат программы.

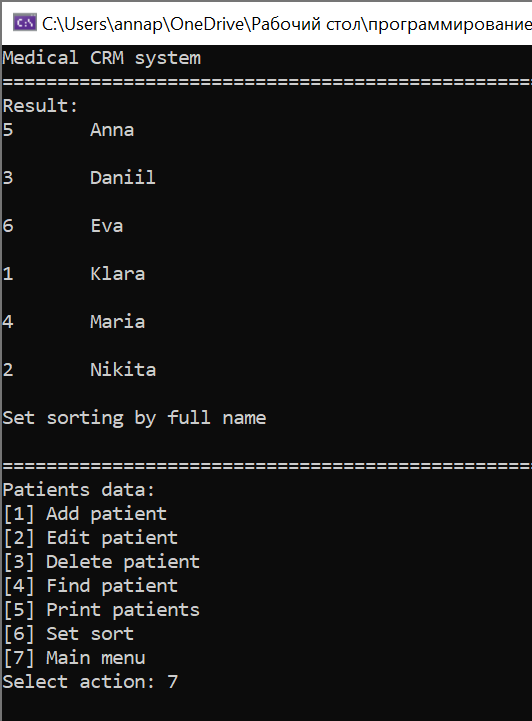


Рисунок 46 – Результат сортировки

Вернёмся в главное меню (рис. 46).

Перейдём в подменю **запись на приём** «Receptions data». Для этого выберем пункт главного меню под № 3 (рис. 47).

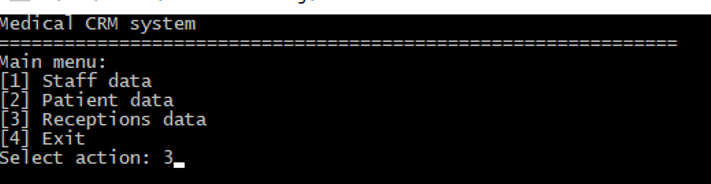


Рисунок 47 – Переход в подменю запись на приём «Receptions data»

Выведем содержимое файла receptions.txt на экран, для этого выберем пункт меню номер 7 (рис. 48), а затем пункт номер 5 (рис. 49).

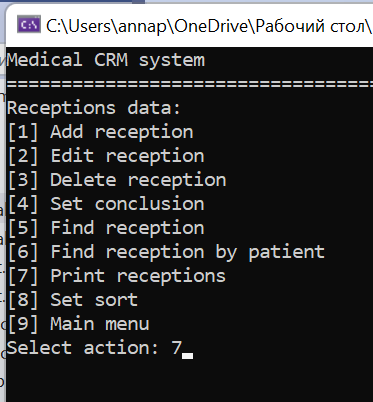


Рисунок 48 – Вывод списка записей

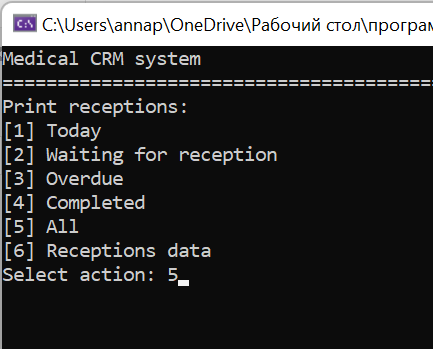


Рисунок 49 – Выбор вывода всех записей

Видим список записей на экране(рис. 50). Добавим новую запись. Выберем пункт меню под № 1 (рис. 50).

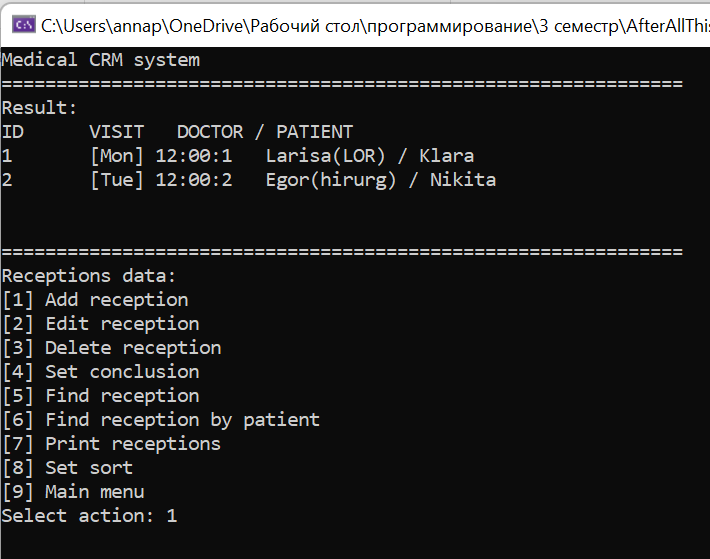


Рисунок 50 – Список всех записей

Заполним данные по приёму и выберем удобное время посещения из доступных вариантов (рис. 51).

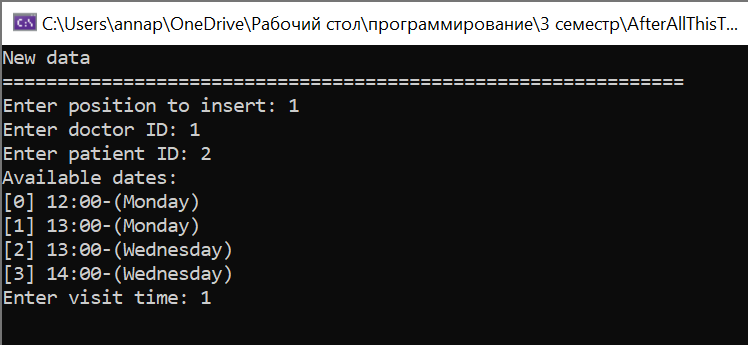


Рисунок 51 – Создание записи на приём

Данные, введенные с консоли:

1)Позиция для вставки: после 1 элемента

2)ID доктора: 1

3)ID пациента: 2

4) Время визита: 13:00 Понедельник

Автоматически возвращаемся в меню и видим сообщение об успешном создании записи на приём (рис. 52).

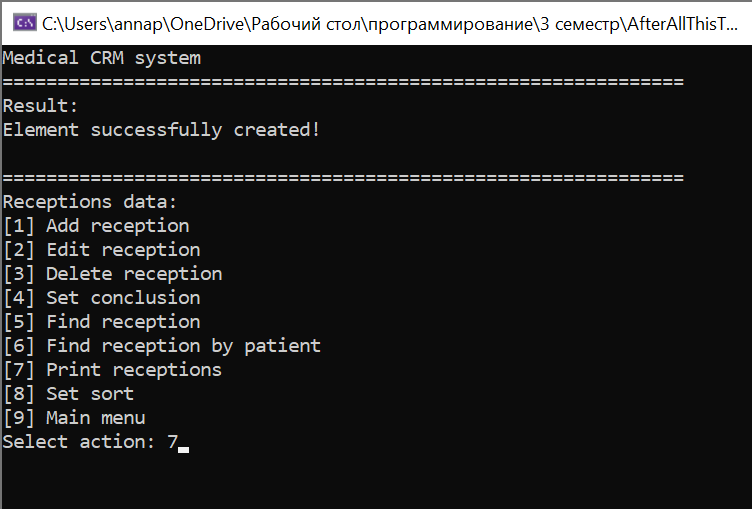


Рисунок 52 – Сообщение успешного создания записи на приём

Выведем список всех записей. Выберем пункт № 7 (рис. 52) и перейдём в подменю вывода данных «Print receptions». Выберем вывод всех элементов «All» под № 5 (рис. 53).

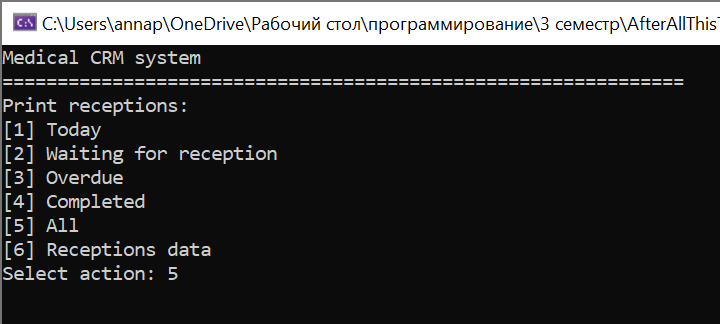


Рисунок 53 – Вывод всех записей на приём

Видим результат вывода с новой записью, с ID под № 3 (рис. 54).

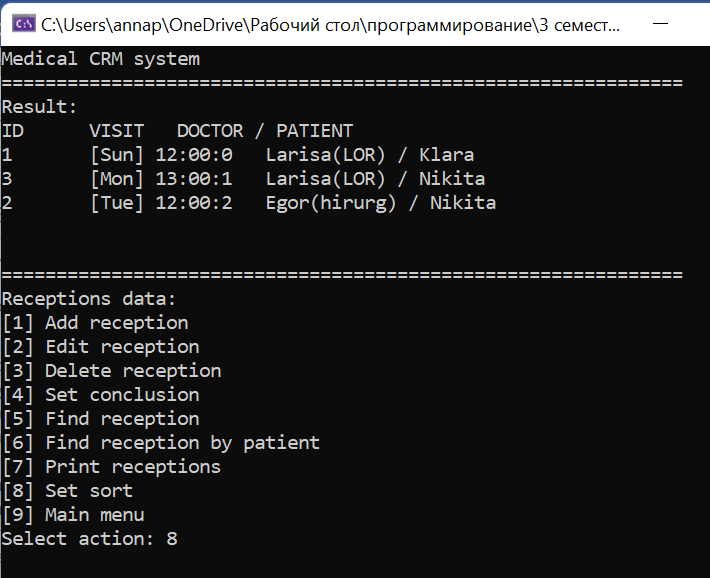


Рисунок 54 – Вывод всех записей

Изменим сортировку списка. Для этого перейдём в подменю сортировки «Set sort» под № 8 (рис. 54). Выберем сортировку по врачу «By doctor», пункт под № 2 (рис. 55).

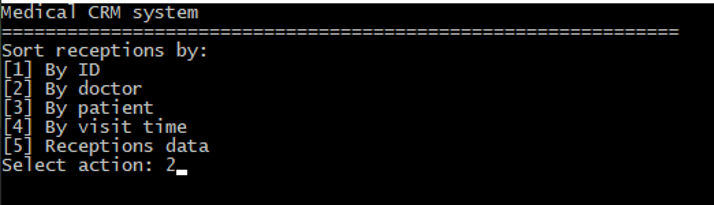


Рисунок 55 – Установка сортировки записей на приём по врачу

Видим результат работы сортировки. Записи отобразились в алфавитном порядке по ФИО врача (рис. 56).

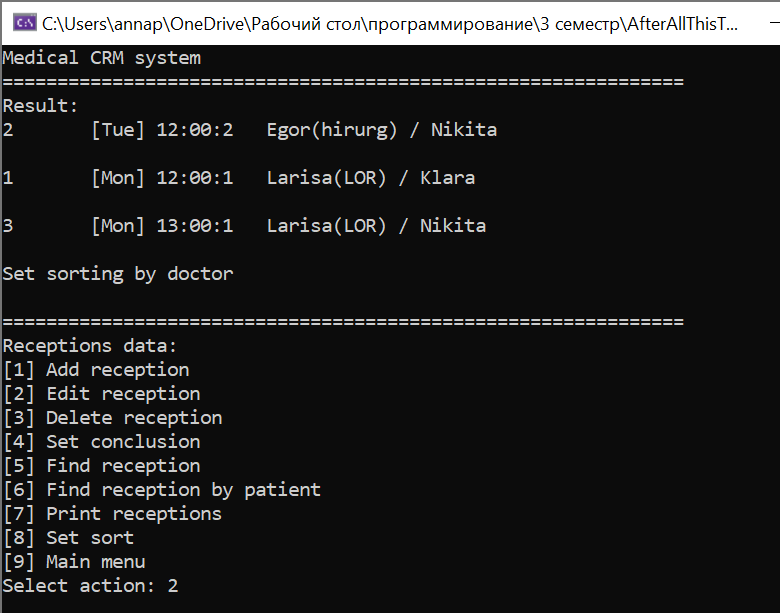


Рисунок 56 – Записи на приём отсортированные по врачу

Отредактируем запись на приём. Для этого выберем пункт меню под № 2 (рис. 57).

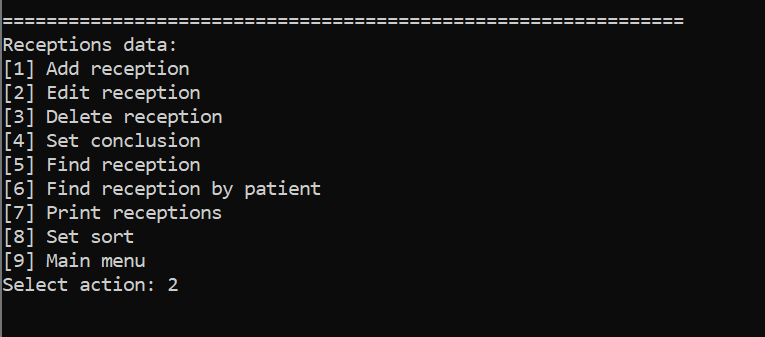


Рисунок 57 – Переход к редактированию элемента

Введём ID записи № 3, изменим врача (рис. 58).

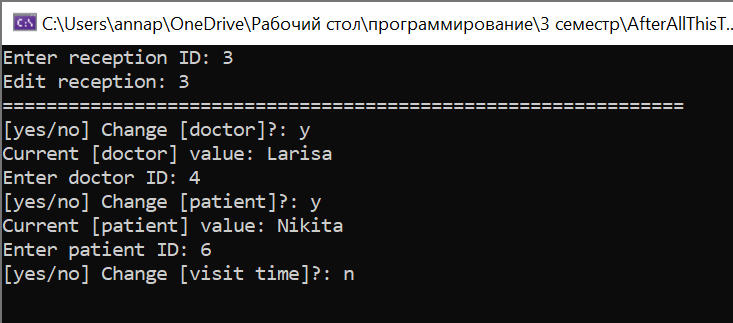


Рисунок 58 – Редактирование записи на приём

Видим сообщение об удачном редактировании (рис. 59). Выведем новый список записей выбрав пункт номер 7 (рис. 59), а затем номер 5 (рис. 60).

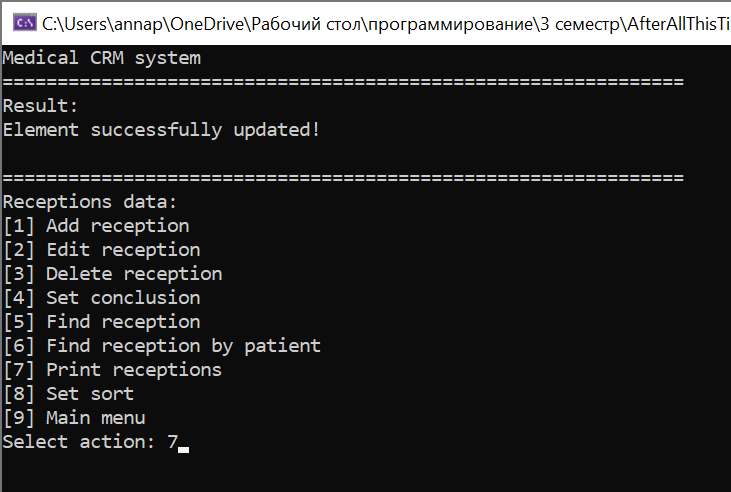


Рисунок 59 – Вывод на экран полученных записей

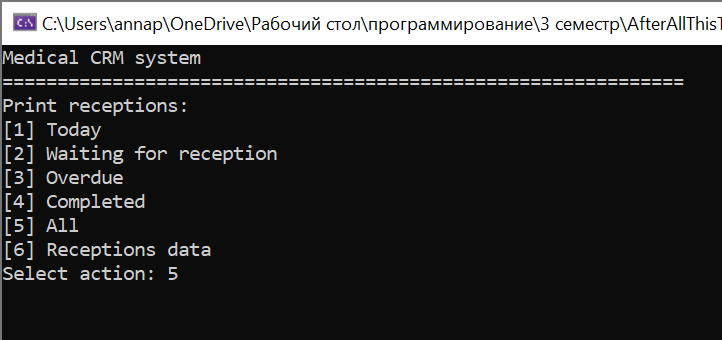


Рисунок 60 – Выбор всех записей

Видим успешно измененную запись (рис. 61).

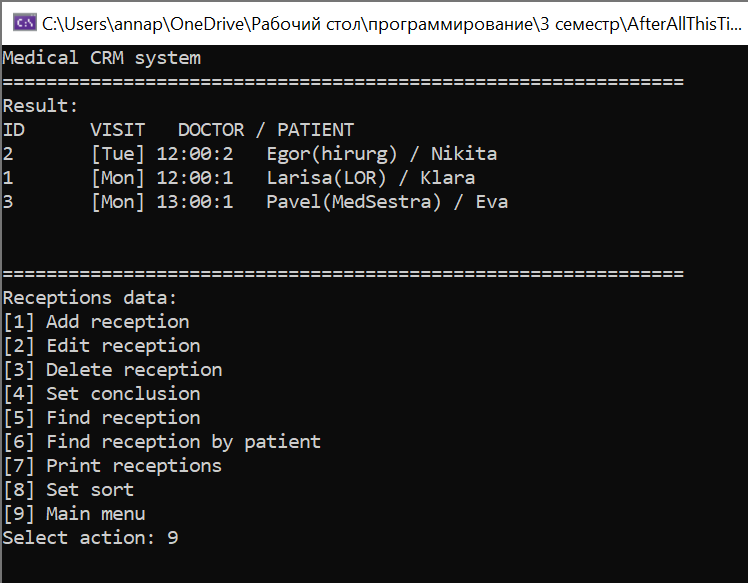


Рисунок 61 – Список записей на приём

Теперь удалим запись с ID под № 3, но не напрямую, а через удаление пациента «Eva». Вернёмся в главное меню. Выберем пункт меню под № 9 (рис. 60).

Перейдём в меню данные пациентов «Patients data». Выберем пункт меню под № 2 (рис. 62).

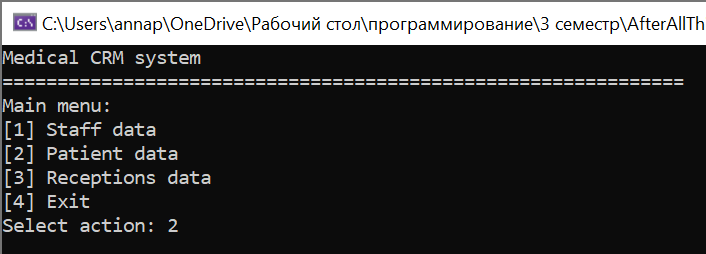


Рисунок 62 – Переход в Patient data

Выведем список пациентов, чтобы узнать нужный нам ID. Выберем пункт меню под № 5 (рис. 63).

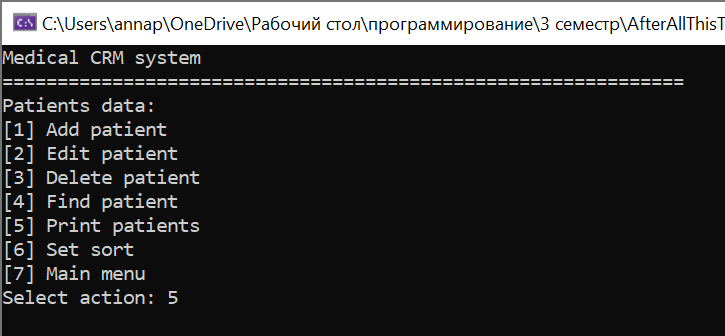


Рисунок 63 – Вывод списка пациентов

Удалим пациента. Для этого выберем пункт меню под № 3 (рис. 64).

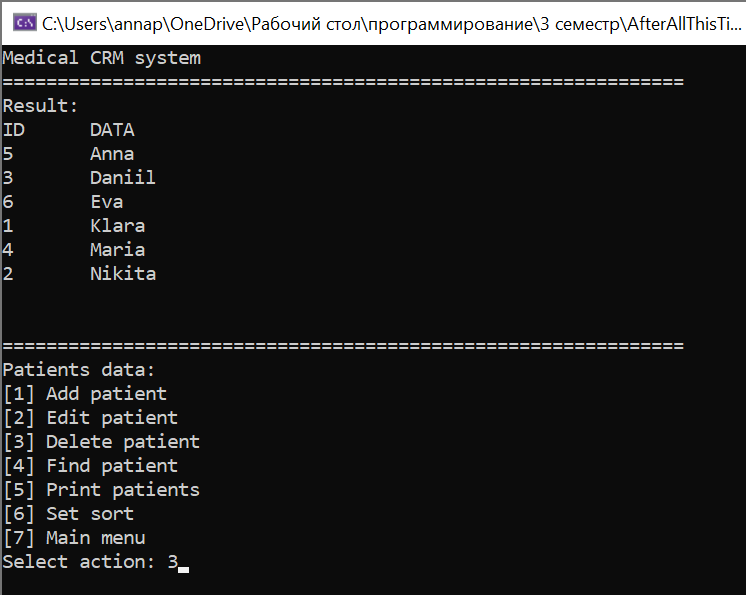


Рисунок 64 – Переход к удалению пациента

Введём ID № 6, пациента «Eva» (рис. 65) для удаления.

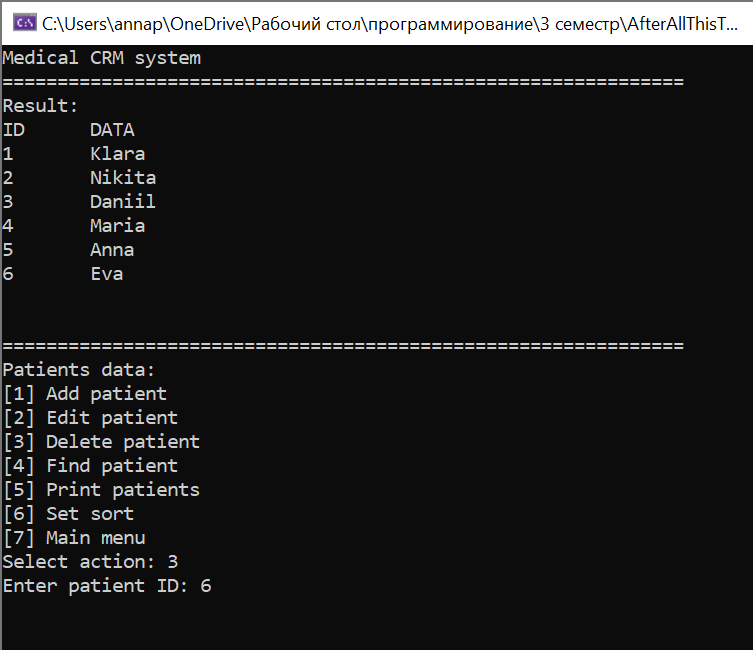


Рисунок 65 – Удаление пациента

Видим сообщение о том, что пациент и запись успешно удалены (рис. 66).

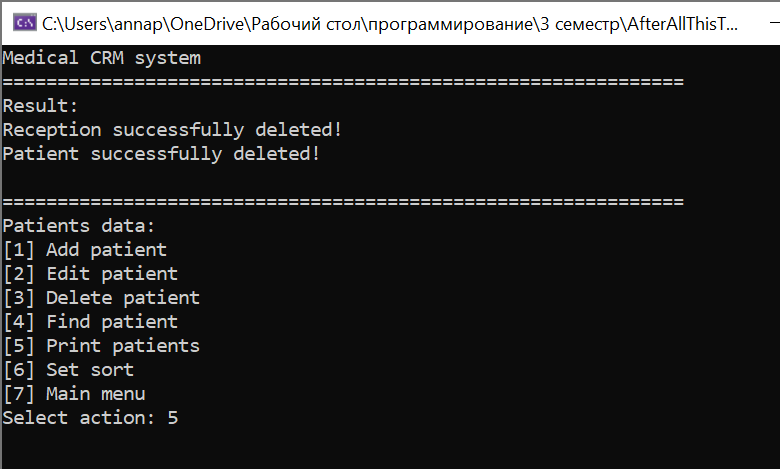


Рисунок 66 – Сообщение об успешном удалении пациента и записи

Выведем список пациентов (рис. 67), чтобы убедиться, что удаление сработало.

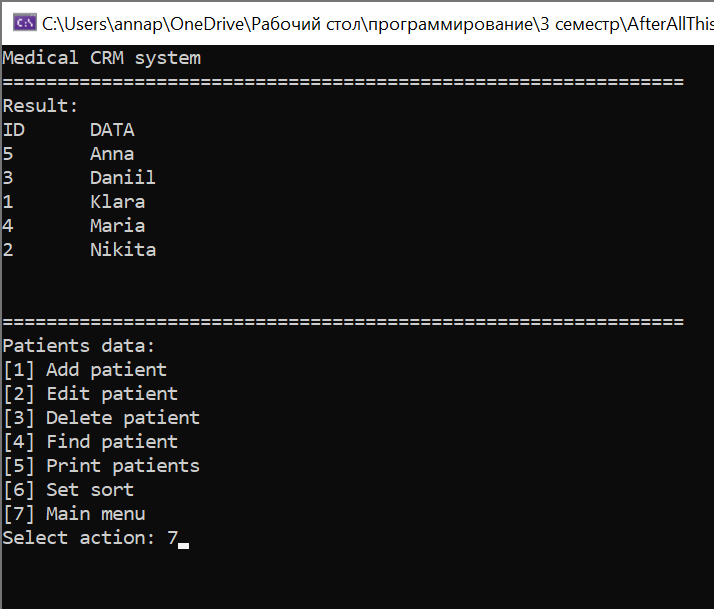


Рисунок 67 – Список пациентов после удаления элемента

Выберем пункт № 7, чтобы вернуться в главное меню (рис. 68).

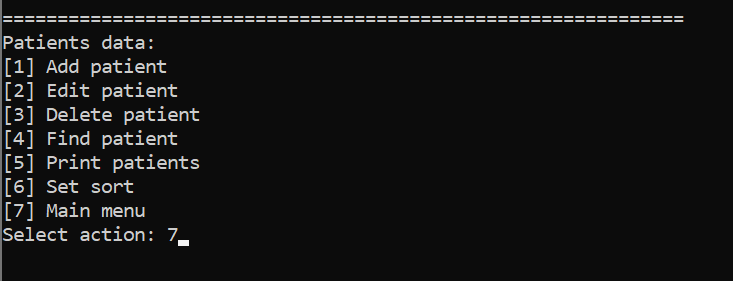


Рисунок 68 – Переход в главное меню

Выберем пункт меню № 3 и вернёмся в меню «Receptions data» (рис. 69).

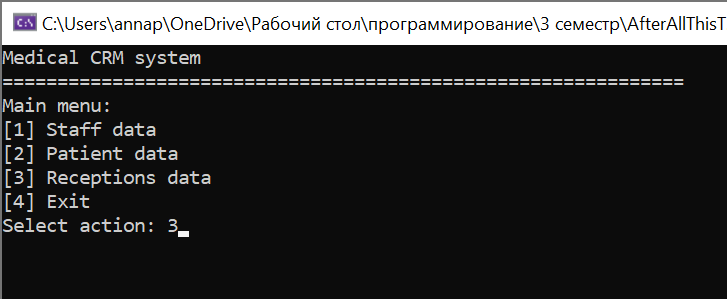


Рисунок 69 – Переход в меню «Receptions data»

Выведем список всех записей. Выберем пункт № 7 (рис. 70) в основном меню и пункт № 5 в подменю (рис. 71). Убедимся, что запись ID под № 3 удалена (рис. 72).

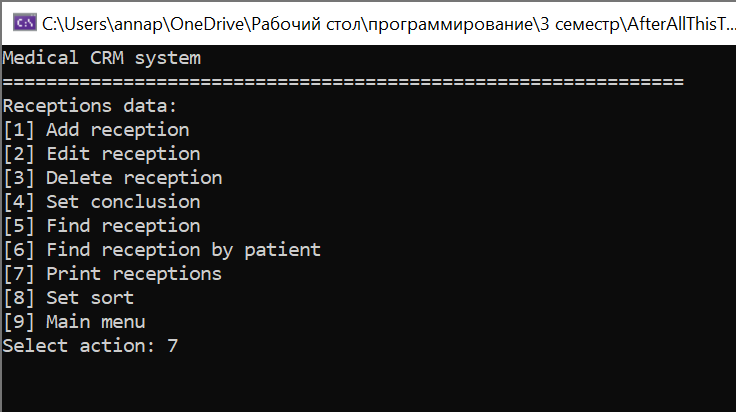


Рисунок 70 – Вывод на экран записей

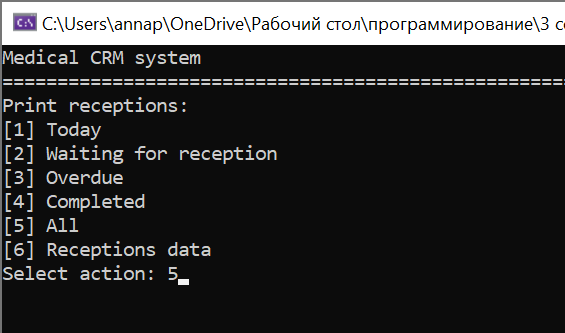


Рисунок 71 – Вывод всех записей на экран

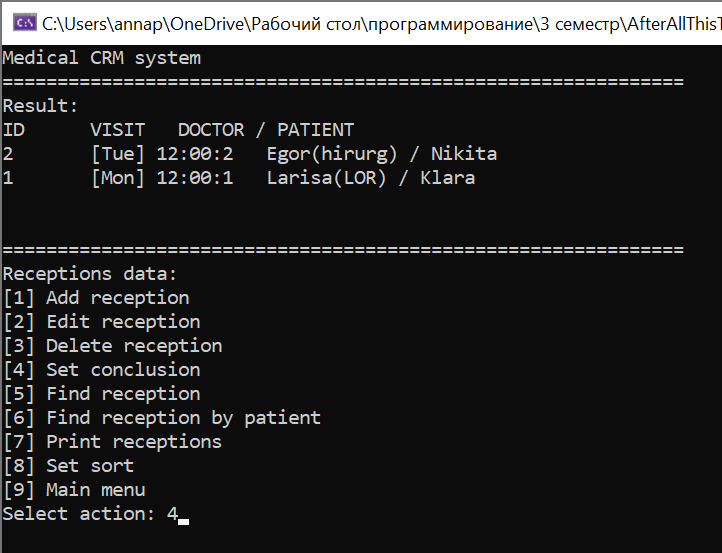


Рисунок 72 – Список записей на приём после удаления пациента

Добавим заключение врача в запись, для этого выберем пункт №4 (рис. 72).

Добавляем заключение к записи №2 (рис. 73).

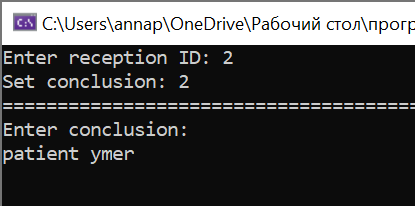


Рисунок 73 – Ввод текста в заключение

На рис. 74 видим, что элеменет успешно сохранился.

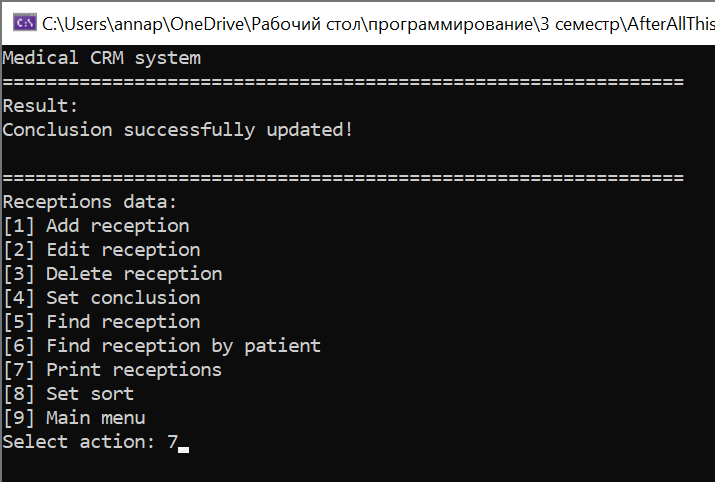


Рисунок 74 – сохранение элемента

Выберем пункт меню №7 (рис. 74) и выведем на экран все записи (пункт меню №5) (рис. 75).

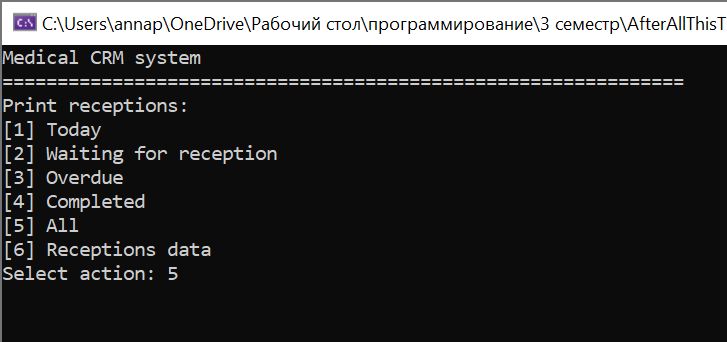


Рисунок 75 – Вывод всех записей

Видим, что заключение успешно установилось (рис. 76).

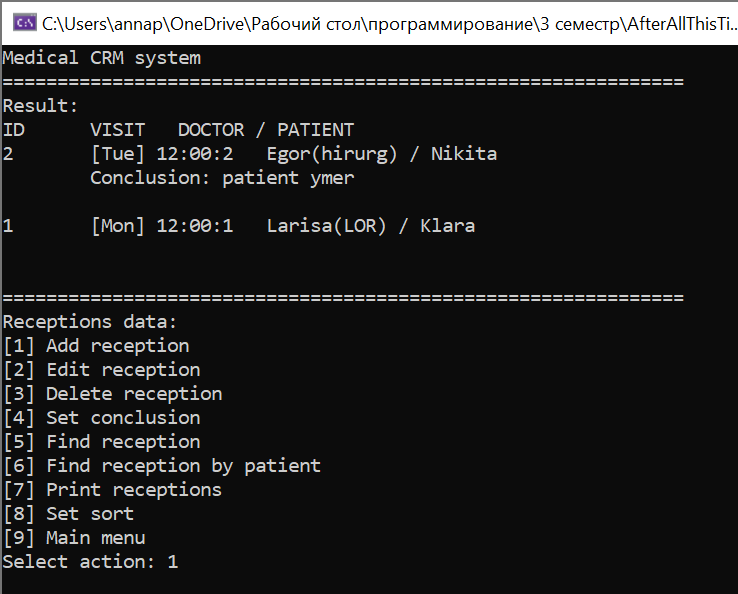


Рисунок 76 – Отредактированные записи

Проверим условие, что «каждый пациент в один день может быть записан не более чем к D врачам». Для этого попробуем записть пациента Klara в один день к нескольким врачам. Одна запись у нас уже есть (под номером 1). Переходим в пункт №1 (рис. 76), выбираем позицию для вставки и создаем новую запись (рис. 77).

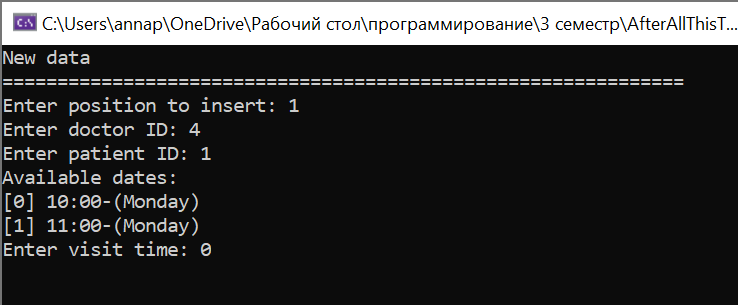


Рисунок 77 – Создание второй записи для пациента

Повторяем данную манипуляцию еще 2 раза только для разных врачей (рис. 78, рис. 79).

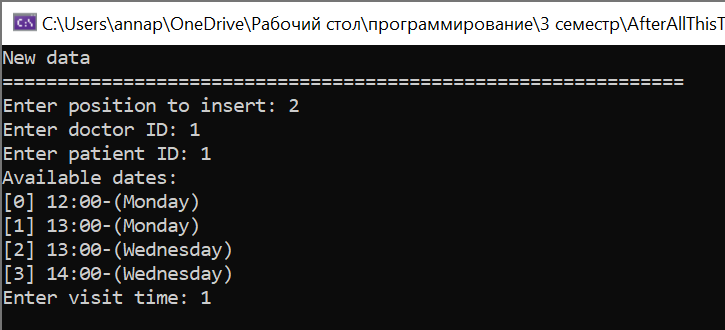


Рисунок 78 – Создание третьей записи для пациента

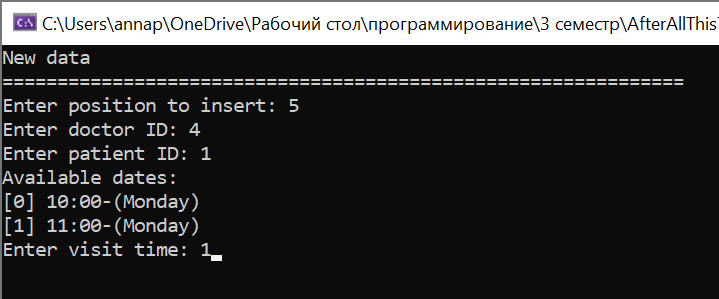


Рисунок 79 – Попытка создания четвертой записи для пациента

Видим на (рис.80), что у нас не получилось создать больше 3х записей в один день.

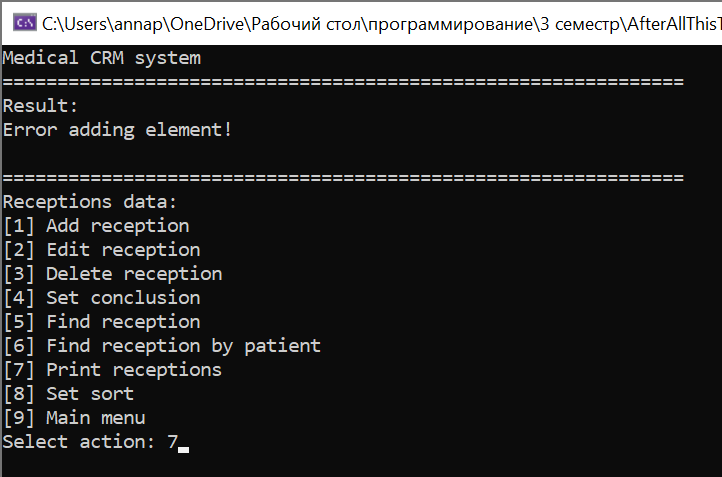


Рисунок 80 – Вывод сообщения о невозможности создать запись.

Выводим список записей на экран и смотрим результат (рис. 80), затем выбираем все записи(рис. 81).

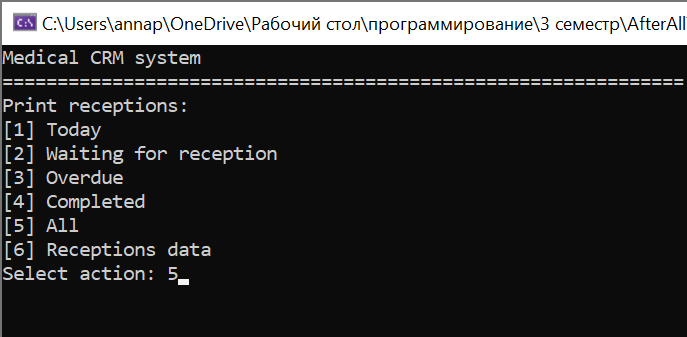
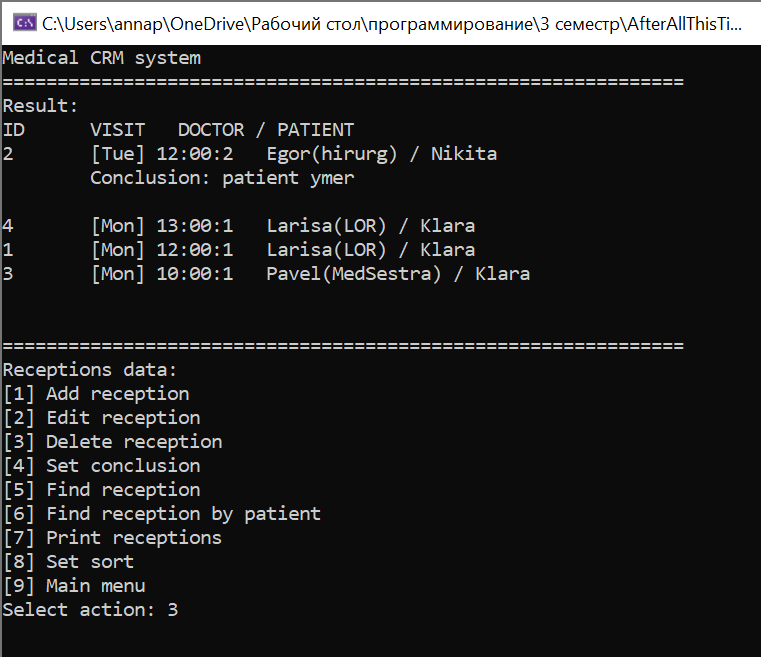


Рисунок 81 – Вывод всех записей

  
Рисунок 82 – Список всех записей

Удалим запись на приём по ее ID. Выбираем пункт меню № 3 (рис. 82 ), выбираем ID записи (рис. 83).

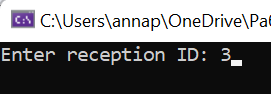


Рисунок 83 – Ввод ID

Видим сообщение об успешном удалении (рис. 84). Выведем список записей после удаления (рис. 84).

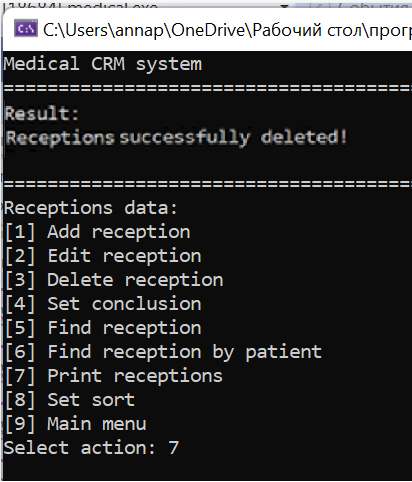


Рисунок 84 – Успешное удаление

Выберем пункт 5 (рис. 85).

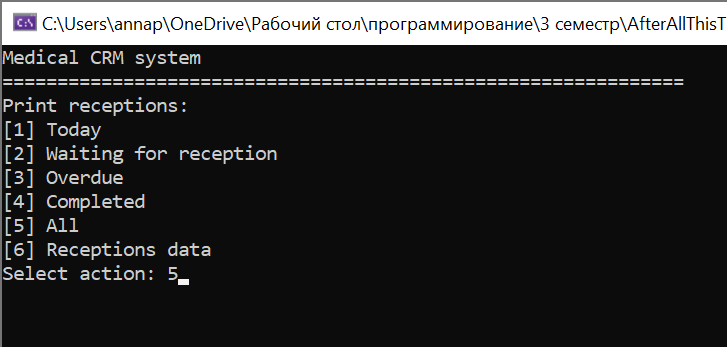


Рисунок 85 – Вывод всех записей

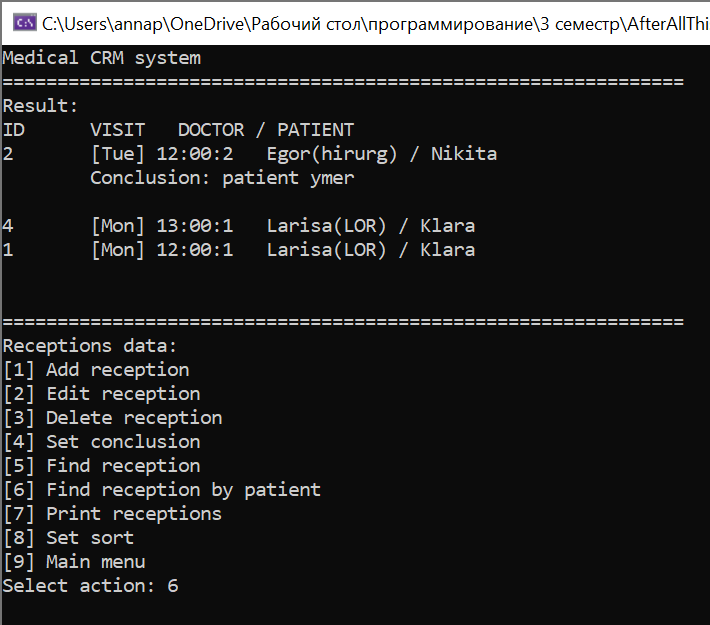


Рисунок 86 – Вывод всех записей после удаления

Попробуем найти запись по имени пациента, для этого выберем пункт 6 (рис. 87).

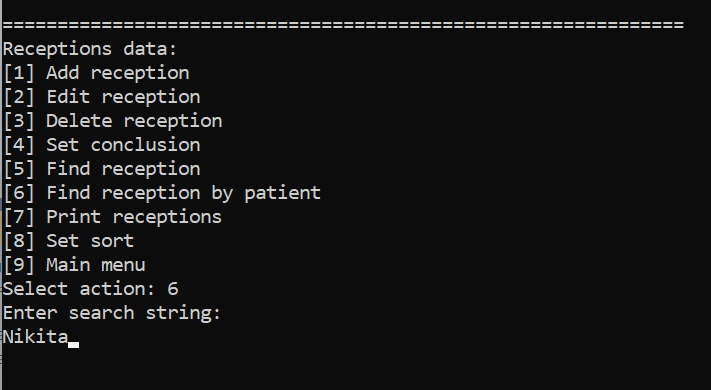


Рисунок 87 – Поиск по имени пациента

На рис. 88 мы видим, что программа отрабатывает верно.

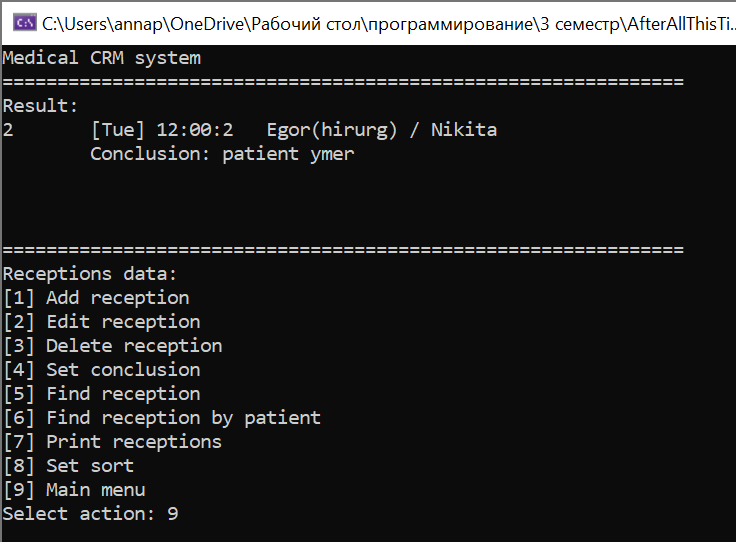


Рисунок 88 – Результат поиска по пациенту

Вернёмся в главное меню (рис. 88) и выйдем из программы (рис. 89).

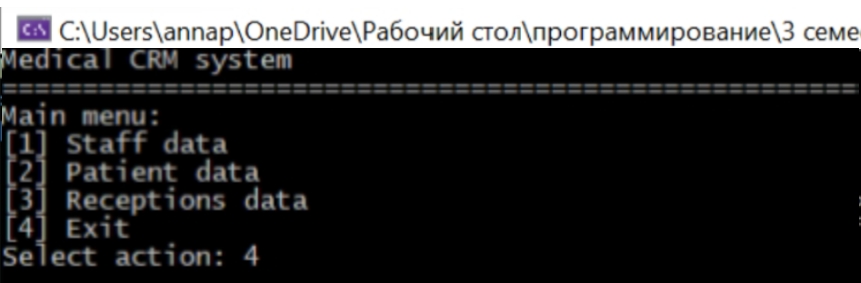


Рисунок 89 – Выход из программы

Под конец проверим текстовые файлы, в которые сохраняются данные между запусками программы. Как видно на рис. 90, 91, 92, все изменения были зафиксированы в файлах.

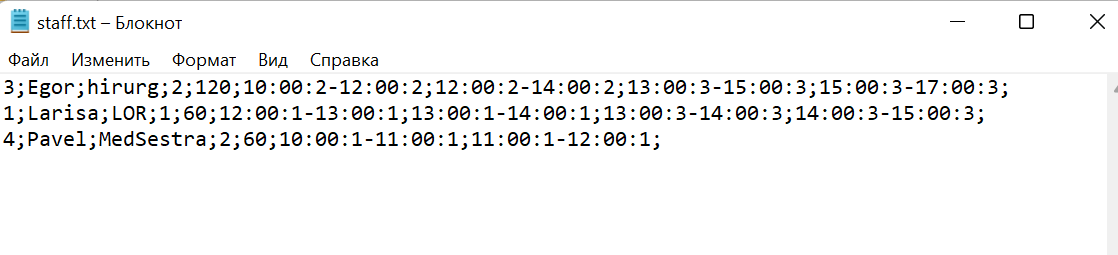


Рисунок 90 – Итоговые данные в файле staff.txt

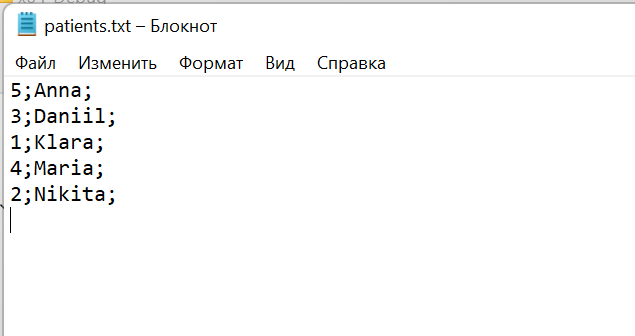


Рисунок 91 – Итоговые данные в файле patient.txt

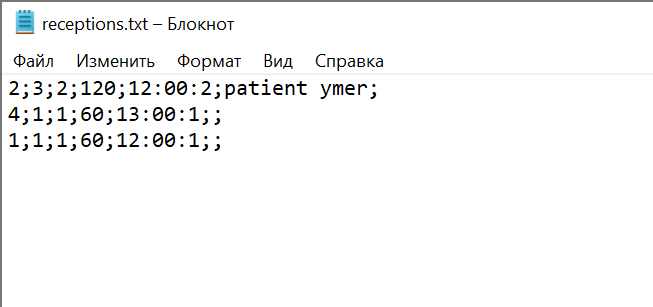


Рисунок 92 – Итоговые данные в файле receptions.txt

# ЗАКЛЮЧЕНИЕ

В данной курсовой работе была разработана программа, выполняющая функцию электронной регистратуры для медицинского учреждения.

С помощью интерактивного меню пользователь может создавать, редактировать и удалять данные о врачах, пациентах и записях на приём. Реализована возможность поиска данных по подстроке и сортировки по выбранному критерию.

Между записями программы данные сохраняются в текстовых файлах.

Взаимодействие с записями построено на структуре данных, однонаправленного линейного списка.

В ходе выполнения работы удалось улучшить понимания процессов работы со структурами данных типа список в языке C++.

# СПИСОК используемых источников

1. Страуструп Б. Язык программирования C++. Специальное издание. / М.: Бином; СПб.: Невский диалект, 2008. 105 - 266 с.
2. Документация по Visual Studio  - Microsoft Docs – URL:

[https:-docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2022&preserve-view=true](https://docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2022&preserve-view=true) (дата обращения: 20.12.2021 г. ).

1. struct tm - cplusplus.com. – URL: [https:-www.cplusplus.com/reference/ctime/tm/](https://www.cplusplus.com/reference/ctime/tm/) (дата обращения: 20.12.2021 г.).
2. Связный список - Wikipedia – URL: [https:-en.wikipedia.org/wiki/Linked\_list](https://en.wikipedia.org/wiki/Linked_list) (дата обращения: 20.12.2021 г.).

# Приложение 1. Листинг программного кода

main.cpp

#include "src/medical.h"

int main() {

mdc::loadData();

mdc::loopMenu();

mdc::clearMemory();

return 0;

}

utils.h

#include "string"

#ifndef MEDICAL\_UTILS\_H

#define MEDICAL\_UTILS\_H

using namespace std;

namespace utl {

const char STORAGE\_SEPARATOR = ';';

const char HR\_LINE[] = "==============================================================";

const char FIELD\_ERROR\_TEMPLATE[] = "Incorrect field value: [%s] %s";

const char TIME\_FORMAT[] = "%H:%M:%w";

- const char TIME\_FORMAT[] = "%H:%M:%A";

const char DATETIME\_FORMAT[] = "%d.%m.%Y %H:%M";

- const char DATETIME\_FORMAT[] = "%d.%m.%Y";

const string WEEK\_DAYS[7] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};

const string WEEK\_DAYS\_SHORT[7] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};

const int MAX\_INPUT\_TRIES = 4;

void clearScreen();

bool readAnswerFromConsole(const char title[]);

int readNumberFromConsole(const char title[]);

string readWordFromConsole(const char title[]);

string readLineFromConsole(const char title[]);

string getWeekDayName(int index, bool abbreviation = false);

tm \*getCurrDateTime();

tm \*readTimeFromConsole(const char fieldName[]);

bool readLineFromConsoleTo(const char fieldName[], string &field);

bool readNumberFromConsoleTo(const char fieldName[], int &field, int min, int max);

string strToLower(string &s);

bool isMenuMessage();

void setMenuMessage(const char title[]);

void clearMenuMessage();

void printMenuMessage();

void parseStorageStringToTime(string &storageStr, tm &start, tm &end);

void parseStorageStringDateTime(string &storageStr, tm &dt);

string reprTimePair(tm \*start, tm \*end);

string reprDateTime(tm \*dt);

bool isCorrectTimePair(tm \*start, tm \*end, int duration);

bool isDTGreaterThan(tm \*a, tm \*b);

bool isSameDay(tm \*a, tm \*b);

}

#endif -MEDICAL\_UTILS\_H

utils.cpp

#include <cstdlib>

#include <iostream>

#include <string>

#include <sstream>

#include <iomanip>

#include <ctime>

#include "utils.h"

using namespace std;

namespace utl {

string operationMessage;

void clearScreen() {

cout << string(10, '\n');

system("cls");

}

bool readAnswerFromConsole(const char title[]) {

cout << "[yes/no] ";

string answer = readWordFromConsole(title);

return tolower(answer[0]) == 'y';

}

int readNumberFromConsole(const char title[]) {

int tries = 0;

int buffer;

string input;

while (tries < MAX\_INPUT\_TRIES) {

cout << title << ": ";

cin >> input;

try {

buffer = stoi(input);

return buffer;

} catch (invalid\_argument &err) {

tries++;

cout << "Not a number. Try again!" << endl;

} catch (out\_of\_range &err) {

tries++;

cout << "Value is overflow. Try again!" << endl;

}

}

return NULL;

}

string readWordFromConsole(const char title[]) {

cout << title << ": ";

string line;

cin >> line;

return line;

}

string readLineFromConsole(const char title[]) {

cout << title << ":" << endl;

int tries = 0;

string line;

do {

getline(cin, line);

tries++;

} while (line.empty() || tries >= MAX\_INPUT\_TRIES);

return line;

}

string getWeekDayName(int index, bool abbreviation) {

string name;

if (abbreviation) {

name = WEEK\_DAYS\_SHORT[index];

} else {

name = WEEK\_DAYS[index];

}

return name;

}

tm \*getCurrDateTime() {

time\_t curr\_time{};

time(&curr\_time);

tm \*curr\_tm = localtime(&curr\_time);

return curr\_tm;

}

tm \*readTimeFromConsole(const char fieldName[]) {

tm \*buffer = new tm;

string value;

stringstream ss;

cout << "Please, enter the ";

ss << readWordFromConsole(fieldName);

ss >> std::get\_time(buffer, TIME\_FORMAT);

if (ss.fail()) {

cout << "Error reading time" << endl;

delete buffer;

return nullptr;

}

tm \*curr\_tm = new tm;

\*curr\_tm = \*getCurrDateTime();

curr\_tm->tm\_hour = buffer->tm\_hour;

curr\_tm->tm\_min = buffer->tm\_min;

delete buffer;

return curr\_tm;

}

bool readLineFromConsoleTo(const char fieldName[], string &field) {

char buffer[150];

string detail, label;

label += "Enter ";

label += fieldName;

string value = utl::readLineFromConsole(label.c\_str());

string::size\_type pos = value.find(STORAGE\_SEPARATOR);

if (value.empty()) {

detail = "is empty";

} else if (pos != string::npos) {

detail = "contain restrict char [";

detail += STORAGE\_SEPARATOR;

detail += ']';

}

if (detail.empty()) {

field = value;

return true;

}

sprintf(buffer, FIELD\_ERROR\_TEMPLATE, fieldName, detail.c\_str());

utl::setMenuMessage(buffer);

return false;

}

bool readNumberFromConsoleTo(const char fieldName[], int &field, int min, int max) {

char buffer[150];

string detail, label;

label += "Enter ";

label += fieldName;

int number = utl::readNumberFromConsole(label.c\_str());

if (number < min) {

detail = "less than " + to\_string(min);

} else if (number > max) {

detail = "greater than " + to\_string(max);

}

if (detail.empty()) {

field = number;

return true;

}

sprintf(buffer, FIELD\_ERROR\_TEMPLATE, fieldName, detail.c\_str());

utl::setMenuMessage(buffer);

return false;

}

string strToLower(string &s) {

string buffer;

for (char c : s) {

buffer += tolower(c);

}

return buffer;

}

bool isMenuMessage() {

return !operationMessage.empty();

}

void setMenuMessage(const char title[]) {

operationMessage += title;

operationMessage += '\n';

}

void clearMenuMessage() {

operationMessage.clear();

}

void printMenuMessage() {

cout << "Result: " << endl << operationMessage << endl;

cout << utl::HR\_LINE << endl;

clearMenuMessage();

}

void parseStorageStringToTime(string& storageStr, tm& start, tm& end) {

stringstream startStr, endStr;

- int day\_week = (storageStr[11] - '0');

int day\_week = storageStr[6] - '0';

-startStr << storageStr.substr(0, 7);

startStr << storageStr.substr(0, 7);

-endStr << storageStr.substr(8);

endStr << storageStr.substr(8, 7);

tm \*curr\_tm = getCurrDateTime();

start = \*curr\_tm;

end = \*curr\_tm;

startStr >> get\_time(&start, TIME\_FORMAT);

endStr >> get\_time(&end, TIME\_FORMAT);

mktime(&start);

mktime(&end);

start.tm\_wday = day\_week;

end.tm\_wday = day\_week;

}

void parseStorageStringDateTime(string& storageStr, tm& dt) {

stringstream buffer;

int day\_week = storageStr[6] - '0';

dt = \*getCurrDateTime();

buffer << storageStr;

buffer >> get\_time(&dt, TIME\_FORMAT);

mktime(&dt);

dt.tm\_wday = day\_week;

}

string reprTimePair(tm \*start, tm \*end) {

stringstream repr;

repr << put\_time(start, TIME\_FORMAT) << '-' << put\_time(end, TIME\_FORMAT);

return repr.str();

}

string reprDateTime(tm \*dt) {

stringstream repr;

repr << put\_time(dt, TIME\_FORMAT);

return repr.str();

}

bool isCorrectTimePair(tm \*start, tm \*end, int duration) {

if (start == nullptr || end == nullptr) {

return false;

}

-stringstream startStr, endStr;

-- int day\_week = (storageStr[11] - '0');

-int day\_week = storageStr[6] - '0';

--startStr << storageStr.substr(0, 7);

-startStr << storageStr.substr(0, 7);

--endStr << storageStr.substr(8);

-endStr << storageStr.substr(8, 7);

/\*tm\* curr\_tm = getCurrDateTime();

start = \*curr\_tm;

end = \*curr\_tm;

startStr >> get\_time(&start, TIME\_FORMAT);

endStr >> get\_time(&end, TIME\_FORMAT);

mktime(&start);

mktime(&end);

start.tm\_wday = day\_week;

end.tm\_wday = day\_week;\*/

time\_t startTime = mktime(start);

time\_t endTime = mktime(end);

return difftime(endTime, startTime) >= (duration \* 60);

}

bool isDTGreaterThan(tm \*a, tm \*b) {

if (a == nullptr || b == nullptr) {

return false;

}

time\_t a\_time = mktime(a);

time\_t b\_time = mktime(b);

return difftime(a\_time, b\_time) > 0;

}

bool isSameDay(tm \*a, tm \*b) {

return (a->tm\_year == b->tm\_year && a->tm\_mon == b->tm\_mon && a->tm\_mday == b->tm\_mday);

}

}

staff.h

#include <string>

#include <ctime>

using namespace std;

#ifndef MEDICAL\_STAFF\_H

#define MEDICAL\_STAFF\_H

namespace staff {

struct WorkDay {

bool is\_day\_off = false;

tm \*start\_at = nullptr;

tm \*end\_at = nullptr;

WorkDay\* next = nullptr;

};

struct Doctor {

int id = NULL;

string full\_name;

string specialty;

int category\_id;

Doctor \*next = nullptr;

WorkDay \*workdays;

int duration = NULL;

};

struct DoctorList {

Doctor \*root = nullptr;

Doctor \*curr = nullptr;

};

const int MIN\_DURATION\_MIN = 15;

const int MAX\_DURATION\_MIN = 120;

const char STORAGE\_FILEPATH[] = "./staff.txt";

const int LIST\_SORT\_TYPE\_ID = 0;

const int LIST\_SORT\_TYPE\_NAME = 1;

const int LIST\_SORT\_TYPE\_SPECIALTY = 2;

const int LIST\_SORT\_TYPE\_CATEGORY = 3;

const string LIST\_SORT\_TYPES[] = {"ID", "full name", "specialty", "category"};

const string DOCTORS\_CATEGORIES[] = {"second", "first", "great"};

void addItem(); -3

void editItem(); -4

void deleteItem(); -4

void findItems(); -4

void printList(); -3

void setSort(int sortType); -4

void loadData(); -3

void saveData(); -3

void clearMemory(); -3-4

bool deleteItemByID(int id); -4

void sortList(); -4

bool isNeedSwapBySortType(Doctor \*curr, Doctor \*next); -4

int getLastID();

Doctor \*getLastNode();

bool findItemByID(); -4

Doctor \*getItemByID(int id);

string getListRepresentation();

string getItemRepresentation(Doctor \*node);

bool isNotOnVacation(Doctor \*node);

void deleteItemWeKnowID(int staffID);

}

#endif -MEDICAL\_STAFF\_H

staff.cpp

#include <iostream>

#include <fstream>

#include "./utils.h"

#include "./staff.h"

using namespace std;

namespace staff {

DoctorList\* nodeList;

int sortListIndex = LIST\_SORT\_TYPE\_ID;

void addItem() {

utl::clearScreen();

auto\* node = new Doctor;

cout << "Enter staff data" << endl;

cout << utl::HR\_LINE << endl;

int insert\_position = utl::readNumberFromConsole("Enter position to insert");

bool validFullName = utl::readLineFromConsoleTo("full name", node->full\_name);

bool validSpecialty = utl::readLineFromConsoleTo("specialty", node->specialty);

cout << "Available categories:" << endl;

for (int i = 0; i < 3; i++) {

cout << "[" << i << "] " << DOCTORS\_CATEGORIES[i] << endl;

}

bool validCategoryID = utl::readNumberFromConsoleTo("category ID", node->category\_id, 0, 2);

cout << "Set reception duration in minutes. From " << MIN\_DURATION\_MIN;

cout << " to " << MAX\_DURATION\_MIN << " minutes." << endl;

bool validDuration = utl::readNumberFromConsoleTo("reception duration", node->duration, MIN\_DURATION\_MIN, MAX\_DURATION\_MIN);

if (validFullName && validSpecialty && validCategoryID && validDuration) {

cout << "Enter workdays" << endl;

cout << utl::HR\_LINE << endl;

node->workdays = nullptr;

auto current\_wd = node->workdays;

for (int i = 0; i < 7; i++) {

cout << utl::getWeekDayName(i) << ':' << endl;

bool workday = utl::readAnswerFromConsole("Is workday?");

auto\* wd = new WorkDay;

wd->is\_day\_off = !workday;

if (workday) {

wd->start\_at = utl::readTimeFromConsole("start time");

wd->end\_at = utl::readTimeFromConsole("end time");

if (utl::isCorrectTimePair(wd->start\_at, wd->end\_at, node->duration)) {

time\_t startTime = mktime(wd->start\_at);

time\_t endTime = mktime(wd->end\_at);

int count = difftime(endTime, startTime) / 60 / node->duration;

for (int j = 0; j < count; j++)

{

auto\* new\_wd = new WorkDay;

new\_wd->next = nullptr;

new\_wd->start\_at = new tm;

new\_wd->start\_at->tm\_min = wd->start\_at->tm\_min + j \* node->duration;

new\_wd->start\_at->tm\_sec = wd->start\_at->tm\_sec;

new\_wd->start\_at->tm\_hour = wd->start\_at->tm\_hour;

new\_wd->start\_at->tm\_mday = wd->start\_at->tm\_mday;

new\_wd->start\_at->tm\_mon = wd->start\_at->tm\_mon;

new\_wd->start\_at->tm\_year = wd->start\_at->tm\_year;

new\_wd->start\_at->tm\_yday = wd->start\_at->tm\_yday;

new\_wd->start\_at->tm\_isdst = wd->start\_at->tm\_isdst;

new\_wd->start\_at->tm\_wday = wd->start\_at->tm\_wday;

mktime(new\_wd->start\_at);

new\_wd->start\_at->tm\_wday = i;

new\_wd->end\_at = new tm;

new\_wd->end\_at->tm\_min = wd->start\_at->tm\_min + (j + 1) \* node->duration;

new\_wd->end\_at->tm\_hour = wd->start\_at->tm\_hour;

new\_wd->end\_at->tm\_sec = wd->start\_at->tm\_sec;

new\_wd->end\_at->tm\_mday = wd->start\_at->tm\_mday;

new\_wd->end\_at->tm\_mon = wd->start\_at->tm\_mon;

new\_wd->end\_at->tm\_year = wd->start\_at->tm\_year;

new\_wd->end\_at->tm\_wday = wd->start\_at->tm\_wday;

new\_wd->end\_at->tm\_yday = wd->start\_at->tm\_yday;

new\_wd->end\_at->tm\_isdst = wd->start\_at->tm\_isdst;

mktime(new\_wd->end\_at);

new\_wd->end\_at->tm\_wday = i;

if (node->workdays == nullptr)

{

node->workdays = new\_wd;

current\_wd = node->workdays;

}

else {

current\_wd->next = new\_wd;

current\_wd = current\_wd->next;

}

}

}

}

}

Doctor\* tmp = nodeList->root;

node->id = getLastID() + 1;

if (tmp == nullptr)

{

nodeList->root = node;

nodeList->curr = node;

saveData();

utl::setMenuMessage("Element successfully created!");

}

else {

Doctor\* tmp = nodeList->root;

Doctor\* serchid = nodeList->root;

int schet = 0;

while (tmp != nullptr) {

if (tmp->id == insert\_position) {

schet++;

node->next = tmp->next;

tmp->next = node;

break;

}

tmp = tmp->next;

}

if (schet==0) {

delete node;

utl::setMenuMessage("Error adding element!");

}

else {

nodeList->curr = node;

saveData();

utl::setMenuMessage("Element successfully created!");

}

}

}

else {

delete node;

utl::setMenuMessage("Error adding element!");

}

}

void editItem() {

utl::clearScreen();

if (findItemByID()) {

Doctor\* node = nodeList->curr;

string buffer;

int bufferNum = NULL;

bool changed = false;

cout << "Edit staff: " << node->full\_name << " [" << node->id << ']' << endl;

cout << utl::HR\_LINE << endl;

cout << "Current [full name] value: " << node->full\_name << endl;

if (utl::readAnswerFromConsole("Change [full name]?")) {

if (utl::readLineFromConsoleTo("full name", buffer)) {

node->full\_name = buffer;

changed = true;

}

else {

cout << "Incorrect data! [full name] value does not change. " << endl;

}

buffer.clear();

}

cout << "Current [specialty] value: " << node->specialty << endl;

if (utl::readAnswerFromConsole("Change [specialty]?")) {

if (utl::readLineFromConsoleTo("specialty", buffer)) {

node->specialty = buffer;

changed = true;

}

else {

cout << "Incorrect data! [specialty] value does not change. " << endl;

}

buffer.clear();

}

cout << "Current [category] value: " << DOCTORS\_CATEGORIES[node->category\_id] << endl;

if (utl::readAnswerFromConsole("Change [category]?")) {

cout << "Available categories:" << endl;

for (int i = 0; i < 3; i++) {

cout << "[" << i << "] " << DOCTORS\_CATEGORIES[i] << endl;

}

if (utl::readNumberFromConsoleTo("category ID", bufferNum, 0, 2)) {

node->category\_id = bufferNum;

changed = true;

}

else {

cout << "Incorrect data! [category] value does not change. " << endl;

}

bufferNum = NULL;

}

cout << "Current [reception duration] value: " << node->duration << " minutes" << endl;

if (utl::readAnswerFromConsole("Change [reception duration]?")) {

cout << "Set reception duration in minutes. From " << MIN\_DURATION\_MIN;

cout << " to " << MAX\_DURATION\_MIN << " minutes." << endl;

if (utl::readNumberFromConsoleTo("reception duration", bufferNum, MIN\_DURATION\_MIN, MAX\_DURATION\_MIN)) {

node->duration = bufferNum;

changed = true;

}

else {

cout << "Incorrect data! [reception duration] value does not change. " << endl;

}

bufferNum = NULL;

}

if (utl::readAnswerFromConsole("Change [workdays]?")) {

changed = true;

cout << "Enter workdays" << endl;

cout << utl::HR\_LINE << endl;

auto current\_wd = node->workdays;

for (int i = 0; i < 7; i++) {

cout << utl::getWeekDayName(i) << ':' << endl;

bool workday = utl::readAnswerFromConsole("Is workday?");

auto\* wd = &(node->workdays[i]);

wd->is\_day\_off = !workday;

if (workday) {

wd->start\_at = utl::readTimeFromConsole("start time");

wd->end\_at = utl::readTimeFromConsole("end time");

if (utl::isCorrectTimePair(wd->start\_at, wd->end\_at, node->duration)) {

time\_t startTime = mktime(wd->start\_at);

time\_t endTime = mktime(wd->end\_at);

int count = difftime(endTime, startTime) / 60 / node->duration;

for (int j = 0; j < count; j++)

{

auto\* new\_wd = new WorkDay;

new\_wd->next = nullptr;

new\_wd->start\_at = new tm;

new\_wd->start\_at->tm\_min = wd->start\_at->tm\_min + j \* node->duration;

new\_wd->start\_at->tm\_sec = wd->start\_at->tm\_sec;

new\_wd->start\_at->tm\_hour = wd->start\_at->tm\_hour;

new\_wd->start\_at->tm\_mday = wd->start\_at->tm\_mday;

new\_wd->start\_at->tm\_mon = wd->start\_at->tm\_mon;

new\_wd->start\_at->tm\_year = wd->start\_at->tm\_year;

new\_wd->start\_at->tm\_yday = wd->start\_at->tm\_yday;

new\_wd->start\_at->tm\_isdst = wd->start\_at->tm\_isdst;

new\_wd->start\_at->tm\_wday = wd->start\_at->tm\_wday;

mktime(new\_wd->start\_at);

new\_wd->start\_at->tm\_wday = i;

new\_wd->end\_at = new tm;

new\_wd->end\_at->tm\_min = wd->start\_at->tm\_min + (j + 1) \* node->duration;

new\_wd->end\_at->tm\_hour = wd->start\_at->tm\_hour;

new\_wd->end\_at->tm\_sec = wd->start\_at->tm\_sec;

new\_wd->end\_at->tm\_mday = wd->start\_at->tm\_mday;

new\_wd->end\_at->tm\_mon = wd->start\_at->tm\_mon;

new\_wd->end\_at->tm\_year = wd->start\_at->tm\_year;

new\_wd->end\_at->tm\_wday = wd->start\_at->tm\_wday;

new\_wd->end\_at->tm\_yday = wd->start\_at->tm\_yday;

new\_wd->end\_at->tm\_isdst = wd->start\_at->tm\_isdst;

mktime(new\_wd->end\_at);

new\_wd->end\_at->tm\_wday = i;

if (node->workdays == nullptr)

{

node->workdays = new\_wd;

current\_wd = node->workdays;

}

else {

current\_wd->next = new\_wd;

current\_wd = current\_wd->next;

}

}

if (!utl::isCorrectTimePair(wd->start\_at, wd->end\_at, node->duration)) {

delete wd->start\_at;

delete wd->end\_at;

wd->is\_day\_off = true;

cout << "Incorrect values! Day was set like day off." << endl;

}

}

node->workdays[i] = \*wd;

}

}

}

}

if (changed) {

saveData();

utl::setMenuMessage("Element successfully updated!");

}

else {

utl::setMenuMessage("Nothing to change!");

}

}

else {

utl::setMenuMessage("Element does not found!");

}

}

void deleteItem() {

utl::clearScreen();

int staffID = utl::readNumberFromConsole("Enter staff ID");

-сохранить идишник для поиска записей удаляемых по этому доктору

if (staffID != NULL) {

-if (deleteItemByID(staffID)) {

if (deleteItemByID(staffID)) {

utl::setMenuMessage("Doctor successfully deleted!");

}

else {

utl::setMenuMessage("Doctor was not found!");

}

}

else {

utl::setMenuMessage("Incorrect ID!");

}

}

void deleteItemWeKnowID(int staffID) {

utl::clearScreen();

-сохранить идишник для поиска записей удаляемых по этому доктору

if (staffID != NULL) {

if (deleteItemByID(staffID)) {

utl::setMenuMessage("Doctor successfully deleted!");

}

else {

utl::setMenuMessage("Doctor was not found!");

}

}

else {

utl::setMenuMessage("Incorrect ID!");

}

}

void findItems() {

string search = utl::readLineFromConsole("Enter search string");

search = utl::strToLower(search);

Doctor\* node = nodeList->root;

string buffer;

while (node != nullptr) {

string representation = getItemRepresentation(node);

string representationLower = utl::strToLower(representation);

string::size\_type matchPos = representationLower.find(search);

if (matchPos != string::npos) {

buffer.append(representation);

}

node = node->next;

}

if (buffer.empty()) {

utl::setMenuMessage("Matches not found!");

}

else {

utl::setMenuMessage(buffer.c\_str());

}

}

void printList() {

string buffer = getListRepresentation();

utl::setMenuMessage(buffer.c\_str());

}

void setSort(int sortType) {

sortListIndex = sortType;

sortList();

Doctor\* temp = nodeList->root;

while (temp)

{

string message = getItemRepresentation(temp);

utl::setMenuMessage(message.c\_str());

if (!temp->next) {

break;

}

else {

temp = temp->next;

}

}

string message = "Set sorting by " + LIST\_SORT\_TYPES[sortListIndex];

utl::setMenuMessage(message.c\_str());

}

void loadData() {

nodeList = new DoctorList;

ifstream storage(STORAGE\_FILEPATH);

if (storage) {

string line;

Doctor\* node, \* prev = nullptr;

while (getline(storage, line)) {

if (!line.empty()) {

node = new Doctor;

node->workdays = nullptr;

auto current = node->workdays;

short cell\_counter = 0;

string::size\_type lastSeparatorPosition = 0;

string::size\_type currentSeparatorPosition = line.find(utl::STORAGE\_SEPARATOR,

lastSeparatorPosition);

while (currentSeparatorPosition != string::npos) {

string value = line.substr(lastSeparatorPosition,

currentSeparatorPosition - lastSeparatorPosition);

switch (cell\_counter) {

case 0:

node->id = stoi(value);

break;

case 1:

node->full\_name = value;

break;

case 2:

node->specialty = value;

break;

case 3:

node->category\_id = stoi(value);

break;

case 4:

node->duration = stoi(value);

break;

default:

auto\* wd = new WorkDay;

wd->is\_day\_off = value.empty();

if (!wd->is\_day\_off) {

wd->start\_at = new tm;

wd->end\_at = new tm;

utl::parseStorageStringToTime(value, \*wd->start\_at, \*wd->end\_at);

}

if (node->workdays == nullptr)

{

node->workdays = wd;

current = node->workdays;

}

else {

current->next = wd;

current = current->next;

}

break;

}

cell\_counter++;

lastSeparatorPosition = currentSeparatorPosition + sizeof(utl::STORAGE\_SEPARATOR);

currentSeparatorPosition = line.find(utl::STORAGE\_SEPARATOR, lastSeparatorPosition);

}

if (prev != nullptr) {

prev->next = node;

}

prev = node;

if (nodeList->root == nullptr) {

nodeList->root = node;

}

nodeList->curr = node;

}

}

}

}

void saveData() {

ofstream storage(STORAGE\_FILEPATH);

Doctor\* node = nodeList->root;

while (node != nullptr) {

string line;

line.append(to\_string(node->id) + utl::STORAGE\_SEPARATOR);

line.append(node->full\_name + utl::STORAGE\_SEPARATOR);

line.append(node->specialty + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->category\_id) + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->duration) + utl::STORAGE\_SEPARATOR);

auto temp = node->workdays;

while (temp) {

WorkDay\* wd = temp;

if (!wd->is\_day\_off) {

line.append(utl::reprTimePair(wd->start\_at, wd->end\_at));

}

line += utl::STORAGE\_SEPARATOR;

temp = temp->next;

}

storage << line << endl;

node = node->next;

}

}

void clearMemory() {

Doctor\* node = nodeList->root;

while (node != nullptr) {

Doctor\* buffer = node->next;

-дописать ф-ция clearmemory для вордейс

while (node->workdays != nullptr) {

WorkDay\* sled = node->workdays->next;

delete node->workdays;

node->workdays = sled;

}

delete node;

node = buffer;

}

delete nodeList;

}

bool deleteItemByID(int id) {

Doctor\* node = nodeList->root;

Doctor\* prev = nullptr;

int schet = 0;

while (node != nullptr) {

if (node->id == id) {

schet += 1;

if (prev == nullptr) {

nodeList->root = node->next;

delete node->workdays;

delete node;

node = nodeList->root;

}

else {

prev->next = node->next;

delete node->workdays;

delete node;

node = prev->next;

}

-continue;

break;

}

prev = node;

node = node->next;

}

saveData();

if (schet == 0) {

return false;

}

else {

return true;

}

}

void sortList() {

bool unsorted = true;

Doctor\* curr, \* next, \* prev, \* buffer;

while (unsorted) {

unsorted = false;

curr = nodeList->root;

prev = nullptr;

while (curr) {

next = curr->next;

if (isNeedSwapBySortType(curr, next)) {

- Меняем значения местами.

swap(curr, next);

if (prev != nullptr) {

prev->next = curr;

}

else {

nodeList->root = curr;

}

buffer = curr->next;

curr->next = next;

next->next = buffer;

unsorted = true;

}

nodeList->curr = curr;

prev = curr;

curr = next;

}

}

}

bool isNeedSwapBySortType(Doctor\* curr, Doctor\* next) {

if (next == nullptr) {

return false;

}

switch (sortListIndex) {

case LIST\_SORT\_TYPE\_ID:

return curr->id > next->id;

case LIST\_SORT\_TYPE\_NAME:

return curr->full\_name > next->full\_name;

case LIST\_SORT\_TYPE\_SPECIALTY:

return curr->specialty > next->specialty;

case LIST\_SORT\_TYPE\_CATEGORY:

return DOCTORS\_CATEGORIES[curr->category\_id] > DOCTORS\_CATEGORIES[next->category\_id];

default:

return false;

}

}

int getLastID() {

int last\_id = 0;

Doctor\* node = nodeList->root;

while (node != nullptr) {

if (last\_id < node->id) {

last\_id = node->id;

}

node = node->next;

}

return last\_id;

}

Doctor\* getLastNode() {

Doctor\* prev = nullptr;

Doctor\* curr = nodeList->root;

while (curr != nullptr) {

prev = curr;

curr = curr->next;

}

return prev;

}

bool findItemByID() {

int staffID = utl::readNumberFromConsole("Enter staff ID");

Doctor\* node = getItemByID(staffID);

if (node != nullptr) {

nodeList->curr = node;

return true;

}

return false;

}

Doctor\* getItemByID(int id) {

Doctor\* node = nodeList->root;

while (node != nullptr) {

if (node->id == id) {

return node;

}

node = node->next;

}

return nullptr;

}

string getListRepresentation() {

Doctor\* node = nodeList->root;

string buffer;

buffer.append("ID\tDATA\n");

int counter = 0;

while (node != nullptr) {

string line = getItemRepresentation(node);

buffer.append(line);

counter++;

node = node->next;

}

if (counter == 0) {

buffer.clear();

buffer.append("Nothing to show...");

}

return buffer;

}

string getItemRepresentation(Doctor\* node) {

string buffer;

buffer.append(to\_string(node->id) + '\t');

buffer.append(node->full\_name);

buffer.append(" (" + node->specialty + '/' + DOCTORS\_CATEGORIES[node->category\_id] + ")\n");

buffer.append("\treception duration: " + to\_string(node->duration) + " minutes\n");

auto temp = node->workdays;

while (temp)

{

WorkDay\* wd = temp;

if (!wd->is\_day\_off) {

buffer += "\t> ";

buffer += utl::reprTimePair(wd->start\_at, wd->end\_at);

buffer += " [";

buffer += utl::getWeekDayName(wd->start\_at->tm\_wday, true);

buffer += "] <\n";

}

temp = temp->next;

}

/\*auto temp = node->workdays;

while (temp)

{

WorkDay\* wd = temp;

if (!wd->is\_day\_off) {

buffer += "\t> ";

buffer.append(to\_string(node->id) + '\t');

buffer += " [";

buffer += utl::reprTimePair(wd->start\_at, wd->end\_at);

buffer += "] <\n";

}

temp = temp->next;

}\*/

buffer += '\n';

return buffer;

}

bool isNotOnVacation(Doctor\* node) {

for (int i = 0; i < 7; i++) {

WorkDay\* wd = &(node->workdays[i]);

if (!wd->is\_day\_off) {

return true;

}

}

return false;

}

}

**patient.h**

#include <string>

using namespace std;

#ifndef MEDICAL\_PATIENTS\_H

#define MEDICAL\_PATIENTS\_H

namespace patient {

struct Pacient {

int id = NULL;

string full\_name;

Pacient \*next = nullptr;

};

struct PatientList {

Pacient \*root = nullptr;

Pacient \*curr = nullptr;

};

const char STORAGE\_FILEPATH[] = "./patients.txt";

const int LIST\_SORT\_TYPE\_ID = 0;

const int LIST\_SORT\_TYPE\_NAME = 1;

const string LIST\_SORT\_TYPES[] = {"ID", "full name"};

void addItem();

void editItem();

void deleteItem();

void findItems();

void printList();

void setSort(int sortType);

void loadData();

void saveData();

void clearMemory();

bool deleteItemByID(int id);

void sortList();

bool isNeedSwapBySortType(Pacient \*curr, Pacient \*next);

int getLastID();

Pacient \*getLastNode();

bool findItemByID();

Pacient \*getItemByID(int id);

string getListRepresentation();

string toString(Pacient \*node);

void deleteItemKnowID(int staffID);

}

#endif -MEDICAL\_PATIENTS\_H

**patients.cpp**

#include <iostream>

#include <fstream>

#include "./utils.h"

#include "patient.h"

using namespace std;

namespace patient {

PatientList \*nodeList;

int sortListIndex = LIST\_SORT\_TYPE\_ID;

void addItem() {

utl::clearScreen();

auto \*node = new Pacient;

cout << "New data" << endl;

cout << utl::HR\_LINE << endl;

int insert\_position = utl::readNumberFromConsole("Enter position to insert");

if (utl::readLineFromConsoleTo("full name", node->full\_name)) {

Pacient\* tmp = nodeList->root;

node->id = getLastID() + 1;

if (tmp == nullptr)

{

nodeList->root = node;

nodeList->curr = node;

saveData();

utl::setMenuMessage("Element successfully created!");

}

-вставка

else {

int schet = 0;

while (tmp != nullptr) {

if (tmp->id== insert\_position) {

schet++;

node->next = tmp->next;

tmp->next = node;

break;

}

tmp = tmp->next;

}

if (schet == 0) {

delete node;

utl::setMenuMessage("Error adding element!");

}

else {

nodeList->curr = node;

saveData();

utl::setMenuMessage("Element successfully created!");

}

}

}

else {

delete node;

utl::setMenuMessage("Error adding element!");

}

}

void editItem() {

utl::clearScreen();

if (findItemByID()) {

Pacient \*node = nodeList->curr;

string buffer;

bool changed = false;

cout << "Edit patient: " << node->full\_name << " [" << node->id << ']' << endl;

cout << utl::HR\_LINE << endl;

if (utl::readAnswerFromConsole("Change [full name?]")) {

if (utl::readLineFromConsoleTo("full name", buffer)) {

node->full\_name = buffer;

changed = true;

} else {

cout << "Incorrect data! [full name] value does not change. " << endl;

}

buffer.clear();

}

if (changed) {

saveData();

utl::setMenuMessage("Element successfully updated!");

} else {

utl::setMenuMessage("Nothing to change!");

}

} else {

utl::setMenuMessage("Error updating element!");

}

}

void deleteItem() {

utl::clearScreen();

int staffID = utl::readNumberFromConsole("Enter patient ID");

if (staffID != NULL) {

if (deleteItemByID(staffID)) {

utl::setMenuMessage("Patient successfully deleted!");

} else {

utl::setMenuMessage("Patient was not found!");

}

} else {

utl::setMenuMessage("Incorrect ID!");

}

}

void deleteItemKnowID(int PatID) {

utl::clearScreen();

if (PatID != NULL) {

if (deleteItemByID(PatID)) {

utl::setMenuMessage("Patient successfully deleted!");

}

else {

utl::setMenuMessage("Patient was not found!");

}

}

else {

utl::setMenuMessage("Incorrect ID!");

}

saveData();

}

void findItems() {

string search = utl::readLineFromConsole("Enter search string");

search = utl::strToLower(search);

Pacient \*node = nodeList->root;

string buffer;

while (node != nullptr) {

string repr = toString(node);

string reprLower = utl::strToLower(repr);

string::size\_type matchPos = reprLower.find(search);

if (matchPos != string::npos) {

buffer.append(repr);

}

node = node->next;

}

if (buffer.empty()) {

utl::setMenuMessage("Matches not found!");

} else {

utl::setMenuMessage(buffer.c\_str());

}

}

void printList() {

string buffer = getListRepresentation();

utl::setMenuMessage(buffer.c\_str());

}

void setSort(int sortType) {

sortListIndex = sortType;

sortList();

Pacient \*temp = nodeList->root;

/\* while (temp->next)

{

string message = toString(temp);

utl::setMenuMessage(message.c\_str());

temp = temp->next;

}\*/

while (temp)

{

string message = toString(temp);

utl::setMenuMessage(message.c\_str());

if (!temp->next) {

break;

}

else {

temp = temp->next;

}

}

string message = "Set sorting by " + LIST\_SORT\_TYPES[sortListIndex];

utl::setMenuMessage(message.c\_str());

}

void loadData() {

nodeList = new PatientList;

ifstream storage(STORAGE\_FILEPATH);

if (storage) {

string line;

Pacient \*node;

Pacient \*prev = nullptr;

while (getline(storage, line)) {

if (!line.empty()) {

node = new Pacient;

short cell\_counter = 0;

string::size\_type lastSepPos = 0;

string::size\_type currSepPos = line.find(utl::STORAGE\_SEPARATOR, lastSepPos);

while (currSepPos != string::npos) {

string value = line.substr(lastSepPos, currSepPos - lastSepPos);

switch (cell\_counter) {

case 0:

node->id = stoi(value);

break;

case 1:

node->full\_name = value;

break;

default:

break;

}

cell\_counter++;

lastSepPos = currSepPos + sizeof(utl::STORAGE\_SEPARATOR);

currSepPos = line.find(utl::STORAGE\_SEPARATOR, lastSepPos);

}

if (prev != nullptr) {

prev->next = node;

} else {

nodeList->root = node;

}

prev = node;

nodeList->curr = node;

}

}

}

}

void saveData() {

ofstream storage(STORAGE\_FILEPATH);

Pacient \*node = nodeList->root;

while (node != nullptr) {

string line;

line.append(to\_string(node->id) + utl::STORAGE\_SEPARATOR);

line.append(node->full\_name + utl::STORAGE\_SEPARATOR);

storage << line << endl;

node = node->next;

}

}

void clearMemory() {

Pacient \*node = nodeList->root;

while (node != nullptr) {

Pacient \*buffer = node->next;

delete node;

node = buffer;

}

delete nodeList;

}

bool deleteItemByID(int idP) {

Pacient \*node = nodeList->root;

Pacient \*prev = nullptr;

int schet = 0;

while (node != nullptr) {

if (node->id == idP) {

schet += 1;

if (prev == nullptr) {

nodeList->root = node->next;

delete node;

node = nodeList->root;

- continue;

} else {

prev->next = node->next;

delete node;

node = prev->next;

}

-continue;

break;

}

prev = node;

node = node->next;

}

saveData();

if (schet==0){

return false; }

else {

return true;

}

}

void sortList() {

bool unsorted = true;

Pacient \*curr, \*next, \*prev, \*buffer;

while (unsorted) {

unsorted = false;

curr = nodeList->root;

prev = nullptr;

while (curr) {

next = curr->next;

if (isNeedSwapBySortType(curr, next)) {

- Меняем значения местами.

swap(curr, next);

if (prev != nullptr) {

prev->next = curr;

} else {

nodeList->root = curr;

}

buffer = curr->next;

curr->next = next;

next->next = buffer;

unsorted = true;

}

nodeList->curr = curr;

prev = curr;

curr = next;

}

}

}

bool isNeedSwapBySortType(Pacient \*curr, Pacient \*next) {

if (next == nullptr) {

return false;

}

switch (sortListIndex) {

case LIST\_SORT\_TYPE\_ID:

return curr->id > next->id;

case LIST\_SORT\_TYPE\_NAME:

return curr->full\_name > next->full\_name;

default:

return false;

}

}

int getLastID() {

int last\_id = 0;

Pacient \*node = nodeList->root;

while (node != nullptr) {

if (last\_id < node->id) {

last\_id = node->id;

}

node = node->next;

}

return last\_id;

}

Pacient \*getLastNode() {

Pacient \*prev = nullptr;

Pacient \*curr = nodeList->root;

while (curr != nullptr) {

prev = curr;

curr = curr->next;

}

return prev;

}

bool findItemByID() {

int staffID = utl::readNumberFromConsole("Enter patient ID");

Pacient \*node = getItemByID(staffID);

if (node != nullptr) {

nodeList->curr = node;

return true;

}

return false;

}

Pacient \*getItemByID(int id) {

Pacient \*node = nodeList->root;

while (node != nullptr) {

if (node->id == id) {

return node;

}

node = node->next;

}

return nullptr;

}

string getListRepresentation() {

Pacient \*node = nodeList->root;

string buffer;

buffer.append("ID\tDATA\n");

int counter = 0;

while (node != nullptr) {

string line = toString(node);

buffer.append(line);

counter++;

node = node->next;

}

if (counter == 0) {

buffer.clear();

buffer.append("Nothing to show...");

}

return buffer;

}

string toString(Pacient \*node) {

string buffer;

buffer.append(to\_string(node->id) + '\t');

buffer.append(node->full\_name);

buffer += '\n';

return buffer;

}

}

**reception.h**

#include <string>

#include <ctime>

#include <windows.h>

#include "staff.h"

#include "patient.h"

using namespace std;

#ifndef MEDICAL\_RECEPTION\_H

#define MEDICAL\_RECEPTION\_H

namespace reception {

struct Reception {

int id = NULL;

staff::Doctor \*doctor = nullptr;

patient::Pacient \*patient = nullptr;

tm \*visit\_at = nullptr;

int duration = NULL;

string conclusion;

Reception \*next = nullptr;

};

struct ReceptionList {

Reception \*root = nullptr;

Reception \*curr = nullptr;

};

const char STORAGE\_FILEPATH[] = "./receptions.txt";

const int LIST\_SORT\_TYPE\_ID = 0;

const int LIST\_SORT\_TYPE\_DOCTOR = 1;

const int LIST\_SORT\_TYPE\_PATIENT = 2;

const int LIST\_SORT\_TYPE\_VISIT\_TIME = 3;

const string LIST\_SORT\_TYPES[] = {"ID", "doctor", "patient", "visit time"};

const int MAX\_RECEPTION\_PER\_DAY = 3;

const int PRINT\_TYPE\_TODAY = 0;

const int PRINT\_TYPE\_WAITING = 1;

const int PRINT\_TYPE\_OVERDUE = 2;

const int PRINT\_TYPE\_COMPLETED = 3;

const int PRINT\_TYPE\_ALL = 4;

const string PRINT\_TYPES[] = {"today", "waiting for reception", "overdue", "completed", "all"};

void addItem();

void editItem();

void setConclusion();

void deleteItem();

void findItems();

void findItemsByPatient();

void printList(int printType);

void setSort(int sortType);

void loadData();

void saveData();

void clearMemory();

bool deleteItemByID(int id);

int deleteItemByDoctor(int IDforDel);

int deleteItemByPatient(int IDforDel);

void sortList();

bool isNeedSwapBySortType(Reception \*curr, Reception \*next);

int getLastID();

Reception \*getLastNode();

bool findItemByID();

Reception \*getItemByID(int id);

string getListRepresentation(int printType);

string toString(Reception \*node);

bool getAvailableDateTime(Reception \*node, tm \*day);

bool readAvailableDateFromConsole(Reception &node);

bool PatientHaveNotOtherReceptionOnThisTime(Reception& node);

bool DoctorHaveNotOtherReceptionOnThisTime(Reception& node);

int readNumberFromConsole(const char title[]);

int deleteItemByPatientID(int id);

}

#endif -MEDICAL\_RECEPTION\_H

**reception.cpp**

#include <iostream>

#include <fstream>

#include <iomanip>

#include "./utils.h"

#include "staff.h"

#include "patient.h"

#include "reception.h"

using namespace std;

namespace reception {

ReceptionList\* nodeList;

int sortListIndex = LIST\_SORT\_TYPE\_ID;

void addItem() {

utl::clearScreen();

Reception\* new\_node = new Reception;

cout << "New data" << endl;

cout << utl::HR\_LINE << endl;

int insert\_position = utl::readNumberFromConsole("Enter position to insert");

-bool front = utl::readAnswerFromConsole("Insert front?");

int doctor\_id = utl::readNumberFromConsole("Enter doctor ID");

new\_node->doctor = staff::getItemByID(doctor\_id);

bool validDoctor = new\_node->doctor != nullptr && staff::isNotOnVacation(new\_node->doctor);

int patient\_id = utl::readNumberFromConsole("Enter patient ID");

new\_node->patient = patient::getItemByID(patient\_id);

bool validPatient = new\_node->patient != nullptr;

if (validDoctor && validPatient && readAvailableDateFromConsole(\*new\_node) && PatientHaveNotOtherReceptionOnThisTime(\*new\_node)

&& DoctorHaveNotOtherReceptionOnThisTime(\*new\_node)&& getAvailableDateTime(new\_node,new\_node->visit\_at)) {

new\_node->id = getLastID() + 1;

Reception\* tmp = nodeList->root;

if (tmp == nullptr)

{

nodeList->root = new\_node;

nodeList->curr = new\_node;

saveData();

utl::setMenuMessage("Element successfully created!");

}

else {

int schet = 0;

while (tmp != nullptr) {

if (tmp->id == insert\_position) {

++schet;

new\_node->next = tmp->next;

tmp->next = new\_node;

break;

}

tmp = tmp->next;

}

if (schet == 0) {

delete new\_node;

utl::setMenuMessage("Error adding element!");

}

else {

nodeList->curr = new\_node;

saveData();

utl::setMenuMessage("Element successfully created!");

}

}

}

else {

delete new\_node;

utl::setMenuMessage("Error adding element!");

}

}

void editItem() {

utl::clearScreen();

if (findItemByID()) {

Reception\* node = nodeList->curr;

bool changed = false;

if (node->conclusion.empty()) {

cout << "Edit reception: " << node->id << endl;

cout << utl::HR\_LINE << endl;

if (utl::readAnswerFromConsole("Change [doctor]?")) {

cout << "Current [doctor] value: " << node->doctor->full\_name << endl;

int doctor\_id = utl::readNumberFromConsole("Enter doctor ID");

staff::Doctor\* doctor = staff::getItemByID(doctor\_id);

if (node->doctor != nullptr && staff::isNotOnVacation(node->doctor)) {

node->doctor = doctor;

changed = true;

}

else {

cout << "Incorrect data! [doctor] value does not change. " << endl;

}

}

if (utl::readAnswerFromConsole("Change [patient]?")) {

cout << "Current [patient] value: " << node->patient->full\_name << endl;

int patient\_id = utl::readNumberFromConsole("Enter patient ID");

patient::Pacient\* patient = patient::getItemByID(patient\_id);

if (node->patient != nullptr) {

node->patient = patient;

changed = true;

}

else {

cout << "Incorrect data! [patient] value does not change. " << endl;

}

}

if (utl::readAnswerFromConsole("Change [visit time]?")) {

cout << "Current [visit time] value: " << utl::reprDateTime(node->visit\_at) << endl;

cout << "Current [duration] value: " << node->duration << " minutes" << endl;

if (readAvailableDateFromConsole(\*node)) {

changed = true;

}

else {

cout << "Incorrect data! [visit time] and [duration] values does not change. " << endl;

}

}

if (changed) {

saveData();

utl::setMenuMessage("Element successfully updated!");

}

else {

utl::setMenuMessage("Nothing to change!");

}

}

else {

utl::setMenuMessage("You can not edit element with conclusion!");

}

}

else {

utl::setMenuMessage("Element does not found!");

}

}

void setConclusion() {

utl::clearScreen();

if (findItemByID()) {

Reception\* node = nodeList->curr;

string buffer;

bool changed = false;

cout << "Set conclusion: " << node->id << endl;

cout << utl::HR\_LINE << endl;

if (node->conclusion.empty()) {

changed = utl::readLineFromConsoleTo("conclusion", node->conclusion);

}

else if (utl::readAnswerFromConsole("Change [conclusion]?")) {

cout << "Current [conclusion] value: " << endl << node->conclusion << endl;

if (utl::readLineFromConsoleTo("conclusion", buffer)) {

node->conclusion = buffer;

changed = true;

}

}

if (changed) {

saveData();

utl::setMenuMessage("Conclusion successfully updated!");

}

else {

utl::setMenuMessage("Nothing to change!");

}

}

else {

utl::setMenuMessage("Element does not found!");

}

}

void deleteItem() {

utl::clearScreen();

int ReceptID = utl::readNumberFromConsole("Enter reception ID");

if (ReceptID == NULL) {

utl::setMenuMessage("Incorrect ID!");

}

if (deleteItemByID(ReceptID)) {

utl::setMenuMessage("Reception successfully deleted!");

}

else {

utl::setMenuMessage("Reception was not found!");

}

}

void findItems() {

string search = utl::readLineFromConsole("Enter search string");

search = utl::strToLower(search);

Reception\* node = nodeList->root;

string buffer;

while (node != nullptr) {

string repr = toString(node);

string reprLower = utl::strToLower(repr);

string::size\_type matchPos = reprLower.find(search);

if (matchPos != string::npos) {

buffer.append(repr);

}

node = node->next;

}

if (buffer.empty()) {

utl::setMenuMessage("Matches not found!");

}

else {

utl::setMenuMessage(buffer.c\_str());

}

}

void findItemsByPatient() {

string buffer;

string search = utl::readLineFromConsole("Enter search string");

search = utl::strToLower(search);

auto\* matchesList = new ReceptionList();

Reception\* node, \* match;

node = nodeList->root;

while (node != nullptr) {

string patientName = utl::strToLower(node->patient->full\_name);

string::size\_type matchPos = patientName.find(search);

if (matchPos != string::npos) {

match = new Reception();

\*match = \*node;

match->next = nullptr;

if (matchesList->root == nullptr) {

matchesList->root = match;

}

else {

matchesList->curr->next = match;

}

matchesList->curr = match;

}

node = node->next;

}

node = matchesList->root;

while (node != nullptr) {

buffer.append(toString(node));

match = node->next;

delete node;

node = match;

}

delete matchesList;

if (buffer.empty()) {

utl::setMenuMessage("Matches not found!");

}

else {

utl::setMenuMessage(buffer.c\_str());

}

}

void printList(int printType) {

string buffer = getListRepresentation(printType);

utl::setMenuMessage(buffer.c\_str());

}

void setSort(int sortType) {

sortListIndex = sortType;

sortList();

Reception\* temp = nodeList->root;

/\*while (temp->next)

{

string message = toString(temp);

utl::setMenuMessage(message.c\_str());

temp = temp->next;

}\*/ while (temp)

{

string message = toString(temp);

utl::setMenuMessage(message.c\_str());

if (!temp->next) {

break;

}

else {

temp = temp->next;

}

}

string message = "Set sorting by " + LIST\_SORT\_TYPES[sortListIndex];

utl::setMenuMessage(message.c\_str());

}

void loadData() {

nodeList = new ReceptionList;

ifstream storage(STORAGE\_FILEPATH);

if (storage) {

string line;

Reception\* node;

Reception\* prev = nullptr;

while (getline(storage, line)) {

if (!line.empty()) {

node = new Reception;

short cell\_counter = 0;

string::size\_type lastSepPos = 0;

string::size\_type currSepPos = line.find(utl::STORAGE\_SEPARATOR, lastSepPos);

while (currSepPos != string::npos) {

string value = line.substr(lastSepPos, currSepPos - lastSepPos);

switch (cell\_counter) {

case 0:

node->id = stoi(value);

break;

case 1:

node->doctor = staff::getItemByID(stoi(value));

break;

case 2:

node->patient = patient::getItemByID(stoi(value));

break;

case 3:

node->duration = stoi(value);

break;

case 4:

node->visit\_at = new tm;

utl::parseStorageStringDateTime(value, \*node->visit\_at);

-mktime(node->visit\_at);

break;

case 5:

node->conclusion = value;

break;

default:

break;

}

cell\_counter++;

lastSepPos = currSepPos + sizeof(utl::STORAGE\_SEPARATOR);

currSepPos = line.find(utl::STORAGE\_SEPARATOR, lastSepPos);

}

if (node->doctor == nullptr || node->patient == nullptr) {

continue;

}

if (prev != nullptr) {

prev->next = node;

}

else {

nodeList->root = node;

}

prev = node;

nodeList->curr = node;

}

}

}

}

void saveData() {

ofstream storage(STORAGE\_FILEPATH);

/\*if (storage)

{

int a = 1;

}\*/

Reception\* node = nodeList->root;

while (node != nullptr) {

string line;

line.append(to\_string(node->id) + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->doctor->id) + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->patient->id) + utl::STORAGE\_SEPARATOR);

line.append(to\_string(node->duration) + utl::STORAGE\_SEPARATOR);

line.append(utl::reprDateTime(node->visit\_at) + utl::STORAGE\_SEPARATOR);

line.append(node->conclusion + utl::STORAGE\_SEPARATOR);

storage << line << endl;

node = node->next;

}

storage.close();

}

void clearMemory() {

Reception\* node = nodeList->root;

while (node != nullptr) {

Reception\* buffer = node->next;

delete node->visit\_at;

delete node;

node = buffer;

}

delete nodeList;

}

bool deleteItemByID(int id) {

Reception\* node = nodeList->root;

Reception\* prev = nullptr;

int schet =0;

while (node != nullptr) {

if (node->id == id) {

if (prev == nullptr) {

schet += 1;

nodeList->root = node->next;

delete node;

node = nodeList->root;

}

else {

prev->next = node->next;

delete node;

node = prev->next;

}

continue;

}

prev = node;

node = node->next;

}

saveData();

if (schet == 0) {

return false;

}

else {

return true;

}

}

int deleteItemByDoctor(int IDforDel) {

Reception\* node = nodeList->root;

Reception\* prev = nullptr;

while (node != nullptr) {

if (node->doctor->id == IDforDel) {

if (prev == nullptr) {

nodeList->root = node->next;

delete node;

node = nodeList->root;

}

else {

prev->next = node->next;

delete node;

node = prev->next;

}

utl::setMenuMessage("Reception successfully deleted!");

continue;

}

prev = node;

node = node->next;

}

saveData();

return IDforDel;

}

int readNumberFromConsole(const char title[]) {

int tries = 0;

int buffer;

string input;

while (tries < utl::MAX\_INPUT\_TRIES) {

cout << title << ": ";

cin >> input;

try {

buffer = stoi(input);

return buffer;

}

catch (invalid\_argument& err) {

tries++;

cout << "Not a number. Try again!" << endl;

}

catch (out\_of\_range& err) {

tries++;

cout << "Value is overflow. Try again!" << endl;

}

}

return NULL;

}

int deleteItemByPatientID(int IDforDel) {

Reception\* node = nodeList->root;

Reception\* prev = nullptr;

while (node != nullptr) {

if (node->patient->id == IDforDel) {

if (prev == nullptr) {

nodeList->root = node->next;

delete node;

node = nodeList->root;

}

else {

prev->next = node->next;

delete node;

node = prev->next;

}

utl::setMenuMessage("Reception successfully deleted!");

continue;

}

prev = node;

node = node->next;

}

saveData();

return IDforDel;

}

void sortList() {

bool unsorted = true;

Reception\* curr, \* next, \* prev, \* buffer;

while (unsorted) {

unsorted = false;

curr = nodeList->root;

prev = nullptr;

while (curr) {

next = curr->next;

if (isNeedSwapBySortType(curr, next)) {

- Меняем значения местами.

swap(curr, next);

if (prev != nullptr) {

prev->next = curr;

}

else {

nodeList->root = curr;

}

buffer = curr->next;

curr->next = next;

next->next = buffer;

unsorted = true;

}

nodeList->curr = curr;

prev = curr;

curr = next;

}

}

}

bool isNeedSwapBySortType(Reception\* curr, Reception\* next) {

if (next == nullptr) {

return false;

}

switch (sortListIndex) {

case LIST\_SORT\_TYPE\_ID:

return curr->id > next->id;

case LIST\_SORT\_TYPE\_DOCTOR:

return curr->doctor->full\_name > next->doctor->full\_name;

case LIST\_SORT\_TYPE\_PATIENT:

return curr->patient->full\_name > next->patient->full\_name;

case LIST\_SORT\_TYPE\_VISIT\_TIME:

return utl::isDTGreaterThan(curr->visit\_at, next->visit\_at);

default:

return false;

}

}

int getLastID() {

int last\_id = 0;

Reception\* node = nodeList->root;

while (node != nullptr) {

if (last\_id < node->id) {

last\_id = node->id;

}

node = node->next;

}

return last\_id;

}

Reception\* getLastNode() {

Reception\* prev = nullptr;

Reception\* curr = nodeList->root;

while (curr != nullptr) {

prev = curr;

curr = curr->next;

}

return prev;

}

bool findItemByID() {

int staffID = utl::readNumberFromConsole("Enter reception ID");

Reception\* node = getItemByID(staffID);

if (node != nullptr) {

nodeList->curr = node;

return true;

}

return false;

}

Reception\* getItemByID(int id) {

Reception\* node = nodeList->root;

while (node != nullptr) {

if (node->id == id) {

return node;

}

node = node->next;

}

return nullptr;

}

string getListRepresentation(int printType) {

Reception\* node = nodeList->root;

string buffer;

time\_t today\_time{};

time(&today\_time);

tm\* today = localtime(&today\_time);

buffer.append("ID\tVISIT\tDOCTOR / PATIENT\n");

int counter = 0;

bool canPrint;

while (node != nullptr) {

switch (printType) {

case PRINT\_TYPE\_TODAY:

canPrint = node->conclusion.empty() && utl::isSameDay(today, node->visit\_at);

break;

case PRINT\_TYPE\_WAITING:

canPrint = node->conclusion.empty() && utl::isDTGreaterThan(node->visit\_at, today);

break;

case PRINT\_TYPE\_OVERDUE:

canPrint = node->conclusion.empty() && utl::isDTGreaterThan(today, node->visit\_at);

break;

case PRINT\_TYPE\_COMPLETED:

canPrint = !node->conclusion.empty();

break;

case PRINT\_TYPE\_ALL:

canPrint = true;

break;

default:

canPrint = false;

break;

}

if (canPrint) {

string line = toString(node);

buffer.append(line);

counter++;

}

node = node->next;

}

if (counter == 0) {

buffer.clear();

buffer.append("Nothing to show...");

}

return buffer;

}

string toString(Reception\* node) {

string buffer;

buffer.append(to\_string(node->id) + '\t');

buffer.append('[' + utl::getWeekDayName(node->visit\_at->tm\_wday, true) + "] ");

buffer.append(utl::reprDateTime(node->visit\_at) + '\t');

buffer.append(node->doctor->full\_name + '(' + node->doctor->specialty + ") / " + node->patient->full\_name);

if (!node->conclusion.empty()) {

buffer.append("\n\tConclusion: " + node->conclusion + '\n');

}

buffer += '\n';

return buffer;

}

-bool getAvailableDateTime(Reception\* node, tm\* day, staff::WorkDay\* wd) {

bool getAvailableDateTime(Reception\* node, tm\* day) {

staff::Doctor\* doctor = node->doctor;

patient::Pacient\* patient = node->patient;

tm\* available = new tm;

\*available = \*day;

/\*available->tm\_hour = wd->start\_at->tm\_hour;

available->tm\_min = wd->start\_at->tm\_min;\*/

int receptCnt = 0;

Reception\* curr = nodeList->root;

while (curr != nullptr && receptCnt < MAX\_RECEPTION\_PER\_DAY) {

tm\* visit\_at = curr->visit\_at;

if (curr->conclusion.empty() && utl::isSameDay(day, visit\_at)) {

bool isSamePatient = curr->patient->id == patient->id;

/\*if (curr->doctor->id == doctor->id) {

if (isSamePatient) {

delete available;

return false;

}\*/

/\*bool validHour = visit\_at->tm\_hour >= available->tm\_hour;

bool validMinutes = visit\_at->tm\_min >= available->tm\_min;

if (validHour && validMinutes) {

available->tm\_hour = visit\_at->tm\_hour;

available->tm\_min = visit\_at->tm\_min;

available->tm\_min += curr->duration;

}\*/

-}

if (isSamePatient && visit\_at->tm\_wday == available->tm\_wday) {

receptCnt++;

}

}

curr = curr->next;

}

mktime(available);

-if (receptCnt < MAX\_RECEPTION\_PER\_DAY && utl::isCorrectTimePair(wd->end\_at, available, doctor->duration)) {

if (receptCnt < MAX\_RECEPTION\_PER\_DAY) {

delete available;

return true;

}

else {

delete available;

}

return false;

}

bool readAvailableDateFromConsole(Reception& node) {

time\_t buffer{};

time(&buffer);

int counter = 0;

tm\* day = localtime(&buffer);

struct visit

{

visit\* next = nullptr;

tm time;

};

visit\* head = nullptr;

cout << "Available dates: " << endl;

auto temp = node.doctor->workdays;

while (temp) {

mktime(day);

staff::WorkDay\* wd = temp;

if (wd->is\_day\_off) {

continue;

}

tm\* available = wd->start\_at;

if (available != nullptr) {

auto new\_visit = new visit;

new\_visit->time = \*available;

if (head == nullptr)

{

head = new\_visit;

}

else {

auto r = head;

while (r->next)

{

r = r->next;

}

r->next = new\_visit;

}

-std::cout << '[' << counter << "] " << std::put\_time(available, "%d.%m.%Y (%A) %H:%M") << endl;

-std::cout << '[' << counter << "] " << std::put\_time(available, "%m.%Y (%A) %H:%M") << endl;

std::cout << '[' << counter << "] " << std::put\_time(available, "%H:%M-(%A)") << endl;

counter++;

}

temp = temp->next;

}

int choice;

if (counter > 0) {

if (utl::readNumberFromConsoleTo("visit time", choice, 0, counter - 1)) {

tm\* dt = new tm;

auto temp2 = head;

int temp2\_counter = 0;

while (temp2)

{

if (temp2\_counter == choice)

{

\*dt = temp2->time;

break;

}

temp2 = temp2->next;

temp2\_counter++;

}

node.visit\_at = dt;

node.duration = node.doctor->duration;

return true;

}

}

else {

cout << "No dates available..." << endl;

}

return false;

}

-bool PatientHaveNotOtherReceptionOnThisTime(Reception& node) {

- /\* поиск записей пациента по ид пациента\*/

- -nodeList

- Reception\* tmp\_rec = nodeList->root;

- while (tmp\_rec) {

- if (node.patient->id == tmp\_rec->patient->id) {

- auto tmp\_time1 = node.visit\_at;

- auto tmp\_time2 = tmp\_rec->visit\_at;

- if ((tmp\_time1->tm\_min == tmp\_time2->tm\_min) && (tmp\_time1->tm\_hour == tmp\_time2->tm\_hour)

- && (tmp\_time1->tm\_mday == tmp\_time2->tm\_mday) && (tmp\_time1->tm\_mon == tmp\_time2->tm\_mon)

- && (tmp\_time1->tm\_year == tmp\_time2->tm\_year)) {

- return false;

- }

- }

- tmp\_rec = tmp\_rec->next;

- }

- return true;

-}

bool PatientHaveNotOtherReceptionOnThisTime(Reception& node) {

Reception\* tmp\_rec = nodeList->root;

while (tmp\_rec) {

if (node.patient->id == tmp\_rec->patient->id) {

auto tmp\_time1 = node.visit\_at;

auto tmp\_time2 = tmp\_rec->visit\_at;

if ((tmp\_time1->tm\_min == tmp\_time2->tm\_min) && (tmp\_time1->tm\_hour == tmp\_time2->tm\_hour)

&& (tmp\_time1->tm\_wday == tmp\_time2->tm\_wday)) {

return false;

}

}

tmp\_rec = tmp\_rec->next;

}

return true;

}

bool DoctorHaveNotOtherReceptionOnThisTime(Reception& node) {

Reception\* tmp\_rec = nodeList->root;

while (tmp\_rec) {

if (node.doctor->id == tmp\_rec->doctor->id) {

auto tmp\_time1 = node.visit\_at;

auto tmp\_time2 = tmp\_rec->visit\_at;

if ((tmp\_time1->tm\_min == tmp\_time2->tm\_min) && (tmp\_time1->tm\_hour == tmp\_time2->tm\_hour)

&& (tmp\_time1->tm\_wday == tmp\_time2->tm\_wday)) {

return false;

}

}

tmp\_rec = tmp\_rec->next;

}

return true;

}

}

Medical.h

#include <string>

#ifndef MEDICAL\_MEDICAL\_H

#define MEDICAL\_MEDICAL\_H

using namespace std;

namespace mdc {

const char APP\_TITLE[] = "Medical CRM system";

const int MENU\_SIZE\_LIMIT = 10;

- Системные коды.

const int MC\_EXIT = 1;

const int MC\_MAIN = 2;

const int MC\_UNDECLARED = 3;

- Коды основного меню.

const int MC\_STAFF\_DATA = 100;

const int MC\_PATIENT\_DATA = 200;

const int MC\_RECEPTION\_DATA = 300;

- Коды меню: Данные персонала.

const int MC\_STAFF\_DATA\_ADD = 1000;

const int MC\_STAFF\_DATA\_DELETE = 1001;

const int MC\_STAFF\_DATA\_EDIT = 1002;

const int MC\_STAFF\_DATA\_FIND = 1003;

const int MC\_STAFF\_DATA\_PRINT = 1004;

const int MC\_STAFF\_DATA\_SET\_SORT = 1005;

const int MC\_STAFF\_DATA\_SET\_SORT\_BY\_ID = 100501;

const int MC\_STAFF\_DATA\_SET\_SORT\_BY\_NAME = 100502;

const int MC\_STAFF\_DATA\_SET\_SORT\_BY\_SPECIALTY = 100503;

const int MC\_STAFF\_DATA\_SET\_SORT\_BY\_CATEGORY = 100504;

- Коды меню: Данные пациентов.

const int MC\_PATIENT\_DATA\_ADD = 2000;

const int MC\_PATIENT\_DATA\_DELETE = 2001;

const int MC\_PATIENT\_DATA\_EDIT = 2002;

const int MC\_PATIENT\_DATA\_FIND = 2003;

const int MC\_PATIENT\_DATA\_PRINT = 2004;

const int MC\_PATIENT\_DATA\_SET\_SORT = 2005;

const int MC\_PATIENT\_DATA\_SET\_SORT\_BY\_ID = 200501;

const int MC\_PATIENT\_DATA\_SET\_SORT\_BY\_NAME = 200502;

- Коды меню: Данные персонала.

const int MC\_RECEPTION\_DATA\_ADD = 3000;

const int MC\_RECEPTION\_DATA\_DELETE = 3001;

const int MC\_RECEPTION\_DATA\_EDIT = 3002;

const int MC\_RECEPTION\_DATA\_FIND = 3003;

const int MC\_RECEPTION\_DATA\_FIND\_BY\_PATIENT = 3007;

const int MC\_RECEPTION\_DATA\_PRINT = 3004;

const int MC\_RECEPTION\_DATA\_SET\_SORT = 3005;

const int MC\_RECEPTION\_DATA\_SET\_CONCLUSION = 3006;

const int MC\_RECEPTION\_DATA\_PRINT\_TODAY = 300401;

const int MC\_RECEPTION\_DATA\_PRINT\_WAITING = 300402;

const int MC\_RECEPTION\_DATA\_PRINT\_OVERDUE = 300403;

const int MC\_RECEPTION\_DATA\_PRINT\_COMPLETED = 300404;

const int MC\_RECEPTION\_DATA\_PRINT\_ALL = 300405;

const int MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_ID = 300501;

const int MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_DOCTOR = 300502;

const int MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_PATIENT = 300503;

const int MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_VISIT\_TIME = 300504;

void loopMenu();

void printPage();

void displayMenu();

void pushMenuItem(const char \*name, int code, string \*menu);

void readUserMenuChoice();

void doUserChoiceAction();

void loadData();

void clearMemory();

}

#endif -MEDICAL\_MEDICAL\_H

Medical.cpp

#include <iostream>

#include "utils.h"

#include "staff.h"

#include "patient.h"

#include "reception.h"

#include "medical.h"

using namespace std;

namespace mdc {

int menuCodes[MENU\_SIZE\_LIMIT];

int curMenuCode = MC\_MAIN;

void loadData() {

staff::loadData();

patient::loadData();

reception::loadData();

}

void clearMemory() {

staff::clearMemory();

patient::clearMemory();

reception::clearMemory();

}

void loopMenu() {

while (curMenuCode != MC\_EXIT) {

printPage();

readUserMenuChoice();

doUserChoiceAction();

}

}

void printPage() {

utl::clearScreen();

cout << APP\_TITLE << endl;

cout << utl::HR\_LINE << endl;

if (utl::isMenuMessage()) {

utl::printMenuMessage();

}

displayMenu();

}

void displayMenu() {

string menu[MENU\_SIZE\_LIMIT];

for (int &menuCode : menuCodes) {

menuCode = MC\_UNDECLARED;

}

switch (curMenuCode) {

case MC\_MAIN:

cout << "Main menu:" << endl;

pushMenuItem("Staff data", MC\_STAFF\_DATA, menu);

pushMenuItem("Patient data", MC\_PATIENT\_DATA, menu);

pushMenuItem("Receptions data", MC\_RECEPTION\_DATA, menu);

break;

case MC\_STAFF\_DATA:

cout << "Staff data:" << endl;

pushMenuItem("Add staff", MC\_STAFF\_DATA\_ADD, menu);

pushMenuItem("Edit staff", MC\_STAFF\_DATA\_EDIT, menu);

pushMenuItem("Delete staff", MC\_STAFF\_DATA\_DELETE, menu);

pushMenuItem("Find staff", MC\_STAFF\_DATA\_FIND, menu);

pushMenuItem("Print staff", MC\_STAFF\_DATA\_PRINT, menu);

pushMenuItem("Set sort", MC\_STAFF\_DATA\_SET\_SORT, menu);

break;

case MC\_STAFF\_DATA\_SET\_SORT:

cout << "Sort staff by:" << endl;

pushMenuItem("By ID",MC\_STAFF\_DATA\_SET\_SORT\_BY\_ID , menu);

pushMenuItem("By name", MC\_STAFF\_DATA\_SET\_SORT\_BY\_NAME, menu);

pushMenuItem("By specialty", MC\_STAFF\_DATA\_SET\_SORT\_BY\_SPECIALTY, menu);

pushMenuItem("By category", MC\_STAFF\_DATA\_SET\_SORT\_BY\_CATEGORY, menu);

break;

case MC\_PATIENT\_DATA:

cout << "Patients data:" << endl;

pushMenuItem("Add patient", MC\_PATIENT\_DATA\_ADD, menu);

pushMenuItem("Edit patient", MC\_PATIENT\_DATA\_EDIT, menu);

pushMenuItem("Delete patient", MC\_PATIENT\_DATA\_DELETE, menu);

pushMenuItem("Find patient", MC\_PATIENT\_DATA\_FIND, menu);

pushMenuItem("Print patients", MC\_PATIENT\_DATA\_PRINT, menu);

pushMenuItem("Set sort", MC\_PATIENT\_DATA\_SET\_SORT, menu);

break;

case MC\_PATIENT\_DATA\_SET\_SORT:

cout << "Sort patients by:" << endl;

pushMenuItem("By ID", MC\_PATIENT\_DATA\_SET\_SORT\_BY\_ID, menu);

pushMenuItem("By name", MC\_PATIENT\_DATA\_SET\_SORT\_BY\_NAME, menu);

break;

case MC\_RECEPTION\_DATA:

cout << "Receptions data:" << endl;

pushMenuItem("Add reception", MC\_RECEPTION\_DATA\_ADD, menu);

pushMenuItem("Edit reception", MC\_RECEPTION\_DATA\_EDIT, menu);

pushMenuItem("Delete reception", MC\_RECEPTION\_DATA\_DELETE, menu);

pushMenuItem("Set conclusion", MC\_RECEPTION\_DATA\_SET\_CONCLUSION, menu);

pushMenuItem("Find reception", MC\_RECEPTION\_DATA\_FIND, menu);

pushMenuItem("Find reception by patient", MC\_RECEPTION\_DATA\_FIND\_BY\_PATIENT, menu);

pushMenuItem("Print receptions", MC\_RECEPTION\_DATA\_PRINT, menu);

pushMenuItem("Set sort", MC\_RECEPTION\_DATA\_SET\_SORT, menu);

break;

case MC\_RECEPTION\_DATA\_PRINT:

cout << "Print receptions:" << endl;

pushMenuItem("Today", MC\_RECEPTION\_DATA\_PRINT\_TODAY, menu);

pushMenuItem("Waiting for reception", MC\_RECEPTION\_DATA\_PRINT\_WAITING, menu);

pushMenuItem("Overdue", MC\_RECEPTION\_DATA\_PRINT\_OVERDUE, menu);

pushMenuItem("Completed", MC\_RECEPTION\_DATA\_PRINT\_COMPLETED, menu);

pushMenuItem("All", MC\_RECEPTION\_DATA\_PRINT\_ALL, menu);

break;

case MC\_RECEPTION\_DATA\_SET\_SORT:

cout << "Sort receptions by:" << endl;

pushMenuItem("By ID", MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_ID, menu);

pushMenuItem("By doctor", MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_DOCTOR, menu);

pushMenuItem("By patient", MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_PATIENT, menu);

pushMenuItem("By visit time", MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_VISIT\_TIME, menu);

break;

default:

cout << "Undefined operation!" << endl;

break;

}

- Определяем куда будет вести кнопка назад

switch (curMenuCode) {

case MC\_MAIN:

pushMenuItem("Exit", MC\_EXIT, menu);

break;

case MC\_STAFF\_DATA\_SET\_SORT:

pushMenuItem("Staff data", MC\_STAFF\_DATA, menu);

break;

case MC\_PATIENT\_DATA\_SET\_SORT:

pushMenuItem("Patients data", MC\_PATIENT\_DATA, menu);

break;

case MC\_RECEPTION\_DATA\_PRINT:

case MC\_RECEPTION\_DATA\_SET\_SORT:

pushMenuItem("Receptions data", MC\_RECEPTION\_DATA, menu);

break;

default:

pushMenuItem("Main menu", MC\_MAIN, menu);

break;

}

for (int i = 0; i < MENU\_SIZE\_LIMIT; i++) {

string &menuItem = menu[i];

if (menuItem.empty()) {

break;

}

cout << "[" << i + 1 << "] " << menuItem << endl;

}

}

void pushMenuItem(const char \*name, int code, string \*menu) {

for (int i = 0; i < MENU\_SIZE\_LIMIT; i++) {

if (menu[i].empty()) {

menu[i] = name;

menuCodes[i] = code;

break;

}

}

}

void readUserMenuChoice() {

int choice = utl::readNumberFromConsole("Select action");

if (choice != NULL) {

if (choice <= 0 || choice > MENU\_SIZE\_LIMIT || menuCodes[choice - 1] == MC\_UNDECLARED) {

utl::setMenuMessage("Invalid choice!");

} else {

curMenuCode = menuCodes[choice - 1];

}

} else {

utl::setMenuMessage("Input error for 4 attempts. Operation is interrupted!");

}

}

void doUserChoiceAction() {

switch (curMenuCode) {

- Работа с данными по сотрудникам.

case MC\_STAFF\_DATA\_ADD:

staff::addItem();

curMenuCode = MC\_STAFF\_DATA;

break;

case MC\_STAFF\_DATA\_EDIT:

staff::editItem();

curMenuCode = MC\_STAFF\_DATA;

break;

case MC\_STAFF\_DATA\_DELETE:

staff::deleteItemWeKnowID(reception::deleteItemByDoctor(reception::readNumberFromConsole("Enter staff ID")));

-поменять местами, сравнивать с ид, а не с 0

curMenuCode = MC\_STAFF\_DATA;

break;

case MC\_STAFF\_DATA\_FIND:

staff::findItems();

curMenuCode = MC\_STAFF\_DATA;

break;

case MC\_STAFF\_DATA\_PRINT:

staff::printList();

curMenuCode = MC\_STAFF\_DATA;

break;

case MC\_STAFF\_DATA\_SET\_SORT\_BY\_ID:

staff::setSort(staff::LIST\_SORT\_TYPE\_ID);

curMenuCode = MC\_STAFF\_DATA;

break;

case MC\_STAFF\_DATA\_SET\_SORT\_BY\_NAME:

staff::setSort(staff::LIST\_SORT\_TYPE\_NAME);

curMenuCode = MC\_STAFF\_DATA;

break;

case MC\_STAFF\_DATA\_SET\_SORT\_BY\_SPECIALTY:

staff::setSort(staff::LIST\_SORT\_TYPE\_SPECIALTY);

curMenuCode = MC\_STAFF\_DATA;

break;

case MC\_STAFF\_DATA\_SET\_SORT\_BY\_CATEGORY:

staff::setSort(staff::LIST\_SORT\_TYPE\_CATEGORY);

curMenuCode = MC\_STAFF\_DATA;

break;

- Работа с данными по пациентам.

case MC\_PATIENT\_DATA\_ADD:

patient::addItem();

curMenuCode = MC\_PATIENT\_DATA;

break;

case MC\_PATIENT\_DATA\_EDIT:

patient::editItem();

curMenuCode = MC\_PATIENT\_DATA;

break;

case MC\_PATIENT\_DATA\_DELETE:

-patient::deleteItem();

--поменять местами, передать идишник

-reception::deleteItemByPatient();

patient::deleteItemKnowID(reception::deleteItemByPatientID(reception::readNumberFromConsole("Enter patient ID")));

curMenuCode = MC\_PATIENT\_DATA;

break;

case MC\_PATIENT\_DATA\_FIND:

patient::findItems();

curMenuCode = MC\_PATIENT\_DATA;

break;

case MC\_PATIENT\_DATA\_PRINT:

patient::printList();

curMenuCode = MC\_PATIENT\_DATA;

break;

case MC\_PATIENT\_DATA\_SET\_SORT\_BY\_ID:

patient::setSort(patient::LIST\_SORT\_TYPE\_ID);

curMenuCode = MC\_PATIENT\_DATA;

break;

case MC\_PATIENT\_DATA\_SET\_SORT\_BY\_NAME:

patient::setSort(patient::LIST\_SORT\_TYPE\_NAME);

curMenuCode = MC\_PATIENT\_DATA;

break;

- Работа с данными по записям на приём.

case MC\_RECEPTION\_DATA\_ADD:

reception::addItem();

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_EDIT:

reception::editItem();

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_SET\_CONCLUSION:

reception::setConclusion();

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_DELETE:

reception::deleteItem();

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_FIND:

reception::findItems();

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_FIND\_BY\_PATIENT:

reception::findItemsByPatient();

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_PRINT\_TODAY:

reception::printList(reception::PRINT\_TYPE\_TODAY);

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_PRINT\_WAITING:

reception::printList(reception::PRINT\_TYPE\_WAITING);

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_PRINT\_OVERDUE:

reception::printList(reception::PRINT\_TYPE\_OVERDUE);

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_PRINT\_COMPLETED:

reception::printList(reception::PRINT\_TYPE\_COMPLETED);

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_PRINT\_ALL:

reception::printList(reception::PRINT\_TYPE\_ALL);

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_ID:

reception::setSort(reception::LIST\_SORT\_TYPE\_ID); curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_DOCTOR:

reception::setSort(reception::LIST\_SORT\_TYPE\_DOCTOR);

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_PATIENT:

reception::setSort(reception::LIST\_SORT\_TYPE\_PATIENT);

curMenuCode = MC\_RECEPTION\_DATA;

break;

case MC\_RECEPTION\_DATA\_SET\_SORT\_BY\_VISIT\_TIME:

reception::setSort(reception::LIST\_SORT\_TYPE\_VISIT\_TIME);

curMenuCode = MC\_RECEPTION\_DATA;

break;

default:

break;

}

}

}