Изображение выглядит как цепь, медальон

Автоматически созданное описание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Автоматизированные системы обработки информации и управления | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Программирование | | |
| Курс | I | | | Группа | 423 |

Отчёт по контрольной работе №1

Вариант № 12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 423 |  |  |  | Лянко Максим Владимирович |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверил(и): |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |
|  |  |  |  |
|  |  |  |  |

Содержание

[1. Постановка задачи 3](#_Toc105730859)

[2. Исходные данные 3](#_Toc105730860)

[3. Особые ситуации 3](#_Toc105730861)

[4. Математические методы и алгоритмы решения задач 3](#_Toc105730862)

[5. Форматы представления данных 4](#_Toc105730863)

[6. Структура программы 5](#_Toc105730864)

[7. Блок-схемы алгоритмов программы 9](#_Toc105730865)

[8. Описание хода выполнения контрольной работы 11](#_Toc105730866)

[9. Результат работы программы 12](#_Toc105730867)

[10. Исходный текст программы 15](#_Toc105730868)

# Постановка задачи

Необходимо разработать класс для указанной предметной области. Доступ к данным реализовать с помощью методов Set, Get, Show.

Квартира: адрес, этаж, количество комнат, площадь. Создать массив объектов. Реализовать возможность получения:

– списка квартир, имеющих заданное число комнат,

– списка квартир, имеющих заданное число комнат и расположенных на этаже, который находится в определенном промежутке,

– списка квартир, имеющих площадь, превосходящую заданную

# Исходные данные

В качестве исходных данных программа использует введенные пользователем или считаные из файла количество и данные о квартирах.

# Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

1. Ввод пользователем некорректного имени файла
2. Загрузка некорректных данных с файла
3. Некорректный ввод значений переменных
4. Непредвиденное поведение программы при нехватке памяти

# Математические методы и алгоритмы решения задач

Для вывода списка квартир, имеющих конкретное количество комнат, программа поочередно сравнивает количество комнат каждой квартиры с количеством комнат, введенным пользователем. Если числа совпадают, то выводится информация о квартире.

Чтобы сформировать список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в определенном промежутке, программа поочередно сравнивает количество комнат каждой квартиры с количеством комнат, введенным пользователем и этаж квартиры с верхней и нижней границами интервала, и выводит данные о квартире, если квартира с количеством комнат введенным пользователем расположена на этаже который попадает в заданный пользователем интервал.

Для вывода списка квартир, имеющих площадь, превосходящую заданную, программа поочередно сравнивает площадь каждой квартиры с площадью, введенным пользователем. Если площадь квартиры, больше площади введенной пользователем, то выводится информация о квартире.

# Форматы представления данных

Программа использует следующие переменные:

Таблица 1 — Список переменных, используемых в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| userMenuChoice | MENU\_ITEM | Пользовательский выбор в меню |
| userСhoice | CHOICE | Пользовательский выбор |
| userSecondChoice | SECOND\_CHOICE | Пользовательский выбор |
| apartments | Apartment\* | Массив, содержащий список пациентов |
| apartmentsResult | Apartment\* | Массив, содержащий список пациентов, удовлетворяющих условию поиска |
| address | string | Адрес квартиры |
| intervalLow | int | Нижняя граница интервала |
| intervalUp | int | Верхняя граница интервала |
| size | int | Размер массива apartments |
| count\_patients | int | Размер массива apartmentsResult |
| isDataSaved | bool | Определяет, сохранены ли данные |
| is\_file\_correct | bool | Определяет корректность файла для ввода данных |
| is\_data\_loaded | bool | Определяет, загружены ли данные |
| test | UnitTest | Модульные тесты |
| filePath | string | Путь к файлу |
| current\_path | string | Путь к директории с программой |
| file | ofstream | Сохранение в файл |
| file | ifstream | Чтение из файла |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| error | bad\_alloc& | Описывает исключение о том, что запрос на выделение памяти не может быть выполнен |

Для хранения данных о квартирах используются объекты класса Apartment, в котором задаются адрес, этаж, количество комнат, площадь квартиры.

Модульные тесты описаны как методы класса UnitTest, представлено 3 различных наборов данных, которые проверяют три основных алгоритма программы — поиска квартир по заданному количеству комнат, поиска квартир по количеству комнат и заданному интервалу этажей.

# Структура программы

Программа разделена на семь исходных и семь заголовочных файлов.

Таблица 2 — Список заголовочных файлов

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| algorithms.h | Содержит объявления функций для работы с массивом пациентов |
| fileWork.h | Содержит объявления функций для работы с файлами |
| inputValidation.h | Содержит объявления функций для пользовательского ввода |
| interface.h | Содержит объявления функций пользовательского интерфейса |
| enums.h | Содержит объявления классов перечисления |
| apartment.h | Содержит объявление класса Apartment и его методов |
| tests.h | Содержит объявление класса Tests и его методов |

Таблица 3 — Список исходных файлов

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| algorithms.cpp | Содержит определения функций для работы с массивом пациентов |
| fileWork.cpp | Содержит определения функций для работы с файлами |
| inputValidation.cpp | Содержит определения функций для пользовательского ввода |
| interface.cpp | Содержит определения функций для пользовательского интерфейса |
| main.cpp | Содержит функцию main, точка входа в программу |
| apartment.cpp | Содержит определения методов класса Apartment |
| tests.cpp | Содержит определения методов класса Tests |

Таблица 4 – Функции, составляющие программу

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| allocateArray | Выделить память под массив |
| clearData | Очистка массива данных из памяти |
| showApartments | Вывести список квартир |
| fillApartments | Заполнить список квартир пользовательским вводом |
| getCountApartmentsForNumbersOfRooms | Получить количество подходящих квартир для поиска по количеству комнат |
| getCountApartmentsForNumbersOfRoomsAndInterval | Получить количество подходящих квартир для поиска в интервале и с подходящим количеством комнат |
| searchByNumbersOfRooms | Искать квартиры по количеству комнат |
| searchInInterval | Искать квартиры в заданном интервале и по количеству комнат |
| searchBySquare | Искать квартиры с площадью превосходящую заданную |
| saveInFile | Сохранить данные в файл |
| uploadFromFile | Загрузить данные из файла |

Продолжение таблицы 4

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| getSizeFromFile | Получить размер массива из файла |
| isFileCorrect | Проверить файл на корректность |
| getInt | Получить целочисленное число |
| getPositiveInt | Получить целочисленное число *>* 0 |
| getPositiveDouble | Получить вещественное число > 0 |
| getMenuItem | Получить пользовательский выбор в меню |
| getChoice | Получить пользовательский выбор типа перечисления CHOICE |
| getSecondChoice | Получить пользовательский выбор типа перечисления SECOND\_CHOICE |
| getSize | Получить размер массива |
| showInfo | Вывод информации на консоль |
| showMenu | Вывод пользовательского меню на консоль |
| showChoice | Вывод диалогового окна на консоль |

Таблица 5 – Методы классов, составляющих программу

|  |  |  |
| --- | --- | --- |
| **Имя** | **Класс** | **Описание** |
| getSurname | Apartment | Получить фамилию пациента |
| getName | Patient | Получить имя пациента |
| getPatronymic | Patient | Получить отчество пациента |
| getAddress | Patient | Получить адрес пациента |
| getId | Patient | Получить номер медицинской карты пациента |
| getDiagnosis | Patient | Получить диагноз пациента |
| setSurname | Patient | Ввести фамилию пациента |
| setName | Patient | Ввести имя пациента |
| setPatronymic | Patient | Ввести отчество пациента |

Продолжение таблицы 5

|  |  |  |
| --- | --- | --- |
| **Имя** | **Класс** | **Описание** |
| setId | Patient | Ввести номер медицинской карты пациента |
| setDiagnosis | Patient | Ввести диагноз пациента |
| cleanCounter | Patient | Очистить счетчик номеров медицинских карт |
| showInfo | Patient | Получить информацию о пациенте |
| setAddress | Patient | Ввести адрес пациента |
| testCaseOne | UnitTest | Запустить тест с тестовым набором данных №1 |
| testCaseTwo | UnitTest | Запустить тест с тестовым набором данных №2 |
| testCaseThree | UnitTest | Запустить тест с тестовым набором данных №3 |
| testCaseFour | UnitTest | Запустить тест с тестовым набором данных №4 |
| testCaseFive | UnitTest | Запустить тест с тестовым набором данных №5 |
| runAllTests | UnitTest | Запустить все тесты |

# Блок-схемы алгоритмов программы

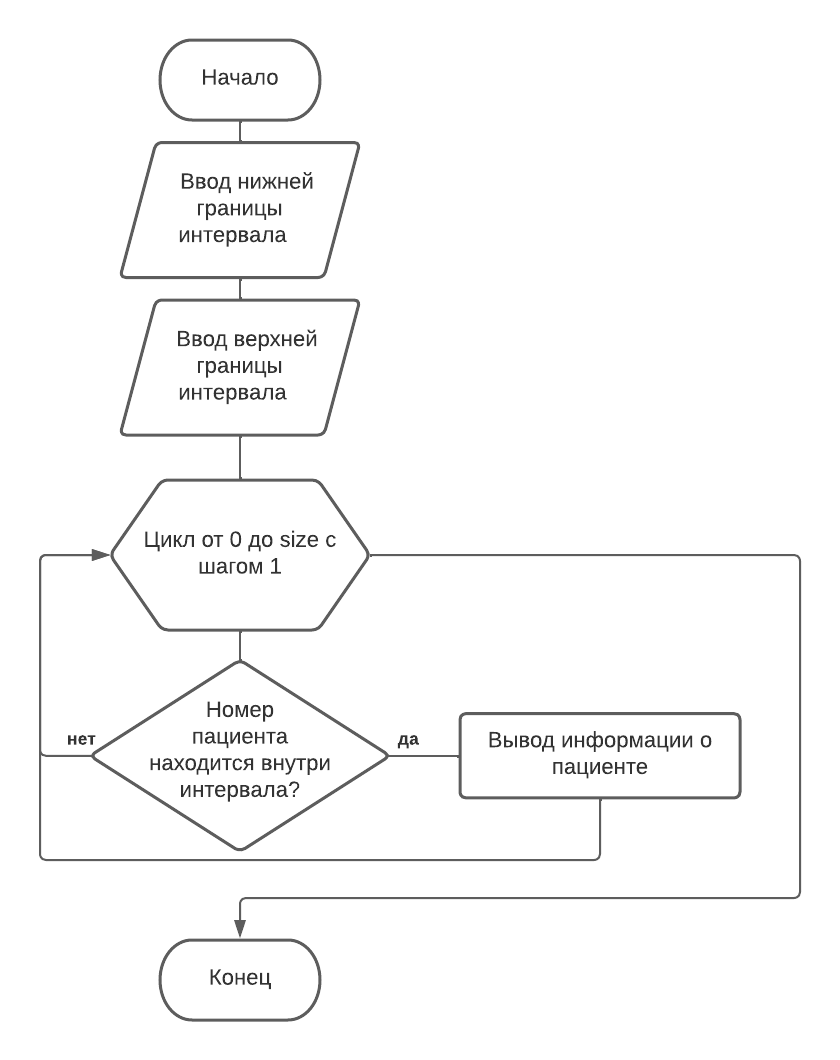


Рисунок 1 – Блок-схема алгоритма поиска пациентов, номер медицинской карты которых находится в заданном интервале

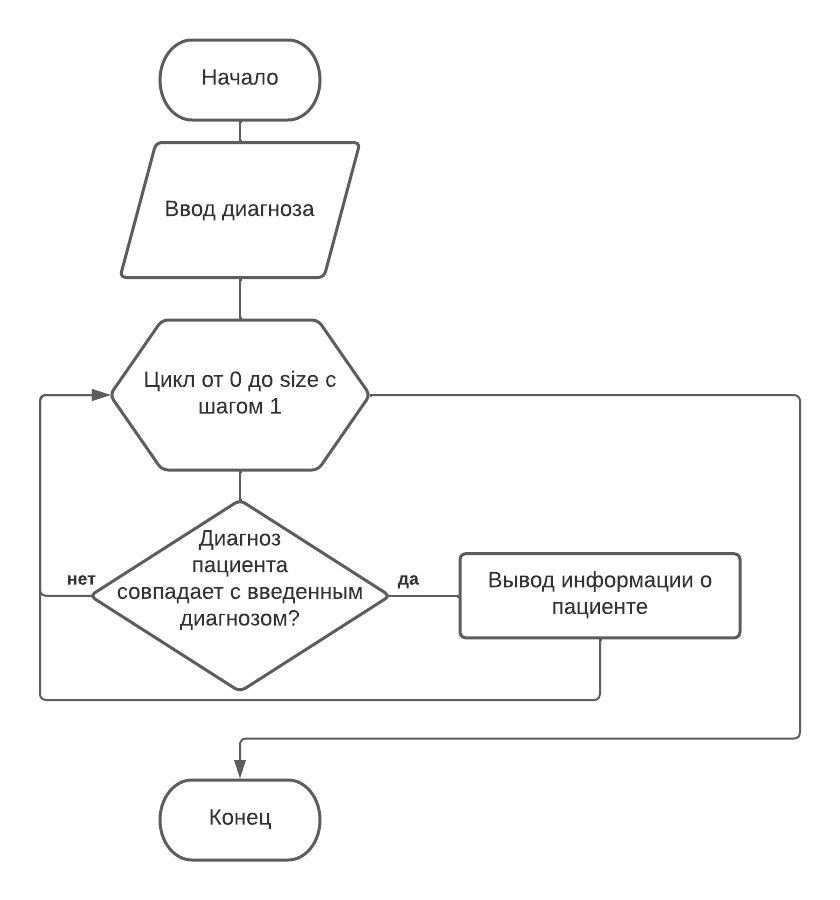


Рисунок 2 – Блок-схема алгоритма поиска информации о пациентах по заданному диагнозу

# Описание хода выполнения контрольной работы

1. В ходе контрольной работы было создано решение (Solution) в интегрированной среде разработки Microsoft Visual Studio C++ 2017. В нём был создан проект.
2. После набора текста программы выяснилось, что вывод текста на экран консольного приложения работает неправильно из-за различия кодировок консольного приложения и среды разработки. Для решения этой проблемы были использованы функции SetConsoleCP и SetConsoleOutputCP, которые устанавливают кодовую страницу win-cp 1251 в поток ввода и вывода.
3. Для сохранения данных о пациенте и работе с ними, был создан класс Patient, содержащий методы Show, Get, Set.
4. Для того, чтобы предотвратить нехватку памяти была описана функция AllocateArray, которая при нехватке памяти не позволяла создать список пациентов.
5. В начальном варианте программы при добавлении нового списка пациентов не удалялся ранее заполненный. Для этого была описана функция ClearData, которая позволяла удалять данные из памяти.
6. Так же в начальном варианте программы при считывании данных с файла не была реализована защита ввода c файла, размер массива мог быть отрицательным или равным нулю. Для решения этой проблемы была создана функция GetSizeFromFile.

# Результат работы программы

При выборе соответствующих пунктов меню, в результате, программа выводит список пациентов по заданному диагнозу или список пациентов в заданном интервале.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Пример работы функции поиска по диагнозу №1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Пример работы функции поиска по диагнозу №2

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Пример работы функции поиска пациентов в заданном интервале №1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 – Пример работы функции поиска пациентов в заданном интервале №2

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 – Пример создания нового списка пациентов

# Исходный текст программы

**[ Начало программы ]**

**[ Начало файла Algorithms.h ]**

#pragma once

#include "Patient.h"

bool AllocateArray(Patient\*\* array, int size);

void ClearData(Patient\* patients);

void ShowPatients(Patient\* patients, int size);

void FillPatients(Patient\* patients, int size);

int GetCountPatientsForDiagnosis(Patient\* patients, int size, std::string diagnosis);

int GetCountPatientsForInterval(Patient\* patients, int size, int interval\_low, int interval\_up);

Patient\* SearchByDiagnosis(Patient\* patients, Patient\* patient\_result, int size, std::string diagnosis);

Patient\* SearchInInterval(Patient\* patients, Patient\* patient\_result, int size, int interval\_low, int interval\_up);

**[ Конец файла Algorithms.h ]**

**[ Начало файла FileWork.h ]**

#pragma once

#include "Patient.h"

#include <fstream>

void SaveInFile(Patient\* patient\_result, int size);

Patient\* UploadFromFile(Patient\* patients, int size, std::ifstream &file);

int GetSizeFromFile(int size, std::ifstream & file);

bool isFileCorrect(std::string file\_path);

**[ Конец файла FileWork.h ]**

**[ Начало файла InputValidation.h ]**

#pragma once

#include "Enums.h"

int GetInt();

int GetPositiveInt();

MENU\_ITEM GetMenuItem();

CHOICE GetChoice();

SECOND\_CHOICE GetSecondChoice();

int GetSize(const char\* message);

**[ Конец файла InputValidation.h ]**

**[ Начало файла Interface.h ]**

#pragma once

void ShowInterface();

void ShowChoice();

**[ Конец файла Interface.h ]**

**[ Начало файла Enums.h ]**

#pragma once

enum class MENU\_ITEM { BY\_DIAGNOSIS = 1, BY\_ID = 2, EDIT = 3, SHOW = 4, TESTS = 5, QUIT = 6 };

enum class CHOICE { YES = 1, NO = 2};

enum class SECOND\_CHOICE { YES = 1, NO = 2 };

**[ Конец файла Enums.h ]**

**[ Начало файла Patient.h ]**

#pragma once

#include <iostream>

#include <string>

class Patient {

private:

std::string surname\_;

std::string name\_;

std::string patronymic\_;

std::string address\_;

int id\_;

std::string diagnosis\_;

static int counter\_;

public:

~Patient();

Patient(std::string surname = "", std::string name = "", std::string patronymic = "", std::string address = "", std::string diagnosis = "");

std::string getSurname();

std::string getName();

std::string getPatronymic();

std::string getAddress();

int getId();

std::string getDiagnosis();

void setSurname();

void setName();

void setPatronymic();

void setAddress();

void setId(int id);

void setDiagnosis();

void cleanCounter();

void setSurname(std::string surname);

void setName(std::string name);

void setPatronymic(std::string patronymic);

void setAddress(std::string address);

void setDiagnosis(std::string diagnosis);

void showInfo();

friend std::istream& operator>> (std::istream&, Patient&);

friend std::ostream& operator<< (std::ostream&, Patient&);

};

**[ Конец файла Patient.h ]**

**[ Начало файла Tests.h ]**

#pragma once

class UnitTest {

public:

bool testCaseOne();

bool testCaseTwo();

bool testCaseThree();

bool testCaseFour();

bool testCaseFive();

void runAllTests();

};

**[ Конец файла Tests.h ]**

**[ Начало файла Algorithms.cpp ]**

#include "Algorithms.h"

bool AllocateArray(Patient \*\*array, int size)

{

try {

\*array = new Patient[size];

}

catch (const std::bad\_alloc& error) {

std::cout << "\nОшибка выделения памяти: " << error.what() << ".\n";

std::cout << "ОС не дала выделить память. Попробуйте снова." << std::endl;

return false;

}

return true;

}

void ClearData(Patient \*patients)

{

delete[] patients;

patients = nullptr;

}

void ShowPatients(Patient \*patients, int size)

{

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

patients[i].showInfo();

}

}

void FillPatients(Patient \*patients, int size)

{

std::cout << "Заполните информацию о пациентах.\n" << std::endl;

int patient\_id = 1;

for (int i = 0; i < size; i++, patient\_id++) {

std::cout << "\tЗаполните информацию о пациенте # " << patient\_id << "." << std::endl;

std::cout << "\tФамилия: ";

patients[i].setSurname();

std::cout << "\tИмя: ";

patients[i].setName();

std::cout << "\tОтчество: ";

patients[i].setPatronymic();

std::cout << "\tАдрес: ";

patients[i].setAddress();

std::cout << "\tМедицинская карта: ";

std::cout << patients[i].getId() << std::endl;

std::cout << "\tДиагноз: ";

patients[i].setDiagnosis();

std::cout << std::endl;

}

}

int GetCountPatientsForDiagnosis(Patient\* patients, int size, std::string diagnosis)

{

int count\_patients = 0;

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

{

if (diagnosis == patients[i].getDiagnosis()) {

count\_patients++;

}

}

return count\_patients;

}

int GetCountPatientsForInterval(Patient\* patients, int size, int interval\_low, int interval\_up)

{

int count\_patients = 0;

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

{

if (patients[i].getId() >= interval\_low && patients[i].getId() <= interval\_up) {

count\_patients++;

}

}

return count\_patients;

}

Patient\* SearchByDiagnosis(Patient\* patients, Patient\* patient\_result, int size, std::string diagnosis)

{

Patient\* patient\_temp = nullptr;

int count\_patients = 0;

if (AllocateArray(&patient\_temp, size) == false) {

return patient\_result;

}

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

if (diagnosis == patients[i].getDiagnosis()) {

count\_patients++;

patient\_temp[i] = patients[i];

}

}

patient\_temp->cleanCounter();

int j = 0;

if (count\_patients == 0) {

delete[] patient\_temp;

patient\_temp = nullptr;

patient\_result = nullptr;

return patient\_result;

}

else {

if (AllocateArray(&patient\_result, count\_patients) == false) {

return patient\_result;

}

else {

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

if (j < count\_patients) {

if (patient\_temp[i].getDiagnosis() != diagnosis) {

continue;

}

else {

patient\_result[j] = patient\_temp[i];

j++;

}

}

}

}

}

delete[] patient\_temp;

patient\_temp = nullptr;

return patient\_result;

}

Patient\* SearchInInterval(Patient\* patients, Patient\* patient\_result, int size, int interval\_low, int interval\_up)

{

Patient\* patient\_temp = nullptr;

int count\_patients = 0;

if (AllocateArray(&patient\_temp, size) == false) {

return patient\_result;

}

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

if (patients[i].getId() >= interval\_low && patients[i].getId() <= interval\_up) {

count\_patients++;

patient\_temp[i] = patients[i];

}

}

patient\_temp->cleanCounter();

int j = 0;

if (count\_patients == 0) {

delete[] patient\_temp;

patient\_temp = nullptr;

patient\_result = nullptr;

return patient\_result;

}

else {

if (AllocateArray(&patient\_result, count\_patients) == false) {

return patient\_result;

}

else {

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

if (j < count\_patients) {

if (patient\_temp[i].getId() >= interval\_low && patient\_temp[i].getId() <= interval\_up) {

patient\_result[j] = patient\_temp[i];

j++;

}

else {

continue;

}

}

}

}

}

delete[] patient\_temp;

patient\_temp = nullptr;

return patient\_result;

}

**[ Конец файла Algorithms.cpp ]**

**[ Начало файла FileWork.cpp ]**

#include "FileWork.h"

#include "Enums.h"

#include "InputValidation.h"

#include "Interface.h"

#include "Algorithms.h"

#include <filesystem>

void SaveInFile(Patient\* patients, int size)

{

bool is\_data\_saved = false;

do {

std::string file\_path = "";

std::string current\_path = std::filesystem::current\_path().string() + "\\";

std::cout << "\nБез указания типа файла. Данные сохраняются в .txt файл." << std::endl;

std::cout << "Введите название файла: ";

getline(std::cin, file\_path);

file\_path = current\_path + file\_path + ".txt";

if (std::ifstream(file\_path)) {

std::cout << "Файл существует. Хотите его перезаписать?" << std::endl;

CHOICE user\_choice = static\_cast<CHOICE>(0);

ShowChoice();

user\_choice = GetChoice();

if (user\_choice == CHOICE::NO && user\_choice <= static\_cast<CHOICE>(3)) {

continue;

}

}

std::ofstream file(file\_path, std::ofstream::app);

if (!file) {

std::cout << "Запись недоступна. Попробуйте снова." << std::endl;

file.close();

continue;

}

file.close();

file.open(file\_path, std::ofstream::trunc);

file << size << std::endl;

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

file << patients[i];

}

file.close();

is\_data\_saved = true;

std::cout << "Данные сохранены." << std::endl;

system("pause");

} while (is\_data\_saved == false);

}

Patient\* UploadFromFile(Patient\* patients, int size, std::ifstream& file)

{

bool is\_data\_loaded = false;

do {

int i = 0;

for (; i < size; ++i)

{

if (!(file >> patients[i]))

{

std::cout << "Некорректные данные." << std::endl;

delete[] patients;

patients = nullptr;

return patients;

}

}

is\_data\_loaded = true;

} while (is\_data\_loaded == false);

return patients;

}

int GetSizeFromFile(int size, std::ifstream& file)

{

if (!(file >> size) || size <= 0) {

std::cout << "Неправильный размер массива." << std::endl;

return 0;

}

return size;

}

bool isFileCorrect(std::string file\_path)

{

bool is\_file\_correct = false;

do {

if (!std::ifstream(file\_path)) {

std::cout << "Файл не существует. Попробуйте снова." << std::endl;

return false;

}

std::error\_code ec;

if (std::filesystem::is\_regular\_file(file\_path, ec) == false) {

std::cout << "Адрес содержит недопустимые значения. Попробуйте снова." << std::endl;

return false;

}

is\_file\_correct = true;

} while (is\_file\_correct == false);

return true;

}

**[ Конец файла FileWork.cpp ]**

**[ Начало файла InputValidation.cpp ]**

#include "Enums.h"

#include "InputValidation.h"

#include <iostream>

template<typename T>

T GetInput()

{

T user\_input = 0;

std::cin >> user\_input;

while (std::cin.fail()) {

std::cout << "Некорректный ввод. Попробуйте снова." << std::endl;

std::cin.clear();

std::cin.ignore(INT\_MAX, '\n');

std::cin >> user\_input;

}

std::cin.ignore(INT\_MAX, '\n');

return user\_input;

}

int GetInt()

{

return GetInput<int>();

}

int GetPositiveInt()

{

int user\_input = 0;

do {

user\_input = GetInt();

if (user\_input < 0) {

std::cout << "Ошибка. Введите положительное число." << std::endl;

}

} while (user\_input < 0);

return user\_input;

}

MENU\_ITEM GetMenuItem()

{

return static\_cast<MENU\_ITEM>(GetInt());

}

CHOICE GetChoice()

{

return static\_cast<CHOICE>(GetInt());

}

SECOND\_CHOICE GetSecondChoice()

{

return static\_cast<SECOND\_CHOICE>(GetInt());

}

int GetSize(const char\* message)

{

int size = 0;

do {

std::cout << message;

size = GetInt();

if (size <= 0) {

std::cout << "Ошибка. Размер списка пациентов не может быть равен или меньше 0." << std::endl;

}

} while (size <= 0);

return size;

}

**[ Конец файла InputValidation.cpp ]**

**[ Начало файла Interface.cpp ]**

#include "Interface.h"

#include <iostream>

void ShowMenu()

{

std::cout << "Меню:" << std::endl

<< "1. Список пациентов с данным диагнозом" << std::endl

<< "2. Список пациентов с медицинскими картами в заданном интервале" << std::endl

<< "3. Создать или удалить лист пациентов" << std::endl

<< "4. Показать данные о пациентах" << std::endl

<< "5. Запустить модульные тесты" << std::endl

<< "6. Выход" << std::endl

<< "> ";

}

void ShowInformation()

{

std::cout << "Контрольная работа #1, вариант 8" << std::endl

<< "И. С. Рябченко, группа 415a, СПбГТИ (ТУ)" << std::endl << std::endl

<< "Описание программы:" << std::endl

<< "Программа позволяет создать или загрузить из файла список пациентов, и предоставляет возможность сортировать список " << std::endl

<< "пациентов по заданному диагнозу или выводить список пациентов в заданном интервале номеров медицинских карт." << std::endl << std::endl;

}

void ShowChoice()

{

std::cout << "1. Да" << std::endl

<< "2. Нет" << std::endl

<< "> ";

}

**[ Конец файла Interface.cpp ]**

**[ Начало файла Patient.cpp ]**

#include "Patient.h"

#include <string>

int Patient::counter\_ = 0;

Patient::~Patient() = default;

Patient::Patient(std::string surname, std::string name, std::string patronymic, std::string address, std::string diagnosis)

{

surname\_ = surname;

name\_ = name;

patronymic\_ = patronymic;

address\_ = address;

id\_ = ++counter\_;

diagnosis\_ = diagnosis;

}

std::string Patient::getSurname()

{

return surname\_;

}

std::string Patient::getName()

{

return name\_;

}

std::string Patient::getPatronymic()

{

return patronymic\_;

}

std::string Patient::getAddress()

{

return address\_;

}

int Patient::getId()

{

return id\_;

}

std::string Patient::getDiagnosis()

{

return diagnosis\_;

}

void Patient::setSurname()

{

std::string surname;

getline(std::cin, surname);

surname\_ = surname;

}

void Patient::setSurname(std::string surname)

{

surname\_ = surname;

}

void Patient::setName()

{

std::string name;

getline(std::cin, name);

name\_ = name;

}

void Patient::setName(std::string name)

{

name\_ = name;

}

void Patient::setPatronymic()

{

std::string patronymic;

getline(std::cin, patronymic);

patronymic\_ = patronymic;

}

void Patient::setPatronymic(std::string patronymic)

{

patronymic\_ = patronymic;

}

void Patient::setAddress()

{

std::string address;

getline(std::cin, address);

address\_ = address;

}

void Patient::setAddress(std::string address)

{

address\_ = address;

}

void Patient::setId(int id)

{

id\_ = id;

}

void Patient::setDiagnosis()

{

std::string diagnosis;

getline(std::cin, diagnosis);

diagnosis\_ = diagnosis;

}

void Patient::setDiagnosis(std::string diagnosis)

{

diagnosis\_ = diagnosis;

}

void Patient::cleanCounter()

{

counter\_ = 0;

}

void Patient::showInfo()

{

std::cout << "Информация о пациенте #" << id\_ << std::endl << std::endl;

std::cout << "\tФамилия: " << surname\_;

std::cout << "\n\tИмя: " << name\_;

std::cout << "\n\tОтчество: " << patronymic\_;

std::cout << "\n\tАдрес: " << address\_;

std::cout << "\n\tМедицинская карта: " << id\_;

std::cout << "\n\tДиагноз: " << diagnosis\_;

std::cout << "\n\n";

}

std::istream& operator>> (std::istream& in, Patient& p)

{

getline(in, p.surname\_);

getline(in, p.name\_);

getline(in, p.patronymic\_);

getline(in, p.address\_);

getline(in, p.diagnosis\_);

return in;

}

std::ostream& operator<<(std::ostream& out, Patient& p)

{

out << p.surname\_ << std::endl << p.name\_ << std::endl << p.patronymic\_ <<

std::endl << p.address\_ << std::endl << p.diagnosis\_ << std::endl;

return out;

}

**[ Конец файла Patient.cpp ]**

**[ Начало файла Tests.cpp ]**

#include "Tests.h"

#include "Patient.h"

#include "Algorithms.h"

bool UnitTest::testCaseOne()

{

const int SIZE = 2;

const int ANSWER = 1;

Patient\* patients = nullptr;

std::string diagnosis = "Грипп";

std::string second\_diagnosis = "Пневмония";

if (AllocateArray(&patients, SIZE) == false) {

std::cout << "Тест 1 провален. Ошибка выделения памяти #1" << std::endl;

return false;

}

patients[0].setDiagnosis(diagnosis);

patients[1].setDiagnosis(second\_diagnosis);

for (int i = 1; i <= SIZE; i++)

{

patients[i - 1].setId(i);

}

Patient\* patients\_result = nullptr;

int count\_patients = GetCountPatientsForDiagnosis(patients, SIZE, diagnosis);

patients->cleanCounter();

if (AllocateArray(&patients\_result, count\_patients) == false) {

std::cout << "Тест 1 провален. Ошибка выделения памяти #2" << std::endl;

return false;

}

patients\_result = SearchByDiagnosis(patients, patients\_result, SIZE, diagnosis);

patients\_result->cleanCounter();

if (count\_patients == ANSWER) {

if (patients\_result[0].getDiagnosis() == diagnosis)

{

return true;

}

else {

std::cout << "Тест 1 провален." << std::endl;

std::cout << "Ожидалось: " << ANSWER << " пациент с диагнозом " << diagnosis << "." << std::endl;

std::cout << "Получено: " << count\_patients << " пациента с диагнозами: " << patients\_result[0].getDiagnosis() << std::endl;

return false;

}

}

else {

std::cout << "Тест 1 провален." << std::endl;

std::cout << "Ожидалось: количество пациентов = " << ANSWER << std::endl;

std::cout << "Получено: количество пациентов = " << count\_patients << std::endl << std::endl;

return false;

}

return false;

}

bool UnitTest::testCaseTwo()

{

const int SIZE = 3;

const int ANSWER = 2;

Patient\* patients = nullptr;

std::string diagnosis = "Грипп";

std::string second\_diagnosis = "Пневмония";

if (AllocateArray(&patients, SIZE) == false) {

std::cout << "Тест 2 провален. Ошибка выделения памяти #1" << std::endl;

return false;

}

patients[0].setDiagnosis(diagnosis);

patients[1].setDiagnosis(second\_diagnosis);

patients[2].setDiagnosis(diagnosis);

for (int i = 1; i <= SIZE; i++)

{

patients[i - 1].setId(i);

}

Patient\* patients\_result = nullptr;

int count\_patients = GetCountPatientsForDiagnosis(patients, SIZE, diagnosis);

patients->cleanCounter();

if (AllocateArray(&patients\_result, count\_patients) == false) {

std::cout << "Тест 2 провален. Ошибка выделения памяти #2" << std::endl;

return false;

}

patients\_result = SearchByDiagnosis(patients, patients\_result, SIZE, diagnosis);

patients\_result->cleanCounter();

if (count\_patients == ANSWER) {

if (patients\_result[0].getDiagnosis() == diagnosis && patients\_result[1].getDiagnosis() == diagnosis)

{

return true;

}

else {

std::cout << "Тест 2 провален." << std::endl;

std::cout << "Ожидалось: " << ANSWER << " пациент с диагнозом " << diagnosis << "." << std::endl;

std::cout << "Получено: " << count\_patients << " пациента с диагнозами: " << patients\_result[0].getDiagnosis()

<< ", " << patients\_result[1].getDiagnosis() << std::endl;

return false;

}

}

else {

std::cout << "Тест 2 провален." << std::endl;

std::cout << "Ожидалось: количество пациентов = " << ANSWER << std::endl;

std::cout << "Получено: количество пациентов = " << count\_patients << std::endl << std::endl;

return false;

}

return false;

}

bool UnitTest::testCaseThree()

{

const int SIZE = 5;

const int ANSWER = 3;

Patient\* patients = nullptr;

std::string diagnosis = "Грипп";

std::string second\_diagnosis = "Пневмония";

std::string third\_diagnosis = "Анемия";

if (AllocateArray(&patients, SIZE) == false) {

std::cout << "Тест 3 провален. Ошибка выделения памяти #1" << std::endl;

return false;

}

patients[0].setDiagnosis(diagnosis);

patients[1].setDiagnosis(second\_diagnosis);

patients[2].setDiagnosis(diagnosis);

patients[3].setDiagnosis(diagnosis);

patients[4].setDiagnosis(second\_diagnosis);

for (int i = 1; i <= SIZE; i++)

{

patients[i - 1].setId(i);

}

Patient\* patients\_result = nullptr;

int count\_patients = GetCountPatientsForDiagnosis(patients, SIZE, diagnosis);

patients->cleanCounter();

if (AllocateArray(&patients\_result, count\_patients) == false) {

std::cout << "Тест 3 провален. Ошибка выделения памяти #2" << std::endl;

return false;

}

patients\_result = SearchByDiagnosis(patients, patients\_result, SIZE, diagnosis);

patients\_result->cleanCounter();

if (count\_patients == ANSWER) {

if (patients\_result[0].getDiagnosis() == diagnosis && patients\_result[1].getDiagnosis() == diagnosis

&& patients\_result[2].getDiagnosis() == diagnosis)

{

return true;

}

else {

std::cout << "Тест 3 провален." << std::endl;

std::cout << "Ожидалось: " << ANSWER << " пациент с диагнозом " << diagnosis << "." << std::endl;

std::cout << "Получено: " << count\_patients << " пациента с диагнозами: " << patients\_result[0].getDiagnosis()

<< ", " << patients\_result[1].getDiagnosis() << ", " << patients\_result[2].getDiagnosis() << std::endl;

return false;

}

}

else {

std::cout << "Тест 3 провален." << std::endl;

std::cout << "Ожидалось: количество пациентов = " << ANSWER << std::endl;

std::cout << "Получено: количество пациентов = " << count\_patients << std::endl << std::endl;

return false;

}

return false;

}

bool UnitTest::testCaseFour()

{

const int SIZE = 2;

const int ANSWER = 1;

int interval\_low = 1;

int interval\_up = 1;

Patient\* patients = nullptr;

if (AllocateArray(&patients, SIZE) == false) {

std::cout << "Тест 4 провален. Ошибка выделения памяти #1" << std::endl;

return false;

}

Patient\* patients\_result = nullptr;

int count\_patients = GetCountPatientsForInterval(patients, SIZE, interval\_low, interval\_up);

patients->cleanCounter();

if (AllocateArray(&patients\_result, count\_patients) == false) {

std::cout << "Тест 4 провален. Ошибка выделения памяти #2" << std::endl;

return false;

}

patients\_result = SearchInInterval(patients, patients\_result, SIZE, interval\_low, interval\_up);

patients\_result->cleanCounter();

if (count\_patients == ANSWER) {

if (patients\_result[0].getId() == 1) {

return true;

}

else {

std::cout << "Тест 4 провален." << std::endl;

std::cout << "Ожидалось: " << ANSWER << " пациента в интервале " << interval\_low << "-" << interval\_up << "." << std::endl;

std::cout << "Получено: " << count\_patients << " пациента с картами: " << patients\_result[0].getId() << std::endl;

return false;

}

}

else {

std::cout << "Тест 4 провален." << std::endl;

std::cout << "Ожидалось: количество пациентов = " << ANSWER << std::endl;

std::cout << "Получено: количество пациентов = " << count\_patients << std::endl << std::endl;

return false;

}

}

bool UnitTest::testCaseFive()

{

const int SIZE = 4;

const int ANSWER = 2;

int interval\_low = 1;

int interval\_up = 2;

Patient\* patients = nullptr;

if (AllocateArray(&patients, SIZE) == false) {

std::cout << "Тест 5 провален. Ошибка выделения памяти #1" << std::endl;

return false;

}

Patient\* patients\_result = nullptr;

int count\_patients = GetCountPatientsForInterval(patients, SIZE, interval\_low, interval\_up);

patients->cleanCounter();

if (AllocateArray(&patients\_result, count\_patients) == false) {

std::cout << "Тест 5 провален. Ошибка выделения памяти #2" << std::endl;

return false;

}

patients\_result = SearchInInterval(patients, patients\_result, SIZE, interval\_low, interval\_up);

patients\_result->cleanCounter();

if (count\_patients == ANSWER) {

if (patients\_result[0].getId() == 1 && patients\_result[1].getId() == 2) {

return true;

}

else {

std::cout << "Тест 5 провален." << std::endl;

std::cout << "Ожидалось: " << ANSWER << " пациента в интервале " << interval\_low << "-" << interval\_up << "." << std::endl;

std::cout << "Получено: " << count\_patients << " пациента с картами: " << patients\_result[0].getId() << ", " << patients\_result[1].getId() << std::endl;

return false;

}

}

else {

std::cout << "Тест 5 провален." << std::endl;

std::cout << "Ожидалось: количество пациентов = " << ANSWER << std::endl;

std::cout << "Получено: количество пациентов = " << count\_patients << std::endl << std::endl;

return false;

}

}

void UnitTest::runAllTests()

{

UnitTest test;

if (test.testCaseOne() && test.testCaseTwo() && test.testCaseThree() && test.testCaseFour() && test.testCaseFive())

{

std::cout << "\nВсе модульные тесты пройдены!\n" << std::endl;

}

else

{

std::cout << "\nМодульные тесты не пройдены.\n" << std::endl;

}

}

**[ Конец файла Tests.cpp ]**

**[ Начало файла Main.cpp]**

// Main.cpp

// Контрольная работа № 1, вариант 8

// Инкапсуляция. Классы в С++ и средства их построения

// Студент группы 415а, Рябченко Иван Сергеевич. 2022 год

// Необходимо разработать класс для указанной предметной области.

// Доступ к данным реализовать с помощью методов Set, Get, Show.

// Пациент: фамилия, имя, отчество, адрес, номер медицинской карты, диагноз.

// Создать массив объектов. Реализовать возможность получения:

// – списка пациентов, имеющих данный диагноз,

// – списка пациентов, номер медицинской карты которых находится в заданном интервале.

#include "InputValidation.h"

#include "Patient.h"

#include "Algorithms.h"

#include "FileWork.h"

#include "Enums.h"

#include "Interface.h"

#include "Tests.h"

#include <Windows.h>

int main()

{

Patient\* patients = nullptr;

Patient\* patients\_result = nullptr;

int size = 0;

int interval\_low = 0;

int interval\_up = 0;

int count\_patients = 0;

std::string diagnosis = "";

MENU\_ITEM user\_menu\_choice = static\_cast<MENU\_ITEM>(0);

SECOND\_CHOICE user\_second\_choice = static\_cast<SECOND\_CHOICE>(0);

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

ShowInformation();

do {

std::string file\_path = "";

ShowMenu();

user\_menu\_choice = GetMenuItem();

switch (user\_menu\_choice) {

case MENU\_ITEM::BY\_DIAGNOSIS:

if (!patients) {

std::cout << "\nСписок пациентов пуст." << std::endl << std::endl;

break;

}

else {

std::cout << "\nНайти пациентов по заданному диагнозу." << std::endl;

std::cout << "\nВведите диагноз: ";

getline(std::cin, diagnosis);

count\_patients = GetCountPatientsForDiagnosis(patients, size, diagnosis);

patients\_result = SearchByDiagnosis(patients, patients\_result, size, diagnosis);

CHOICE user\_choice = static\_cast<CHOICE>(0);

if (patients\_result) {

bool is\_data\_saved = false;

std::cout << "Информация о пациентах с диагнозом " << diagnosis << ".\n\n";

ShowPatients(patients\_result, count\_patients);

patients\_result->cleanCounter();

do

{

std::cout << std::endl << "Сохранить в файл?" << std::endl;

ShowChoice();

user\_choice = GetChoice();

switch (user\_choice) {

case CHOICE::YES:

SaveInFile(patients\_result, count\_patients);

is\_data\_saved = true;

break;

case CHOICE::NO:

std::cout << std::endl << "Вы решили не сохранять данные в файл.\n" << std::endl;

is\_data\_saved = true;

break;

default:

std::cout << "Неправильный ввод, попробуйте снова." << std::endl;

break;

}

} while (user\_choice != CHOICE::NO && is\_data\_saved == false);

}

else

{

std::cout << "\nНет пациентов с данным диагнозом.\n" << std::endl;

}

}

break;

case MENU\_ITEM::BY\_ID:

if (!patients) {

std::cout << "\nСписок пациентов пуст." << std::endl << std::endl;

break;

}

else {

std::cout << "\nНайти пациентов в интервале " << 1 << "-" << size << "." << std::endl;

do

{

std::cout << "Введите нижний предел интервала: ";

interval\_low = GetPositiveInt();

std::cout << "Введите верхний предел интервала: ";

interval\_up = GetPositiveInt();

if (interval\_low > interval\_up) {

std::cout << "Нижняя граница интервала не может быть больше верхней. Попробуйте снова." << std::endl << std::endl;

}

if (interval\_up > size) {

std::cout << "Верхняя граница интервала не может быть больше " << size << "." << std::endl << std::endl;

}

} while (interval\_low > interval\_up || interval\_up > size);

count\_patients = GetCountPatientsForInterval(patients, size, interval\_low, interval\_up);

patients\_result = SearchInInterval(patients, patients\_result, size, interval\_low, interval\_up);

CHOICE user\_choice = static\_cast<CHOICE>(0);

if (patients\_result) {

bool is\_data\_saved = false;

std::cout << "Информация о пациентах с картами в интервале " << interval\_low << "-" << interval\_up << ".\n\n";

ShowPatients(patients\_result, count\_patients);

patients\_result->cleanCounter();

do

{

std::cout << std::endl << "Сохранить в файл?" << std::endl;

ShowChoice();

user\_choice = GetChoice();

switch (user\_choice) {

case CHOICE::YES:

SaveInFile(patients\_result, count\_patients);

is\_data\_saved = true;

break;

case CHOICE::NO:

std::cout << std::endl << "Вы решили не сохранять данные в файл." << std::endl;

is\_data\_saved = true;

break;

default:

std::cout << "Неправильный ввод, попробуйте снова." << std::endl << std::endl;

break;

}

} while (user\_choice != CHOICE::NO && is\_data\_saved == false);

}

else

{

std::cout << "\nВ данном интервале нет пациентов.\n" << std::endl;

}

}

break;

case MENU\_ITEM::EDIT: {

if (patients) {

std::cout << "\n[!] У вас уже есть заполненный список пациентов." << std::endl << std::endl;

std::cout << "Вы можете создать новый лист пациентов." << std::endl;

std::cout << "Создать новый и удалить старые данные?" << std::endl << std::endl;

}

else

{

std::cout << "\nВы можете создать новый лист пациентов." << std::endl;

std::cout << "Создать новый лист?" << std::endl;

}

CHOICE user\_choice = static\_cast<CHOICE>(0);

ShowChoice();

user\_choice = GetChoice();

switch (user\_choice) {

case CHOICE::YES: {

bool is\_data\_loaded = false;

if (patients) {

ClearData(patients);

ClearData(patients\_result);

std::cout << "Прошлые данные удалены." << std::endl;

}

do {

std::cout << "\nЗагрузить данные из файла?" << std::endl;

ShowChoice();

user\_second\_choice = GetSecondChoice();

switch (user\_second\_choice) {

case SECOND\_CHOICE::YES: {

std::cout << "\nЗагрузить из: ";

getline(std::cin, file\_path);

if (isFileCorrect(file\_path) == false) {

continue;

}

else {

std::ifstream file(file\_path);

size = GetSizeFromFile(size, file);

if (size <= 0)

{

break;

}

if (AllocateArray(&patients, size) == false) {

break;

}

file.ignore();

patients = UploadFromFile(patients, size, file);

file.close();

}

patients->cleanCounter();

is\_data\_loaded = true;

std::cout << std::endl;

break;

}

case SECOND\_CHOICE::NO:

std::cout << "\nВы решили заполнить данные о пациентах самостоятельно." << std::endl;

size = GetSize("Введите количество пациентов: ");

std::cout << std::endl;

if (AllocateArray(&patients, size) == false) {

break;

}

FillPatients(patients, size);

patients->cleanCounter();

break;

}

} while ((user\_second\_choice != SECOND\_CHOICE::NO || user\_second\_choice >= static\_cast<SECOND\_CHOICE>(3)) && is\_data\_loaded == false);

break;

}

case CHOICE::NO:

std::cout << "\nВы решили не добавлять пациентов.\n" << std::endl;

break;

default:

std::cout << "\nНеправильный ввод, попробуйте снова.\n" << std::endl;

break;

}

}

break;

case MENU\_ITEM::SHOW:

if (!patients) {

std::cout << "\nСписок пациентов пуст." << std::endl << std::endl;

break;

}

else {

std::cout << "\nИнформация о ваших пациентах." << std::endl << std::endl;

ShowPatients(patients, size);

break;

}

break;

case MENU\_ITEM::TESTS:

UnitTest test;

test.RunAllTests();

patients->cleanCounter();

break;

case MENU\_ITEM::QUIT:

std::cout << "\nВыход из программы.";

break;

default:

std::cout << "\nНеправильный ввод. Попробуйте снова.\n" << std::endl;

break;

}

} while (user\_menu\_choice != MENU\_ITEM::QUIT);

return 0;

}

**[ Конец файла Main.cpp ]**

**[ Конец программы ]**