

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) | |  | Системы автоматизированного проектирования | | |
| 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 № 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 423 |  |  |  | Ефремов Иван Андреевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Макарук Роман Валерьевич |
|  |  |  |  | Федин Алексей Константинович |

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc133082387)

[2 Исходные данные 3](#_Toc133082388)

[3 Особые ситуации 3](#_Toc133082389)

[4 Математические методы и алгоритмы решения задач 4](#_Toc133082390)

[5 Форматы представления данных 4](#_Toc133082391)

[6 Структура программы 6](#_Toc133082392)

[7 Блок-схема алгоритма решения задачи 9](#_Toc133082393)

[8 Описание хода выполнения 10](#_Toc133082394)

[9 Результаты работы программы 11](#_Toc133082395)

[10 Выводы по заданию 13](#_Toc133082396)

[11 Исходный код полученного программного решения 13](#_Toc133082397)

## 1 Постановка задачи

Разработать класс для указанной предметной области. Доступ к данным реализовать с помощью методов Set, Get, Show. Реализовать программу не используя контейнеры и алгоритмы библиотеки STL или аналогичных сторонних библиотек.

## 2 Исходные данные

Исходные данные состоят из фамилии, имени, отчества, адреса, диагноза и номера медицинской карты пациента. Данные вводятся пользователем с консоли или из файла.

## 3 Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

-если пользователь ввёл количество пациентов меньше 1, то программа попросит его ввести данные хотя бы об одном пациенте;

-если пользователь ввёл номер медицинской карты меньше 1, то программа попросит ввести номер медицинской карты как натуральное число;

-если при запросе цифры пользователь ввёл другие символы, то программа укажет на необходимость ввода целых чисел и попросит повторить ввод;

-если при запросе выбрать пункт меню пользователь введет несуществующий вариант, то программа укажет, что такого пункта меню нет, выведет возможные для выбора пункты меню и попросит пользователя ввести номер пункта еще раз;

-если при попытке отфильтровать пациентов пользователь выберет такие условия, под которые не подходит ни один пациент, программа сообщит об этом;

-если после предложения пользователю сохранить отфильтрованные результаты в файл, пользователь введет что-то кроме “n” или “y”, то программа сообщит, какой должен быть формат ввода и попросит пользователя ввести свой выбор еще раз;

-если пользователь попытается открыть несуществующий файл для чтения, программа сообщит, что такого файла не существует, и попросит пользователя указать файл для чтения еще раз;

-если пользователь выбрал фильтрацию по номеру карты в определённом отрезке и указал нижнюю границу фильтра больше верхней, программа сообщит об этом и попросит пользователя заново ввести границы отрезка;

-если пользователь выбрал ввод с файла и в одной из строк файла будет указано неверное количество полей класса, программа сообщит об этом и попросит пользователя открыть новый файл или изменить данные в этом файле;

-если пользователь выбрал ввод с файла и в одной из строк значение поля «номер медицинской карты» не будет целым числом или будет меньше или равно нулю, то программа сообщит об этом и попросит пользователя открыть новый файл или изменить данные в этом файле;

## 4 Математические методы и алгоритмы решения задач

Согласно постановке задачи для получения необходимых списков будут использоваться операторы “>”, “<” и “==”.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 Форматы представления данных Таблица 1 - Основные переменные, используемые в программе   |  |  |  | | --- | --- | --- | | Имя | Тип | Описание | | actionTop | int | Выбор пользователя в меню верхнего уровня | | actionBottom | int | Выбор пользователя в меню нижнего уровня | | patients | Patient\* | Массив с данными о пациентах | | numberOfPatients | Unsigned int | Количество пациентов в массиве |   Таблица 2 - Класс Patient   |  |  |  |  | | --- | --- | --- | --- | | Имя | Описание | Модификатор | Тип | | name | Имя пациента | private | string | | surname | Фамилия пациента | private | string | | patronymic | Отчество пациента | private | string | | address | Адрес пациента | private | string | | diagnosis | Диагноз пациента | private | string | | numberOfMedicalCard | Номер мед. Карты пациента | private | Unsigned int |   Данные, записанные в файле для чтения, должны быть представлены таким образом:  Имя1 Фамилия1 Отчество1 Адрес1 Диагноз1 НомерКарты1  .  .  Имяn Фамилияn Отчествоn Адресn Диагнозn НомерКартыn  Данные о каждом пациенте должны быть расположены на разных строках, причем каждое поле класса должно быть разделено пробелом.  Данные в файле вывода записываются в формате:  Patients with «Условие фильтрации» «Ключ фильтрации»:  #1: ФИО1  .  .  #n: ФИОn 6 Структура программы Таблица 3 - Модули программы   |  |  | | --- | --- | | Название модуля | Описание | | Start | Запуск основных функций программы | | ConsoleInterface | Зацикленное двухуровневое меню, открытие файлов, вывод и запись необходимой информации в консоль и в файл | | GetInfo | Функции для считывания данных с консоли, а также подключение основных библиотек | | DataInput | Заполнение массива с консоли или из файла, разделение строк по пробелам и сортировка массива по возрастанию номера карты | | Tasks | Фильтрация данных из массива по выбору пользователя, функция для сравнения переменных класса string без учёта регистра | | Patient | Описание класса | | ModuleTests | Модульные тесты для проверки корректности работы функций программы |   Таблица 4 - Функции класса Patient   |  |  | | --- | --- | | Имя | Описание | | ShowFIO | Вывести ФИО пациента | | SetInformation | Установить значение в поля класса | | GetFIO | Передать ФИО пациента | | GetMedicalCardNumber | Передать номер мед. Карты пациента | | GetDiagnosis | Передать диагноз пациента | |  |

Таблица 5 – Основные функции модуля ConsoleInterface

|  |  |
| --- | --- |
| Имя | Описание |
| StartProgram | Зацикленное меню верхнего уровня, в котором находятся основные функции программы |
| WorkWithConsole | Заполнение массива данными из консоли и вывод результата фильтрации, выбранной пользователем, в консоль и, по выбору пользователя, в файл |
| WorkWithFile | Заполнение массива данными из файла и вывод результата фильтрации, выбранной пользователем, в файл |
| OpenFile | Открытие файла для чтения или записи, по выбору пользователя |
| SaveResults | Предложение пользователю сохранить результат фильтрации в файл |
| OutputConsole | Вывод результата фильтрации в консоль |
| OutputFile | Запись результата фильтрации в файл |

Таблица 6 – Функции модуля GetInfo

|  |  |
| --- | --- |
| Имя | Описание |
| IgnoreLine | При неверном вводе пользователя очищает поток ввода |
| GetInt | Считывание целого числа из консоли |
| GetString | Считывание строки с консоли |
| GetLink | Считывание пути к файлу с консоли |
| GetChar | Считывание ответа пользователя в формате y/n |

Таблица 7 – Функции модуля DataInput

|  |  |
| --- | --- |
| Имя | Описание |
| SortData | Сортировка массива по возрастанию номера карты пациента |
| SplitString | Разделение строки по пробелам |
| ConsoleInput | Ввод данных с консоли и заполнение массива |
| FileInput | Ввод данных из файла и заполнение массива |

Таблица 8 – Функции модуля Tasks

|  |  |
| --- | --- |
| Имя | Описание |
| Comparison | Сравнение переменных класса string без учета регистра символов |
| SearchDiagnosis | Фильтр пациентов по диагнозу, выбранному пользователем |
| OutputDiagnosis | Вывод результата фильтрации по диагнозу в консоль или в файл, по выбору заполнения массива |
| SearchCardNumber | Фильтр пациентов по вхождению их номера карты в отрезок, заданный пользователем |
| OutputCardNumber | Вывод результата фильтрации по номеру карты в консоль или в файл, по выбору заполнения массива |

Таблица 9 – Функции модуля ModuleTests

|  |  |
| --- | --- |
| Имя | Описание |
| Check | Проверка соответствия считанных данных с данными из программы |
| Test1 | Проверка корректности ввода данных с файла |
| Test2 | Проверка корректности работы фильтра по диагнозу |
| Test3 | Проверка корректности работы фильтра по номеру карты |
| Test4 | Проверка корректности подсчёта строк в файле |
| Test5 | Проверка корректности работы считывания данных с консоли |
| LaunchAllTests | Запуск всех тестов |

## 7 Блок-схема алгоритма решения задачи

На рисунке 1 представлена блок-схема алгоритма решения задачи.



Рисунок 1 – Блок-схема алгоритма решения задачи

## 8 Описание хода выполнения

Для выполнения задания необходимо дополнительно изучить базовые правила языка С++, обучиться написанию классов, методов, созданию функций, работающих с полями класса, а также изучить разбиение программы на модули.

После обучения этому приступили к реализации основных задач программы – созданию класса, заполнению полей и разбиению структуры программы на модули.

Во время работы программа предлагает пользователю два варианта фильтрации массива пациентов: по диагнозу и по номеру медицинской карты. Пользователь вводит необходимый ему диагноз или номер карты. После чего программа перебором отбирает пациентов с таким диагнозом или номером карты, после чего выводит их на экран.

## 9 Результаты работы программы

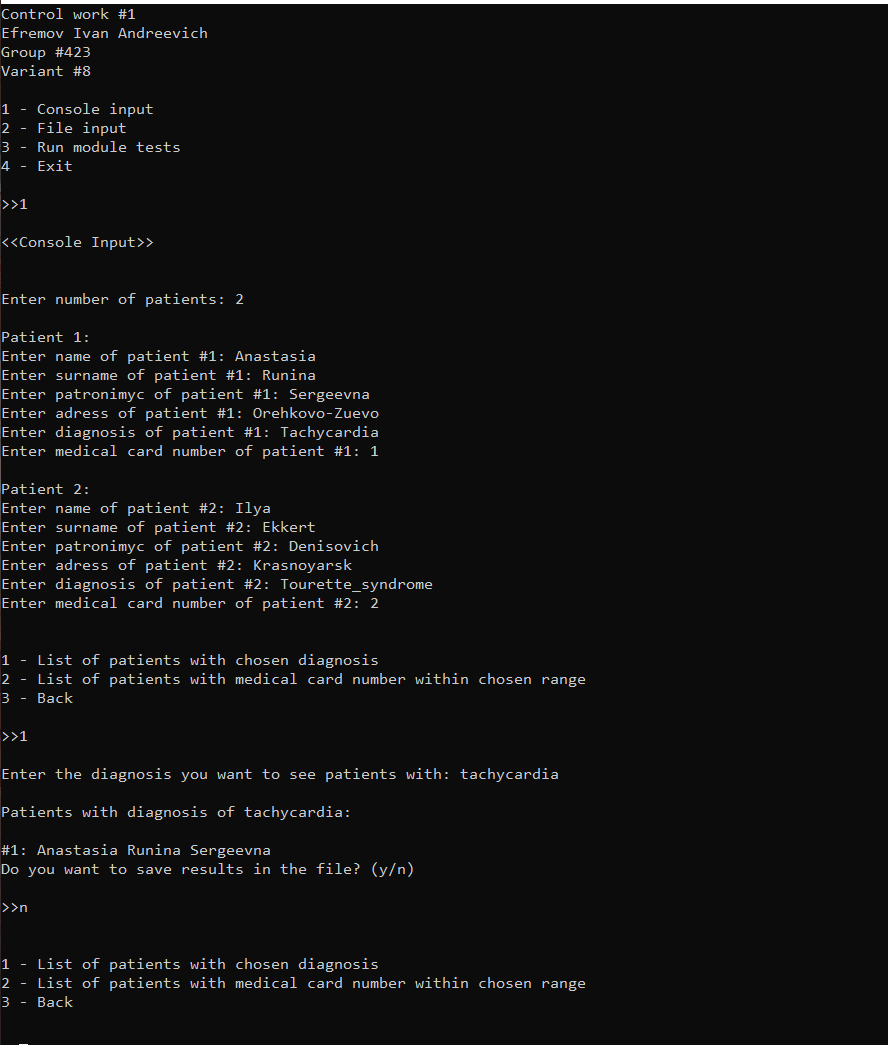


Рисунок 2 – Экранная копия результата работы разработанной программы

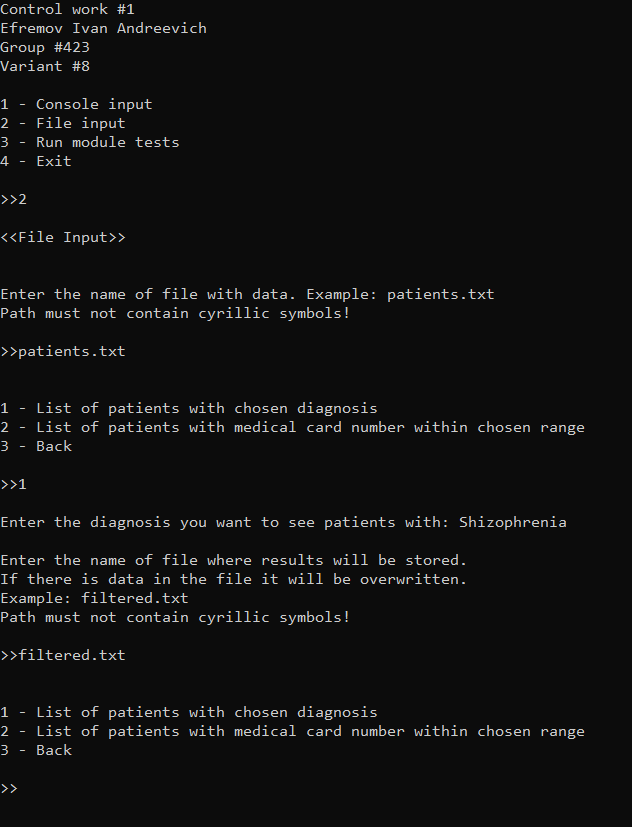


Рисунок 3 – Экранная копия результата работы разработанной программы

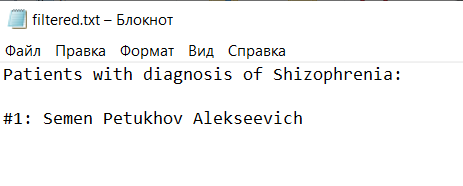


Рисунок 4 – Результат работы программы, записанный в файл

## 10 Выводы по заданию

В ходе выполнения задания были изучены методы работы с языком С++ и реализацией классов, а также были получены навыки работы с многофайловыми проектами. Были устранены недостатки в знаниях библиотек и получены новые навыки программирования.

## 11 Исходный код полученного программного решения

Файл Start:

#include "ConsoleInterface.h"

int main() { setlocale(LC\_ALL, "Russian"); StartProgram();}

Файл ConsoleInterface.h:

#pragma once

#include "Patient.h"

#include <fstream>

enum WorkWithFiles{input, output};

enum TopMenu{console = 1, file, module, quit};

enum BottomMenu{chooseDiagnosis = 1, chooseRange, back};

void WorkWithConsole(Patient \*patients, int &actionBottom);

void WorkWithFile(Patient\* patient, int& actionBottom);

void Fio();

void OptionsTop();

void InputOption(int option);

void OptionsBottom();

void IncorrectOption();

char SaveResults(fstream& fout);

void OutputConsole(Patient a, int cnt);

void OutputFile(fstream &fout, string o, int cnt);

string OpenFile(int option, fstream &file);

void StartProgram();

Файл ConsoleInterface.cpp:

#include "ConsoleInterface.h"

#include "ModuleTests.h"

#include "Tasks.h"

//если пользователь выбрал ввод с консоли

void WorkWithConsole(Patient\* patients, int& actionBottom)

{

ui numberOfPatients = 0;

actionBottom = 1;

//инициализация и заполнение массива пациентов

patients = ConsoleInput(numberOfPatients);

while (actionBottom != BottomMenu::back) {

OptionsBottom();

actionBottom = GetInt(">>");

switch (actionBottom)

{

case BottomMenu::chooseDiagnosis:

SearchDiagnosis(TopMenu::console, numberOfPatients, patients);

break;

case BottomMenu::chooseRange:

SearchCardNumber(TopMenu::console, numberOfPatients, patients);

break;

case BottomMenu::back:

system("cls");

Fio();

break;

default:

IncorrectOption();

break;

}

}

delete[]patients;

}

//если пользователь выбрал ввод с файла

void WorkWithFile(Patient\* patients, int& actionBottom)

{

ui numberOfPatients = 0;

actionBottom = 1;

fstream docIn, docOut;

do {

string inputName = OpenFile(WorkWithFiles::input, docIn);

docOut.close();

patients = FileInput(docIn, numberOfPatients, inputName);

docIn.close();

} while (patients == nullptr);

while (actionBottom != BottomMenu::back) {

OptionsBottom();

actionBottom = GetInt(">>");

switch (actionBottom)

{

case BottomMenu::chooseDiagnosis:

SearchDiagnosis(TopMenu::file, numberOfPatients, patients);

break;

case BottomMenu::chooseRange:

SearchCardNumber(TopMenu::file, numberOfPatients, patients);

break;

case BottomMenu::back:

system("cls");

Fio();

break;

default:

IncorrectOption();

break;

}

}

delete[]patients;

}

//вывод на консоль данных о разработчике

void Fio()

{

cout << "Control work #1\nEfremov Ivan Andreevich\nGroup #423\nVariant #8\n\n";

}

//вывод на консоль верхнего меню

void OptionsTop()

{

cout << "1 - Console input\n2 - File input\n3 - Run module tests\n4 - Exit\n\n";

}

//вывод на консоль нижнего меню

void OptionsBottom()

{

cout << "\n\n1 - List of patients with chosen diagnosis\n";

cout << "2 - List of patients with medical card number within chosen range\n";

cout << "3 - Back\n\n";

}

//вывод на консоль ФИО пациента под номером

void OutputConsole(Patient a, int patientNum)

{

cout << "#" << patientNum << ": ";

a.ShowFIO();

}

//вывод на консоль выбора пользователя по считыванию данных

void InputOption(int option)

{

if (option == TopMenu::console) {

cout << "\n<<Console Input>>\n\n";

}

else if(option == TopMenu::file){

cout << "\n<<File Input>>\n\n";

}

}

//вывод в файл ФИО пациента под номером

void OutputFile(fstream &fout, string o, int patientNum)

{

fout << "#" << patientNum << ": ";

fout << o << '\n';

}

//если выбран некорректный пункт меню

void IncorrectOption()

{

cout << "There is no such option in menu!\n";

}

//предложить пользователю сохранить данные в файл при вводе с консоли

char SaveResults(fstream &fout)

{

string name;

cout << "Do you want to save results in the file? (y/n)\n\n";

char res = 'n';

do {

res = GetChar(">>");

if (res != 'y' && res != 'n') {

cout << "Incorrect input. Type 'y' or 'n' only!\n\n";

}

} while (res != 'y' && res != 'n');

if (res == 'y') {

name = OpenFile(WorkWithFiles::output, fout);

}

return res;

}

//открытие файла с двумя опциями: 0 - для считывания 1 - для записи

string OpenFile(int option, fstream &file) {

string name = "";

if (option == WorkWithFiles::input) {

do {

name = GetLink("\nEnter the name of file with data. Example: patients.txt\n");

file.open(name, ios::in);

if (!file.is\_open()) {

cout << "\nError opening file. Make sure, that file exists!\n";

}

} while (!file.is\_open());

return name;

}

else {

do {

name = GetLink("\nEnter the name of file where results will be stored.\nIf there is data in the file it will be overwritten.\nExample: filtered.txt\n");

file.open(name, ios::out, ios::trunc);

} while (!file.is\_open());

return name;

}

}

//основная структура программы

void StartProgram()

{

Patient\* patients = nullptr;

int actionTop = 1, actionBottom = 1;

Fio();

while (actionTop) {

OptionsTop();

actionTop = GetInt(">>");

InputOption(actionTop);

switch (actionTop)

{

case TopMenu::console:

WorkWithConsole(patients, actionBottom);

break;

case TopMenu::file:

WorkWithFile(patients, actionBottom);

break;

case TopMenu::module:

LaunchAllTests();

break;

case TopMenu::quit:

cout << "\nProgramm finished it's work!\n";

exit(EXIT\_SUCCESS);

default:

IncorrectOption();

break;

}

}

}

Файл DataInput.h:

#pragma once

#include "ConsoleInterface.h"

void SortData(Patient\* patients, ui numberOfPatients);

void SplitString(string s, string\* ans);

Patient\* ConsoleInput(ui &numberOfPatients);

Patient\* FileInput(fstream& fin, ui& numberOfPatients, string name);

Файл DataInput.cpp:

#include "DataInput.h"

#include <fstream>

#include <algorithm>

const ui arguments = 6;

//сортировка массива объектов по возрастанию номера мед. карты

void SortData(Patient\* patients, ui numberOfPatients) {

sort(patients, patients + numberOfPatients,

[](Patient& x, Patient& y) ->bool {

return x.GetMedicalCardNumber() < y.GetMedicalCardNumber();

}

);

}

//разделение строки по пробелам и заполнение массива полученными значениями

void SplitString(string s, string \*ans)

{

string cur = "";

ui cnt = 0;

for (int i = 0; i < arguments; i++) ans[i] = "";

for (int i = 0; i < s.length(); i++) {

if (s[i] == ' ') {

ans[cnt] = cur;

cur = "";

cnt++;

}

else if (i == s.length() - 1) {

cur += s[i];

ans[cnt] = cur;

}

else cur += s[i];

}

}

//выделение памяти и заполнение массива объектов класса при вводе значений с консоли

Patient\* ConsoleInput(ui &numberOfPatients)

{

string n = "", s = "", p = "", a = "", d = "";

string cn = "";

Patient\* patients = nullptr;

numberOfPatients = static\_cast<ui>(GetInt("\nEnter number of patients: "));

patients = new Patient[numberOfPatients];

for (ui i = 0; i < numberOfPatients; i++) {

cout << "\nPatient " << i + 1 << ":\n";

n = GetString("Enter name of patient #", i + 1);

s = GetString("Enter surname of patient #", i + 1);

p = GetString("Enter patronimyc of patient #", i + 1);

a = GetString("Enter adress of patient #", i + 1);

d = GetString("Enter diagnosis of patient #", i + 1);

cn = "Enter medical card number of patient #" + to\_string(i + 1) + ": ";

ui cardNumber = static\_cast<ui>(GetInt(cn));

//заполнение полей класса для i-того элемента

patients[i].SetInformation(n, s, p, a, d, cardNumber);

}

//сортировка массива

SortData(patients, numberOfPatients);

return patients;

}

//выделение памяти и заполнение массива объектов класса при вводе значений с файла

Patient\* FileInput(fstream &fin, ui &numberOfPatients, string name)

{

ui lineNumber = 0, j = 0, cardNumber=0;

string tmp = "";

char\* trash;

string args[6];

Patient\* patients = nullptr;

//подсчет количества данных о разных пациентах в файле

while (getline(fin, tmp)) {

if (!tmp.empty()) numberOfPatients++;

}

fin.close();

patients = new Patient[numberOfPatients];

fin.open(name);

while(getline(fin, tmp) && j < numberOfPatients) {

lineNumber++;

if (!tmp.empty()) {

SplitString(tmp, args);

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

if (args[i] == "" || args[i] == " ") {

cout << "\nERROR!\nCheck file line #" << lineNumber << "\nIt may be incorrect space input or incorrect amount of values\n";

cout << "Choose another file or edit the current one\n\n";

delete[] patients;

patients = nullptr;

return patients;

}

}

cardNumber = (strtol(args[5].c\_str(), &trash, 10));

//проверка на положительность значения номера мед карты

if (cardNumber <= 0) {

cout << "\nERROR!\nCheck file line #" << lineNumber << "\nCard number must be >0!\n";

cout << "Choose another file or edit the current one\n\n";

delete[] patients;

patients = nullptr;

return patients;

}

//проверка перевелась ли строка полностью к типу unsigned int

if (static\_cast<ui>(cardNumber) != static\_cast<ui>(stoi(args[5]))) {

cout << "\nERROR!\nCheck file line #" << lineNumber << "\nCard number must be integer!\n";

cout << "Choose another file or edit the current one\n\n";

delete[] patients;

patients = nullptr;

return patients;

}

//заполнение полей класса для i-того элемента

patients[j].SetInformation(args[0], args[1], args[2], args[3], args[4], cardNumber);

j++;

}

}

fin.close();

//сортировка массива

SortData(patients, numberOfPatients);

return patients;

}

Файл GetInfo.h:

#pragma once

#include <iostream>

#include <limits>

#include <string>

using namespace std;

int GetInt(string s);

string GetString(string s, int i);

string GetLink(string s);

string GetString(string s);

char GetChar(string s);

Файл GetInfo.cpp:

#include "GetInfo.h"

//очистка потока ввода

void IgnoreLine()

{

cin.ignore(numeric\_limits<std::streamsize>::max(), '\n');

}

//считывание целого числа с консоли

int GetInt(string s) {

cout << s;

int tmp = 0;

while (true) {

cin >> tmp;

if (cin.fail()) {

cin.clear();

IgnoreLine();

cout << "\nUse correct values!\n\n";

cout << s;

}

else if (tmp <= 0) {

cout << "\nValue must be >0\n\n";

cin.clear();

IgnoreLine();

cout << s;

}

else {

IgnoreLine();

return tmp;

}

}

}

//считывание строки с консоли, указывая номер объекта для которого строка считывается

string GetString(string s, int i)

{

string a = "";

while (true) {

cout << s << i << ": ";

cin >> a;

if (getchar() != '\n') {

cout << "\nInput only one word per line!\n";

cout << s;

cin.clear();

IgnoreLine();

}

else if (a.empty()) {

cout << "\nYou must enter a word!\n";

cout << s;

cin.clear();

IgnoreLine();

}

else return a;

}

}

//считывание всей строки для указания адреса файла

string GetLink(string s)

{

string b = "";

cout << s;

cout << "Path must not contain cyrillic symbols!\n\n>>";

getline(cin, b);

return b;

}

//считывание строки без нумерации

string GetString(string s)

{

string b = "";

while (true) {

cout << s;

cin >> b;

if (getchar() != '\n') {

cout << "\nInput only one word per line!\n";

cout << s;

cin.clear();

IgnoreLine();

}

else if (b.empty()) {

cout << "\nYou must enter a word!\n";

cout << s;

cin.clear();

IgnoreLine();

}

else return b;

}

return b;

}

//считывание ответа y/n

char GetChar(string s)

{

char tmp = '\n';

cout << s;

while (true) {

tmp = getchar();

if (tmp == '\n') {

cout << "You must type y or n!\n\n";

cout << s;

cin.clear();

IgnoreLine();

}

else if (getchar() != '\n') {

cout << "Incorrect input! Type only y or n!\n\n";

cout << s;

cin.clear();

IgnoreLine();

}

else return tmp;

}

}

Файл Tasks.h:

#pragma once

#include "DataInput.h"

bool Comparison(string diagnosis, string currentDiagnosis);

void SearchDiagnosis(int mode, ui numberOfPatients, Patient\* patients);

void OutputDiagnosis(int mode, pair<Patient, bool>\* filteredPatients, ui numberOfPatients, string chosenDiagnosis);

void SearchCardNumber(int mode, ui numberOfPatients, Patient\* patients);

void OutputCardNumber(int mode, pair<Patient, bool>\* filteredPatients, ui numberOfPatients, pair<ui, ui> ranges);

Файл Tasks.cpp:

#include "Tasks.h"

//сравнение строк без учета регистра

bool Comparison(string diagnosis, string currentDiagnosis)

{

if (diagnosis == currentDiagnosis) return true;

else {

if (diagnosis.length() == currentDiagnosis.length()) {

for (int i = 0; i < diagnosis.length(); i++) {

if (tolower(diagnosis[i]) != tolower(currentDiagnosis[i])) {

return false;

}

}

return true;

}

else return false;

}

}

//функция для поиска совпадений поля "диагноз" и записи ФИО таких людей в файл

void SearchDiagnosis(int mode, ui numberOfPatients, Patient\* patients)

{

string chosenDiagnosis = GetString("\nEnter the diagnosis you want to see patients with: ");

pair<Patient, bool> \*results = new pair<Patient, bool>[numberOfPatients];

for (ui i = 0; i < numberOfPatients; i++) {

if (Comparison(patients[i].GetDiagnosis(), chosenDiagnosis)) {

results[i].second = true;

}

else results[i].second = false;

results[i].first = patients[i];

}

OutputDiagnosis(mode, results, numberOfPatients, chosenDiagnosis);

}

//вывод пациентов с данным диагнозом в консоль или в файл

void OutputDiagnosis(int mode, pair<Patient, bool>\* filteredPatients, ui numberOfPatients, string chosenDiagnosis)

{

int cnt = 1;

fstream fout;

if (mode == TopMenu::console) {

cout << "\nPatients with diagnosis of " << chosenDiagnosis << ":\n\n";

for (ui i = 0; i < numberOfPatients; i++) {

if (filteredPatients[i].second == true) {

OutputConsole(filteredPatients[i].first, cnt);

cnt++;

}

}

cnt = 1;

//предложить пользователю записать результат работы функции в файл

char ans = SaveResults(fout);

if (ans == 'y') {

fout << "Patients with diagnosis of " << chosenDiagnosis << ":\n\n";

for (ui i = 0; i < numberOfPatients; i++) {

if (filteredPatients[i].second == true) {

OutputFile(fout, filteredPatients[i].first.GetFIO(), cnt);

cnt++;

}

}

}

fout.close();

}

else {

string name = OpenFile(WorkWithFiles::output, fout);

fout << "Patients with diagnosis of " << chosenDiagnosis << ":\n\n";

for (ui i = 0; i < numberOfPatients; i++) {

if (filteredPatients[i].second == true) {

OutputFile(fout, filteredPatients[i].first.GetFIO(), cnt);

cnt++;

}

}

fout.close();

}

}

//функция для поиска людей с номером мед. карты в определенном интервале и запись ФИО этих людей в файл

void SearchCardNumber(int mode, ui numberOfPatients, Patient\* patients)

{

pair<ui, ui> ranges = { 0, 1 };

pair<Patient, bool>\* results = new pair<Patient, bool>[numberOfPatients];

//ввод и проверка корректности значений границ интервала

do {

ranges.first = static\_cast<ui>(GetInt("\nEnter lower range of medical card numbers: "));

ranges.second = static\_cast<ui>(GetInt("Enter upper range of medical card numbers: "));

if (ranges.first > ranges.second) {

cout << "\nLower range must be less or equal to the upper range!\n";

continue;

}

else if (ranges.first < 0 || ranges.second < 0) {

cout << "\nBorders must be positive!\n";

continue;

}

break;

} while (ranges.first > ranges.second);

cout << "\n";

//fout << "\nPatients with medical card number in the range of [" << lowerRange << ":" << upperRange << "]:\n\n";

for (ui i = 0; i < numberOfPatients; i++) {

if (patients[i].GetMedicalCardNumber() >= ranges.first && patients[i].GetMedicalCardNumber() <= ranges.second) {

results[i].second = true;

}

else results[i].second = false;

results[i].first = patients[i];

}

OutputCardNumber(mode, results, numberOfPatients, ranges);

}

//вывод пациентов с номером карты в заданном отрезке в консоль или в файл

void OutputCardNumber(int mode, pair<Patient, bool>\* filteredPatients, ui numberOfPatients, pair<ui, ui> ranges)

{

int cnt = 1;

fstream fout;

if (mode == TopMenu::console) {

cout << "Patients with medical card number in the range of [" << ranges.first << ":" << ranges.second << "]:\n\n";

for (ui i = 0; i < numberOfPatients; i++) {

if (filteredPatients[i].second == true) {

OutputConsole(filteredPatients[i].first, cnt);

cnt++;

}

}

cnt = 1;

//предложить пользователю записать результат работы функции в файл

char ans = SaveResults(fout);

if (ans == 'y') {

fout << "Patients with medical card number in the range of [" << ranges.first << ":" << ranges.second << "]:\n\n";

for (ui i = 0; i < numberOfPatients; i++) {

if (filteredPatients[i].second == true) {

OutputFile(fout, filteredPatients[i].first.GetFIO(), cnt);

cnt++;

}

}

}

fout.close();

}

else {

string name = OpenFile(WorkWithFiles::output, fout);

fout << "Patients with medical card number in the range of [" << ranges.first << ":" << ranges.second << "]:\n\n";

for (ui i = 0; i < numberOfPatients; i++) {

if (filteredPatients[i].second == true) {

OutputFile(fout, filteredPatients[i].first.GetFIO(), cnt);

cnt++;

}

}

fout.close();

}

}

Файл Patient.h:

#pragma once

#include "GetInfo.h"

using namespace std;

#define ui unsigned int

class Patient {

private:

string name;

string surname;

string patronimyc;

string adress;

string diagnosis;

ui numberOfMedicalCard;

public:

Patient();

void SetInformation(string n, string s, string p, string a, string d, ui number);

string GetDiagnosis();

ui GetMedicalCardNumber();

string GetFIO();

string GetName();

string GetSurname();

string GetPatronimyc();

string GetAdress();

void ShowFIO();

};

Файл Patient.cpp:

#include "Patient.h"

//конструктор по умолчанию

Patient::Patient() {

name = "";

surname = "";

patronimyc = "";

adress = "";

diagnosis = "";

numberOfMedicalCard = 0;

}

//заполнение всех полей объекта

void Patient::SetInformation(string n, string s, string p, string a, string d, ui number) {

name = n;

surname = s;

patronimyc = p;

adress = a;

diagnosis = d;

numberOfMedicalCard = number;

}

//получение значения поля "диагноз"

string Patient::GetDiagnosis() {

return diagnosis;

}

//получение значения поля "номер мед. карты"

ui Patient::GetMedicalCardNumber() {

return numberOfMedicalCard;

}

//получение строки из полей "имя", "фамилия" и "отчество"

string Patient::GetFIO() {

return name + " " + surname + " " + patronimyc;

}

//получение значения поля "имя"

string Patient::GetName()

{

return name;

}

//получение значения поля "фамилия"

string Patient::GetSurname()

{

return surname;

}

//получение значения поля "отчество"

string Patient::GetPatronimyc()

{

return patronimyc;

}

//получение значения поля "адрес"

string Patient::GetAdress()

{

return adress;

}

//вывод в консоль строки из полей "имя", "фамилия" и "отчество"

void Patient::ShowFIO()

{

cout << GetFIO() << endl;

}

Файл ModuleTests.h:

#pragma once

#include "Patient.h"

#include "Tasks.h"

enum ValuesForModuleTests{first = 2, second = 2, third = 3, fourth = 7};

bool Check(Patient\* testArray, Patient\* testArrayFile, ui lim);

bool Test1();

bool Test2();

bool Test3();

bool Test4();

bool Test5();

void LaunchAllTests();

Файл ModuleTests.cpp:

#include "ModuleTests.h"

#include "DataInput.h"

#include <sstream>

bool Check(Patient \*testArray,Patient \*testArrayFile, ui lim)

{

for (ui i = 0; i < lim; i++) {

if (testArray[i].GetName() != testArrayFile[i].GetName() ||

testArray[i].GetSurname() != testArrayFile[i].GetSurname() ||

testArray[i].GetPatronimyc() != testArrayFile[i].GetPatronimyc() ||

testArray[i].GetAdress() != testArrayFile[i].GetAdress() ||

testArray[i].GetDiagnosis() != testArrayFile[i].GetDiagnosis() ||

testArray[i].GetMedicalCardNumber() != testArrayFile[i].GetMedicalCardNumber()

) return false;

}

return true;

}

bool Test1()

{

//проверка ввода данных с файла

cout << "\tTEST#1\n\tTesting file input...\n\n";

fstream fin;

const string fileName = "test\_info.txt";

fin.open(fileName, ios::in);

if (!fin) {

cout << "FAILURE!\nError opening file!\n\n";

return false;

}

ui numberOfPatients = 0;

Patient testArray[ValuesForModuleTests::first];

testArray[0].SetInformation("Ilya", "Ekkert", "Denisovich", "Berlin", "Tourette\_Syndrome", 1);

testArray[1].SetInformation("Alina", "Galkina", "Aleksandrovna", "Pervomayka", "Koshmar", 2);

Patient\* testArrayFile = FileInput(fin, numberOfPatients, fileName);

if (!Check(testArray, testArrayFile, ValuesForModuleTests::first)) {

cout << "FAILURE!\nValues from inside the program and values from the test file do not match!\n\n";

return false;

}

return true;

}

bool Test2()

{

//фильтр по диагнозу

cout << "\tTEST#2\n\tTesting diagnosis filter function...\n\n";

fstream fin;

const string fileName = "test\_info.txt";

fin.open(fileName, ios::in);

string chosenDiagnosis = "koshmar";

string res = "", resFile="";

if (!fin) {

cout << "FAILURE!\nError opening file!\n\n";

return false;

}

ui numberOfPatients = 0;

Patient testArray[ValuesForModuleTests::second];

testArray[0].SetInformation("Ilya", "Ekkert", "Denisovich", "Berlin", "Tourette\_Syndrome", 1);

testArray[1].SetInformation("Alina", "Galkina", "Aleksandrovna", "Pervomayka", "Koshmar", 2);

Patient\* testArrayFile = FileInput(fin, numberOfPatients, fileName);

for (ui i = 0; i < ValuesForModuleTests::second; i++) {

if (Comparison(testArray[i].GetDiagnosis(), chosenDiagnosis)) res = testArray[i].GetFIO();

}

for (ui i = 0; i < numberOfPatients; i++) {

if (Comparison(testArrayFile[i].GetDiagnosis(), chosenDiagnosis)) resFile = testArrayFile[i].GetFIO();

}

if (res != resFile) {

cout << "FAILURE!\nIncorrect answer\n\n";

return false;

}

return true;

}

bool Test3()

{

//фильтр по диапазону

cout << "\tTEST#3\n\tTesting range filter function...\n\n";

fstream fin;

const string fileName = "test\_info.txt";

fin.open(fileName, ios::in);

string chosenDiagnosis = "koshmar";

string res = "", resFile = "";

if (!fin) {

cout << "FAILURE!\nError opening file!\n\n";

return false;

}

ui numberOfPatients = 0;

Patient testArray[ValuesForModuleTests::third];

testArray[0].SetInformation("Ilya", "Ekkert", "Denisovich", "Berlin", "Tourette\_Syndrome", 1);

testArray[1].SetInformation("Alina", "Galkina", "Aleksandrovna", "Pervomayka", "Koshmar", 2);

testArray[2].SetInformation("Anastasia", "Runina", "Sergeevna", "Astrakhan", "Children", 3);

Patient\* testArrayFile = FileInput(fin, numberOfPatients, fileName);

ui lowerRange = 3, upperRange = 4;

for (ui i = 0; i < ValuesForModuleTests::third; i++) {

if (testArray[i].GetMedicalCardNumber() >= lowerRange && testArray[i].GetMedicalCardNumber() <= upperRange) res = testArray[i].GetFIO();

}

for (ui i = 0; i < numberOfPatients; i++) {

if (testArrayFile[i].GetMedicalCardNumber() >= lowerRange && testArrayFile[i].GetMedicalCardNumber() <= upperRange) resFile = testArrayFile[i].GetFIO();

}

if (res != resFile) {

cout << "FAILURE!\nIncorrect answer\n\n";

return false;

}

return true;

}

bool Test4()

{

//корректность счёта строк в файле

cout << "\tTEST#4\n\tTesting correctness of counting lines in a file...\n\n";

fstream fin;

const string fileName = "test\_info.txt";

fin.open(fileName, ios::in);

if (!fin) {

cout << "FAILURE!\nError opening file!\n\n";

return false;

}

ui numberOfPatients = 0;

Patient\* testArrayFile = FileInput(fin, numberOfPatients, fileName);

if (numberOfPatients != ValuesForModuleTests::fourth) {

cout << "FAILURE!\nIncorrect amount of data in the file!\n\n";

return false;

}

return true;

}

bool Test5()

{

//корректность работы ввода с консоли

cout << "\tTEST#5\n\tTesting console input...\n\n";

{

istringstream test("576\n\n");

streambuf\* cinbuf = cin.rdbuf(test.rdbuf());

bool res = (GetInt("") == 576);

if (!res) {

cout << "FAILURE!\nIncorrect console input!\n\n";

return false;

}

cin.rdbuf(cinbuf);

}

return true;

}

void LaunchAllTests()

{

bool tests[5] = { Test1(), Test2(), Test3(), Test4(), Test5() };

//bool tests[5] = { Test1(), Test2(), Test3(), Test4(), true };

for (ui i = 0; i < 5; i++) {

if(!tests[i]) cout << "TEST#" << i + 1 << "\tFAILURE!\n\n";

}

if (tests[0] && tests[1] && tests[2] && tests[3] && tests[4]) cout << "All tests SUCCEDED!\n\n";

else cout << "Some of the tests failed\n\n";

}