

#include <iostream>

#include <fstream>

#include <string>

#include <Windows.h>

using namespace std;

struct BankClient {

std::string fullName; // Прозвішча, імя кліента банка

std::string accountType; // Тып акаўнта

int accountNumber = 0; // Нумар рахунку

long long accountBalance = 0; // Баланс

std::string lastModifiedDate; // Апошняя дата змены інфармацыі

};

const int MAX\_CLIENTS = 15;

BankClient clients[MAX\_CLIENTS];

int currentClientsCount = 0;

// Увод звестак пра кліента

void inputClientsData();

// Вывад звестак пра ўсіх кліентаў

void displayClientsData();

// Выдаленне звестак

void deleteClient(int accountNumberToDelete);

// Пошук кліента

void searchClient();

// Запіс інфармацыі ў файл

void writeToFile();

// Чытанне інфармацыі з файла

void readFromFile();

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int choice;

do {

cout << "\nСістэма кіравання кліентамі банка\n";

cout << "1. Увод дадзеных кліентаў\n";

cout << "2. Адлюстраваць дадзеныя кліентаў\n";

cout << "3. Выдаліць кліента\n";

cout << "4. Пошук кліента\n";

cout << "5. Запіс у файл\n";

cout << "6. Чытанне з файла\n";

cout << "0. Выхад\n";

cout << "Увядзіце свой выбар: ";

cin >> choice;

switch (choice) {

case 1:

inputClientsData();

break;

case 2:

displayClientsData();

break;

case 3: {

int accountNumberToDelete;

cout << "Увядзеце нумар рахунку кліента для выдалення: ";

cin >> accountNumberToDelete;

deleteClient(accountNumberToDelete);

break;

}

case 4:

searchClient();

break;

case 5:

writeToFile();

break;

case 6:

readFromFile();

break;

case 0:

cout << "Выхад з праграмы....\n";

break;

default:

cout << "Некарэктны выбар\n";

}

} while (choice != 0);

return 0;

}

void inputClientsData() {

if (currentClientsCount >= MAX\_CLIENTS) {

std::cerr << "Максімальная колькасць кліентаў ужо дасягнута\n";

return;

}

BankClient newClient; // Ствараем новы аб'ект структуры

cin.ignore(); // Выдяляем сімвал новага радку з буфера, каб getline не атрымліваў пусты радок замест уводу карыстальніка

// Атрымліваем ад карыстальніка ўсе звесткі і захоўваем

cout << "Увядзіце Прозвішча, імя: ";

std::getline(cin, newClient.fullName);

cout << "Увядзеце тып акаўнта: ";

std::getline(cin, newClient.accountType);

cout << "Увядзеце нумар рахунку: ";

cin >> newClient.accountNumber;

cout << "Увядзеце баланс: ";

cin >> newClient.accountBalance;

cout << "Увядзеце дату апошняй змены: ";

cin.ignore();

std::getline(cin, newClient.lastModifiedDate);

// Павялічваем колькасць кліентаў пры выкарыстанні зменнай у яасці індэксу пры захаванні звестак пра кліента

clients[currentClientsCount++] = newClient;

}

void readFromFile() {

std::ifstream inFile("bank\_clients.txt");

if (!inFile) {

std::cerr << "Памылка адкрыцця файла для чытання.\n";

return;

}

while (currentClientsCount < MAX\_CLIENTS) {

BankClient newClient;

// Для ўвода выконваем праверку, ці былі прачытаны дадзеныя з файла (для апошняга радка ўсе сімвалы скончацца, але без праверкі будзе створаны дапаўняльны аб'ект newClient з пустымі палямі)

if (!std::getline(inFile, newClient.fullName, ',')) break;

std::getline(inFile, newClient.accountType, ',');

(inFile >> newClient.accountNumber);

inFile.ignore(); // ігнаруем сімвал ',' у файле

(inFile >> newClient.accountBalance);

inFile.ignore(); // ігнаруем сімвал ',' у файле

std::getline(inFile, newClient.lastModifiedDate);

clients[currentClientsCount++] = newClient;

}

cout << "Даныя паспяхова прачытаны з файла.\n";

inFile.close();

}

void writeToFile() {

// Открытие файла для записи

std::ofstream outFile("bank\_clients.txt");

// Проверка успешности открытия файла

if (!outFile) {

std::cerr << "Памылка адкрыцця файла для запісу.\n";

return;

}

// Запись данных каждого клиента в файл, разделяя поля запятой

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

outFile << clients[i].fullName << ","

<< clients[i].accountType << ","

<< clients[i].accountNumber << ","

<< clients[i].accountBalance << ","

<< clients[i].lastModifiedDate << "\n";

}

// Вывод сообщения об успешной записи данных в файл

cout << "Даныя паспяхова запісаны ў файл.\n";

// Закрытие файла после завершения записи

outFile.close();

}

void searchClient() {

int accountNumberToSearch;

cout << "Вядзіце нумар рахунку для пошуку: ";

cin >> accountNumberToSearch;

// Праход па масіве кліентаў

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

// Праверка, ці супадае нумар акаўнта з шуканым

if (clients[i].accountNumber == accountNumberToSearch) {

// Выснова інфармацыі аб знойдзеным кліенце

cout << "Знойдзены кліент:\n";

cout << "ПІБ: " << clients[i].fullName << "\n";

cout << "Тып рахунку: " << clients[i].accountType << "\n";

cout << "Нумар рахунку: " << clients[i].accountNumber << "\n";

cout << "Баланс: " << clients[i].accountBalance << "\n";

cout << "Дата апошняй змены: " << clients[i].lastModifiedDate << "\n\n";

return; // Кліент знойдзены, завяршаем функцыю

}

}

// Калі цыкл завяршыўся без вяртання, кліент не знойдзены

cout << "Кліент з нумарам рахунку " << accountNumberToSearch << " не знойдзены.\n";

}

void displayClientsData() {

// Праход па масіве кліентаў для вываду інфармацыі аб кожным

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

// Вывад інфармацыі аб кліенце

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

cout << "ПІБ: " << clients[i].fullName << "\n";

cout << "Тып рахунку: " << clients[i].accountType << "\n";

cout << "Нумар рахунку: " << clients[i].accountNumber << "\n";

cout << "Баланс: " << clients[i].accountBalance << "\n";

cout << "Дата апошняй змены: " << clients[i].lastModifiedDate << "\n\n";

}

}

void deleteClient(int accountNumberToDelete) {

int foundIndex = -1;

// Знаходзім індэкс кліента з дадзеным нумарам рахунку, бо нумар рахунку не роўны нумару кліента ў базе дадзеных (ну мусіць, мне здаецца так)

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

if (clients[i].accountNumber == accountNumberToDelete) {

foundIndex = i;

break;

}

}

if (foundIndex != -1) {

// Зрушваем усе элементы пасля знойдзенага кліента налева на 1 пазіцыю

for (int i = foundIndex; i < currentClientsCount - 1; i++) {

clients[i] = clients[i + 1];

}

// Памяншаем лічыльнік колькасці кліентаў

--currentClientsCount;

cout << "Кліент з нумарам рахунку " << accountNumberToDelete << " выдалены.\n";

}

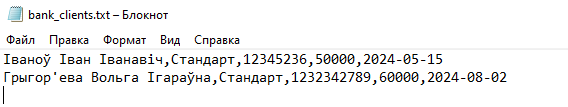
else {

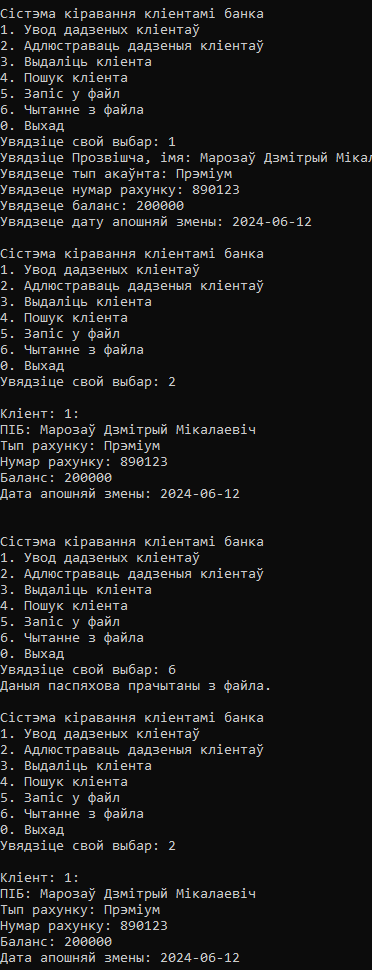
cout << "Кліент з нумарам рахунку " << accountNumberToDelete << " не знойдзены.\n";

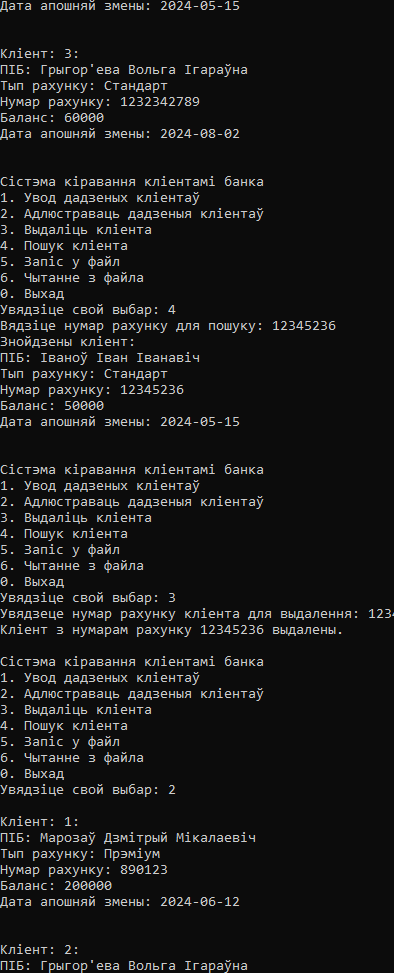
}

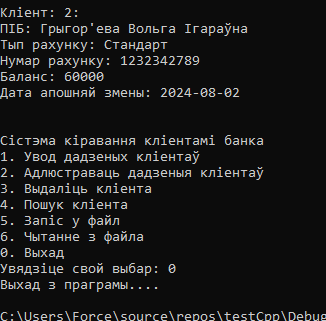
}

Перад запускам

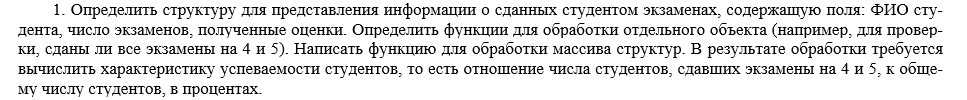








Дап. Заданні



#include <iostream>

#include "Windows.h"

using namespace std;

// Структура для прадстаўлення інфармацыі аб здадзеных іспытах студэнта

struct StudentInfo {

std::string fullName; // Прозвішча, імя студэнта

int examsCount; // Колькасць экзаменаў

int\* grades; // Атрыманыя адзнакі

// Канструктар калі пры ініцыялізацыі указана толькі колькасць экзаменаў, або нічога

StudentInfo(int exams = 0) : examsCount(exams), grades(nullptr) {

if (examsCount > 0) {

grades = new int[examsCount] {};

}

}

// Канструктар для ініцыялізацыі ПІБ і ліку экзаменаў

StudentInfo(const char\* name, int exams) : fullName(name), examsCount(exams), grades(nullptr) {

if (examsCount > 0) {

grades = new int[examsCount] {};

}

}

// Дэструктар для вызвалення памяці

~StudentInfo() {

delete[] grades;

}

};

// Функцыя для праверкі, ці здадзены ўсе іспыты на 4 і 5

bool passedAllExams(const StudentInfo& student) {

for (int i = 0; i < student.examsCount; ++i) {

if (student.grades[i] < 4) {

return false; // Калі хаця б адзін іспыт не здадзены на 4 ці 5, вяртаем false

}

}

return true; // Усе іспыты здадзены на 4 і 5

}

// Функцыя для апрацоўкі масіва структур

double calculateSuccessRate(const StudentInfo\* students, int studentsCount) {

int passedCount = 0; // Лічыльнік студэнтаў, якія здалі іспыты на 4 і 5

// Перабор усіх студэнтаў

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

if (passedAllExams(students[i])) {

passedCount++; // Павялічваем лічыльнік, калі студэнт здаў усе іспыты на 4 і 5

}

}

// Вылічэнне характарыстыкі паспяховасці ў працэнтах

float successRate = (static\_cast<float>(passedCount) / studentsCount) \* 100;

return successRate;

}

int main() {

SetConsoleOutputCP(1251);

// Прыклад выкарыстання

const int studentsCount = 3;

// Стварэнне масіву структуры з інфармацыяй аб студэнтах

// Пры стварэнні ініцыялізуем дадзеныя аб студэнтах (ПІБ, колькасць экзаменаў)

StudentInfo students[studentsCount] = {

{"Ivanov Ivan Ivanovich", 3},

{"Petrov Petrovich", 3},

{"Sidorov Sidor Sidorovich", 3}

};

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

students[i].grades[0] = i + 3; // запаўняю адзнакі так

students[i].grades[1] = i + 2;

students[i].grades[2] = i + 3;

}

//Вывад дадзеных аб студэнтах

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

cout << "Студэнт " << i + 1 << ":\n";

cout << "ПІБ: " << students[i].fullName << "\n";

cout << "Колькасць іспытаў: " << students[i].examsCount << "\n";

cout << "Адзнакі: ";

for (int j = 0; j < students[i].examsCount; ++j) {

cout << students[i].grades[j] << " ";

}

cout << "\n\n";

}

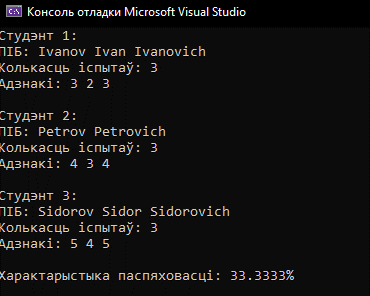
// Вылічэнне і вывад характарыстыкі паспяховасці

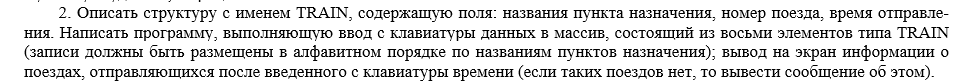
double successRate = calculateSuccessRate(students, studentsCount);

cout << "Характарыстыка паспяховасці: " << successRate << "%\n";

return 0;

}





#include <iostream>

#include "Windows.h"

// Структура для захоўвання інфармацыі аб цягніках

struct TRAIN {

std::string destination; // Назва пункта прызначэння

int trainNumber; // Нумар цягніка

std::string departureTime; // Час адпраўлення

};

// Функцыя для ўводу дадзеных аб цягніках

void inputTrainsData(TRAIN trains[], int size) {

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

std::cout << "Увядзіце інфармацыю аб цягніку #" << i + 1 << ":\n";

std::cout << "Назва пункта прызначэння: ";

std::cin >> trains[i].destination;

std::cout << "Нумар цягніка: ";

std::cin >> trains[i].trainNumber;

std::cout << "Час адпраўлення (гг:хх): ";

std::cin >> trains[i].departureTime;

std::cout << '\n';

}

// Простае сартаванне бурбалкай

// Знешні цыкл праходзіць па ўсіх элементах масіва

for (int i = 0; i < size - 1; ++i) {

// Унутраны цыкл праходзіць ад пачатку масіва да size-i-1

// -i таму што пасля кожнай ітэрацыі знешняга цыклу самы вялікі элемент ужо "усплывае" на сваё месца(у канец масіва) і з ім параўноўваць ня трэба

for (int j = 0; j < size - i - 1; ++j) {

// Параўноўваем бягучы элемент з наступным

if (trains[j].destination > trains[j + 1].destination) {

// Калі бягучы элемент больш наступнага, мяняем іх месцамі

// Гэта ажыццяўляецца шляхам выкарыстання часовай зменнай temp

TRAIN temp = trains[j];

trains[j] = trains[j + 1];

trains[j + 1] = temp;

}

}

}

}

// Функцыя для вываду інфармацыі аб цягніках, якія адпраўляюцца пасля пазначанага часу

void printTrainsAfterTime(const TRAIN trains[], int size, const std::string& time) {

bool found = false;

std::cout << "\nЦягнікі, якія адпраўляюцца пасля часу " << time << ":\n";

// Параўнанне радкоў адбываецца лексікаграфічна, гэта значыць сімвал за сімвалам

// Калі першыя сімвалы супадаюць, праграма пераходзіць да наступнага сімвала і параўноўвае іх.Гэты працэс працягваецца,

// пакуль не будзе знойдзены першы які адрозніваецца знак, і вынік параўнання вызначаецца значэннямі кодаў ASCII або Unicode гэтых знакаў.

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

if (trains[i].departureTime > time) {

std::cout << "Назва: " << trains[i].destination << "\n";

std::cout << "Нумар: " << trains[i].trainNumber << "\n";

std::cout << "Час адпраўлення: " << trains[i].departureTime << "\n\n";

found = true;

}

}

if (!found) {

std::cout << "Цягнікоў, якія адпраўляюцца пасля пазначанага часу, не знойдзена.\n";

}

}

int main() {

SetConsoleOutputCP(1251);

const int arraySize = 4;

TRAIN trains[arraySize];

// Увод дадзеных аб цягніках і сартаванне па назвах пунктаў прызначэння

inputTrainsData(trains, arraySize);

// Выснова інфармацыі аб цягніках, якія адпраўляюцца пасля пазначанага часу

std::string specifiedTime;

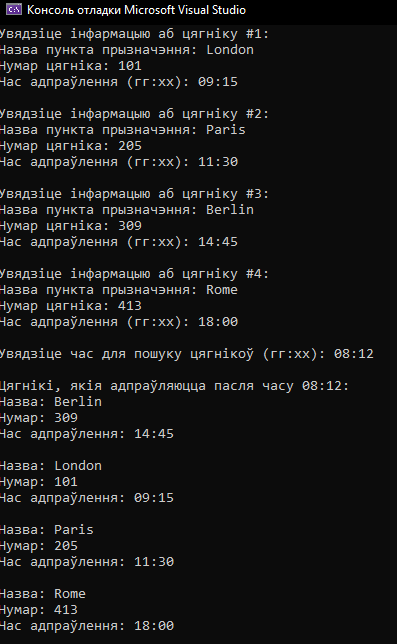
std::cout << "Увядзіце час для пошуку цягнікоў (гг:хх): ";

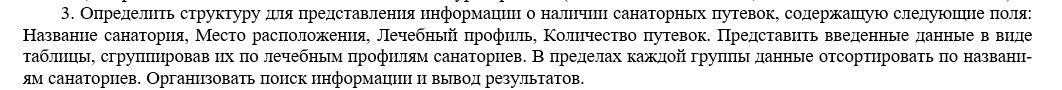
std::cin >> specifiedTime;

printTrainsAfterTime(trains, arraySize, specifiedTime);

return 0;

}





#include <iostream>

#include <vector>

#include <iomanip>

#include "Windows.h"

#include <string>

// Структура для прадстаўлення інфармацыі аб санаторных пуцёўках

struct Sanatorium {

std::string name; // Назва санаторыя

std::string location; // Месца размяшчэння

std::string profile; // Лячэбны профіль

int availableTickets; // Колькасць даступных пуцёвак

// Канструктар для зручнай ініцыялізацыі

Sanatorium(const std::string& n, const std::string& loc, const std::string& prof, int tickets)

: name(n), location(loc), profile(prof), availableTickets(tickets) {}

};

// Функцыя параўнання для сартавання спачатку па профілях, потым па назве

bool compareSanatoriums(const Sanatorium& a, const Sanatorium& b) {

// Спачатку параўноўваем профілі

if (a.profile != b.profile) {

return a.profile < b.profile;

}

// Потым параўноўваем назвы ўнутры аднолькавых профіляў

return a.name < b.name;

}

// Функцыя для вываду дадзеных у выглядзе табліцы

void printSanatoriumTable(std::vector<Sanatorium>& sanatoriums) {

// Сартаванне па лячэбных профілях і потым па назвах санаторыяў

for (int i = 0; i < sanatoriums.size() - 1; ++i) { // size() для атрымання колькасці элементаў у вектары

for (int j = 0; j < sanatoriums.size() - i - 1; ++j) {

// Выкарыстоўваем функцыю параўнання

if (!compareSanatoriums(sanatoriums[j], sanatoriums[j + 1])) {

// Калі парадак не выкананы, мяняем элементы месцамі

Sanatorium temp = sanatoriums[j];

sanatoriums[j] = sanatoriums[j + 1];

sanatoriums[j + 1] = temp;

}

}

}

// Вывад загалоўка табліцы. setw усталёўвае вырыну поля вываду

std::cout << std::setw(20) << "Назва" << std::setw(20) << "Месца" << std::setw(20) << "Профіль" << std::setw(30) << "Колькасць пуцёвак" << "\n\n";

// Вывад дадзеных

for (const auto& sanatorium : sanatoriums) { // З дапамогай ітэратара перабіраем усе элементы вектара і выводзім інфармацыю для кожнага санаторыя ў вектары

std::cout << std::setw(20) << sanatorium.name << std::setw(20) << sanatorium.location

<< std::setw(20) << sanatorium.profile << std::setw(20) << sanatorium.availableTickets << std::endl;

}

}

void search(const std::vector<Sanatorium>& sanatoriums) {

std::cout << "Па якім крытэрыі трэба знайсці санаторый?\n";

std::cout << "1. Назва\n";

std::cout << "2. Размяшчэнне\n";

std::cout << "3. Лячэбны профіль\n";

int choice;

std::cin >> choice;

switch (choice)

{

case 1: {

std::cout << "Увядзіце назву для пошуку (рэгістразалежна): ";

std::string name; // Радок для назвы

std::cin.ignore(); // Ігнаруем сімвал новага радка, які застаўся пасля ўводу выбару

std::getline(std::cin, name); // Атрмліваем радок

for (const auto& sanatorium : sanatoriums) {

if (sanatorium.name == name) { // Калі гэта тое, што шукаў карыстальнік

std::cout << std::setw(20) << sanatorium.name << std::setw(20) << sanatorium.location

<< std::setw(20) << sanatorium.profile << std::setw(20) << sanatorium.availableTickets << std::endl;

}

}

break;

}

case 2: {

std::cout << "Увядзіце размяшчэнне для пошуку (рэгістразалежна): ";

std::string loc;

std::cin.ignore();

std::getline(std::cin, loc);

for (const auto& sanatorium : sanatoriums) {

if (sanatorium.location == loc) {

std::cout << std::setw(20) << sanatorium.name << std::setw(20) << sanatorium.location

<< std::setw(20) << sanatorium.profile << std::setw(20) << sanatorium.availableTickets << std::endl;

}

}

break;

}

case 3: {

std::cout << "Увядзіце профіль для пошуку (рэгістразалежна): ";

std::string prof;

std::cin.ignore();

std::getline(std::cin, prof);

for (const auto& sanatorium : sanatoriums) {

if (sanatorium.profile == prof) {

std::cout << std::setw(20) << sanatorium.name << std::setw(20) << sanatorium.location

<< std::setw(20) << sanatorium.profile << std::setw(20) << sanatorium.availableTickets << std::endl;

}

}

break;

}

default:

break;

}

}

int main() {

SetConsoleOutputCP(1251);

std::vector<Sanatorium> sanatoriums = { // Вектар санаторыяў (выдуманых)

{"Zdravie", "Grodno", "Neurology", 20},

{"Belmed", "Minsk", "Cardiology", 50},

{"Zubr", "Mogilev", "Orthopedics", 30},

{"Solnechny", "Mogilev", "Dermatology", 30},

{"Raduga", "Brest", "Dermatology", 25},

{"BelarusHealth", "Gomel", "Respiratory", 45},

{"Priozerny", "Gomel", "Respiratory", 40},

{"Nadezda", "Vitebsk", "Neurology", 15},

{"Zeleny Bor", "Vitebsk", "Neurology", 35},

{"SpaRelax", "Brest", "Orthopedics", 20},

{"MedicalCare", "Minsk", "Cardiology", 35},

{"Luch", "Grodno", "Orthopedics", 25},

{"Rehabilitation", "Gomel", "Dermatology", 20},

{"HealthResort", "Minsk", "Respiratory", 30},

{"WellnessCenter", "Brest", "Cardiology", 15}

};

// Вывад табліцы

printSanatoriumTable(sanatoriums);

// Пошук

search(sanatoriums);

return 0;

}

