**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИТЕТ**

**им. И.Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра «**Программное обеспечение компьютерных систем**»

Направление: 710400 «**Программная инженерия**»

Дисциплина: ««**Объектно-Ориентированное Программирование**»»

ОТЧЕТ

**Самостоятельная работа**

Тема: “Объектно-ориентированное программирование С++”

Выполнил: Кудайбердиев Эрлан

Группа: ПИ-2-21

Проверил: Мусабаев Э. Б.

**Бишкек – 2024**

Оглавление

[Задача 17.6 3](#_Toc166277314)

[Описание задачи 3](#_Toc166277315)

[Диаграмма use case 4](#_Toc166277316)

[Листинг кода 4](#_Toc166277317)

[Результат программы 8](#_Toc166277318)

[Задача 17.30 11](#_Toc166277319)

[Описание задачи 11](#_Toc166277320)

[Диаграмма use case 12](#_Toc166277321)

[Листинг кода 12](#_Toc166277322)

[Результат программы 16](#_Toc166277323)

[Заключение 18](#_Toc166277324)

# Задача 17.6

Записная книжка. Анкетные данные, адреса, телефоны, место работы или учебы, должность знакомых, коллег и родственников, характер знакомства, деловые качества и так далее. Автоматическое формирование поздравления с днем рождения (по текущей дате). Упорядочение по алфавиту и по дате последней корректировки. Поиск по произвольному шаблону.

## Описание задачи

**Структура Contact**: Эта структура предназначена для хранения информации о контактах. Она содержит следующие поля и методы:

**Поля:**

string name; *//имя*

string address; *//адрес*

string phone; *//номер телефона*

string workplace; *//место работы*

string position; *//должность*

string relation; *//отношения*

string notes; *//примечание*

time\_t lastModified; *//последняя модификация*

**Методы:**

Конструктор по умолчанию: Contact()

Конструктор: Contact(string n, string addr, string ph, string wp, string pos, string rel, string nt)

**Класс** **AddressBook:** Этот класс представляет собой записную книгу. Который содержит следующие поля и методы:

**Поля:**

Contact contacts[MAX\_CONTACTS]; *//приватный массив хранящий список контактов*

int numContacts; *//количество контактов*

**Методы:**

addContact() *//метод для добавление контактов*

sortByAlphabet() *//метод для сортировки контактов по алфавиту*

sortByLastModified() *//метод для сортировки контактов по последней корректировки*

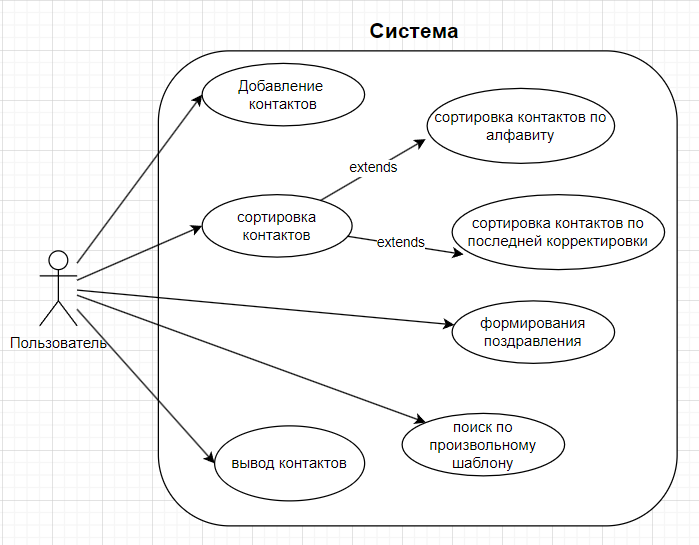
search(const string& query) *//метод для поиска по произвольному шаблону*

displayContacts() *//метод для вывода контактов*

generateСongratulation() *//метод для формирование поздравления по текущей дате*

AddressBook() *//конструктор по умолчанию*

## Диаграмма use case



## Листинг кода

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <algorithm>

#include <ctime>

using namespace std;

const int MAX\_CONTACTS = 100;

struct Contact {

string name;

string address;

string phone;

string workplace;

string position;

string relation;

string notes;

time\_t lastModified;

Contact() : name(""), address(""), phone(""), workplace(""), position(""), relation(""), notes(""), lastModified(0) {}

Contact(string n, string addr, string ph, string wp, string pos, string rel, string nt)

: name(n), address(addr), phone(ph), workplace(wp), position(pos), relation(rel), notes(nt) {

lastModified = time(nullptr);

}

};

class AddressBook {

private:

Contact contacts[MAX\_CONTACTS];

int numContacts;

public:

AddressBook() : numContacts(0) {}

void addContact(const Contact& newContact) {

if (numContacts < MAX\_CONTACTS) {

contacts[numContacts++] = newContact;

}

else {

cout << "Ошибка: превышено максимальное количество контактов.\n" << endl;

}

}

void addContact() {

if (numContacts < MAX\_CONTACTS) {

cout << "Введите имя: ";

cin >> contacts[numContacts].name;

cout << "Введите адрес: ";

cin >> contacts[numContacts].address;

cout << "Введите телефон: ";

cin >> contacts[numContacts].phone;

cout << "Введите место работы: ";

cin >> contacts[numContacts].workplace;

cout << "Введите должность: ";

cin >> contacts[numContacts].position;

cout << "Введите отношение: ";

cin >> contacts[numContacts].relation;

cout << "Введите примечания: ";

cin >> contacts[numContacts].notes;

contacts[numContacts].lastModified = time(nullptr);

cout << "Контакт успешно добавлен!!!\n" << endl;

numContacts++;

}

else {

cout << "Ошибка: превышено максимальное количество контактов.\n" << endl;

}

}

void sortByAlphabet() {

if (numContacts == 0) {

cout << "Список контактов пуст.\n" << endl;

}

else {

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

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

if (contacts[j].name > contacts[j + 1].name) {

swap(contacts[j], contacts[j + 1]);

}

}

}

cout << "Сортировка успешно завершена!!!\n";

}

}

void sortByLastModified() {

if (numContacts == 0) {

cout << "Список контактов пуст.\n" << endl;

}

else {

sort(contacts, contacts + numContacts, [](const Contact& a, const Contact& b) {

return a.lastModified > b.lastModified;

});

cout << "Сортировка успешно завершена!!!\n";

}

}

void search(const string& query) {

bool found = false;

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

if (contacts[i].name == query ||

contacts[i].address == query ||

contacts[i].phone == query ||

contacts[i].workplace == query ||

contacts[i].position == query ||

contacts[i].relation == query ||

contacts[i].notes == query) {

cout << "Контакт найден:" << endl;

cout << "Имя: " << contacts[i].name << endl;

cout << "Адрес: " << contacts[i].address << endl;

cout << "Телефон: " << contacts[i].phone << endl;

cout << "Место работы: " << contacts[i].workplace << endl;

cout << "Должность: " << contacts[i].position << endl;

cout << "Отношение: " << contacts[i].relation << endl;

cout << "Примечания: " << contacts[i].notes << endl;

cout << "Последняя модификация: " << ctime(&contacts[i].lastModified);

found = true;

}

}

if (!found) {

cout << "Контакт не найден.\n" << endl;

}

}

void displayContacts() {

char buffer[26];

if (numContacts == 0) {

cout << "Список контактов пуст.\n" << endl;

}

else {

setlocale(LC\_ALL, "rus");

cout << "Список контактов:" << endl;

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

cout << "Контакт " << i + 1 << ":" << endl;

cout << "Имя: " << contacts[i].name << endl;

cout << "Адрес: " << contacts[i].address << endl;

cout << "Телефон: " << contacts[i].phone << endl;

cout << "Место работы: " << contacts[i].workplace << endl;

cout << "Должность: " << contacts[i].position << endl;

cout << "Отношение: " << contacts[i].relation << endl;

cout << "Примечания: " << contacts[i].notes << endl;

cout << "Последняя модификация: " << ctime(&contacts[i].lastModified);

cout << "\n";

}

}

}

void generateСongratulation() {

if (numContacts == 0) {

cout << "Список контактов пуст.\n" << endl;

}

else {

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

cout << i <<" ) " << contacts[i].name << endl;

}

}

cout << "Выберите контакта для автоматического формирование поздравления с днем рождениия: ";

int number;

cin >> number;

cout << "С днем рождения, " << contacts[number].name << "!\n"<< time(nullptr) << endl;

contacts[number].lastModified = time(nullptr);

}

};

int main() {

setlocale(LC\_ALL, "rus");

string a;

AddressBook addressBook;

Contact contact1("Tologon", "Bishkek", "123456789", "Optima", "manager", "relative", "note\_1");

Contact contact2("Abulhair", "Osh", "987654321", "Kramer", "frontend developer", "colleague", "note\_2");

Contact contact3("Nurislam", "Talas", "456123789", "Bakai", "backend developer", "group\_mate", "note\_3");

addressBook.addContact(contact1);

addressBook.addContact(contact2);

addressBook.addContact(contact3);

int n;

cout << "Введите операцию:\n \n1) Добавить контакт \n2) Сортировать контакты по алфавиту \n3) Сортировать контакты по дате последней корректировки \n4) Вывести список контактов \n5) Меню \n6) Поиск по произвольному шаблону";

cout << "\n7) Формировать поздравление \n0) Выйти из программы \n";

do {

cout << "\nВведите операцию: ";

cin >> n;

switch (n) {

case 1:

system("cls");

cout << "\nВы выбрали операцию 1 (добавления контакта)\n" << endl;

addressBook.addContact();

addressBook.displayContacts();

break;

case 2:

system("cls");

cout << "\nВы выбрали операцию 2 (сортировка контактов по алфавиту)\n" << endl;

addressBook.sortByAlphabet();

addressBook.displayContacts();

break;

case 3:

system("cls");

cout << "\nВы выбрали операцию 3 (сортировка контактов по последней корректировки)\n" << endl;

addressBook.sortByLastModified();

addressBook.displayContacts();

break;

case 4:

system("cls");

cout << "\nВы выбрали операцию 4(вывод контактов)\n" << endl;

addressBook.displayContacts();

break;

case 5:

system("cls");

cout << "\nВы выбрали операцию 5 (вывод меню)\n" << endl;

cout << "Введите операцию:\n \n1) Добавить контакт \n2) Сортировать контакты по алфавиту \n3) Сортировать контакты по дате последней корректировки \n4) Вывести список контактов \n5) Меню \n6) Поиск по произвольному шаблону ";

cout << "\n7) Формировать поздравление \n0) Выйти из программы \n";

break;

case 6:

system("cls");

cout << "\nВы выбрали операцию 6 (поиск по произвольному шаблону)\n" << endl;

cout << "Введите произвольный шаблон для поиска:";

cin >> a;

addressBook.search(a);

break;

case 7:

system("cls");

cout << "\nВы выбрали операцию 7 (формирования поздравления)\n" << endl;

addressBook.generateСongratulation();

break;

}

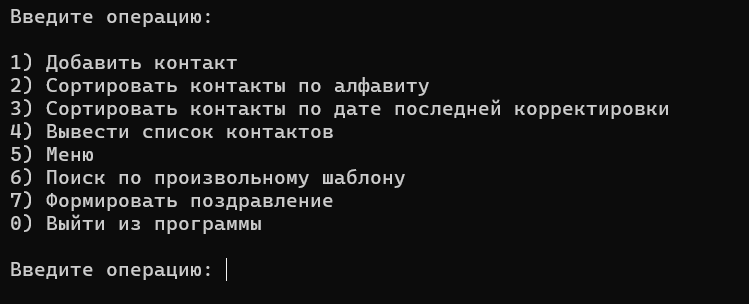
} while (n != 0);

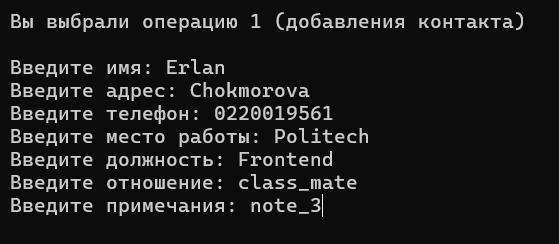
return 0;

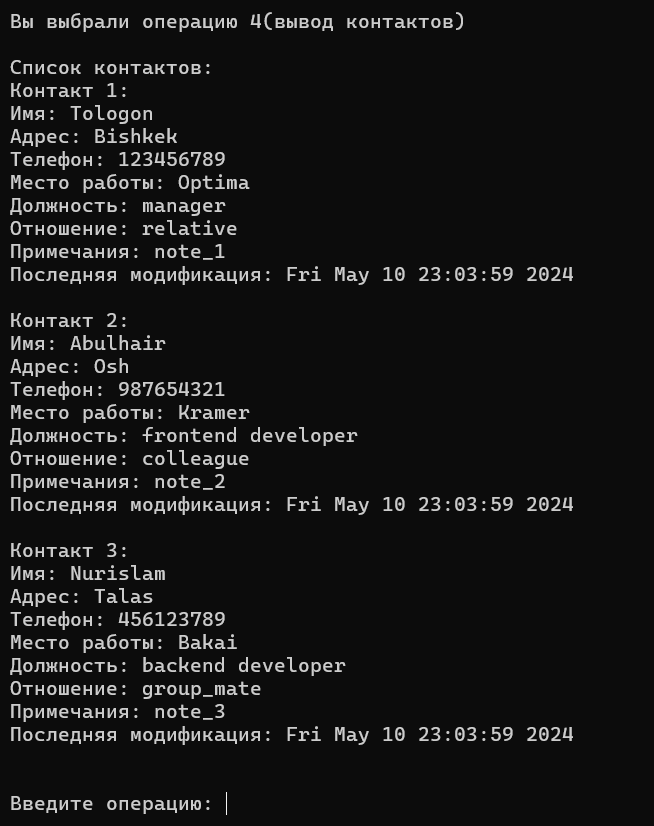
}

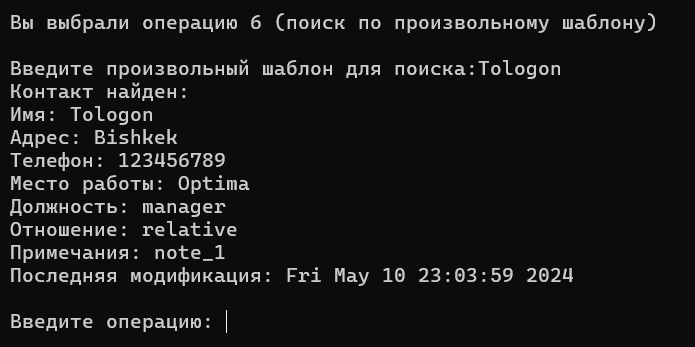
## Результат программы

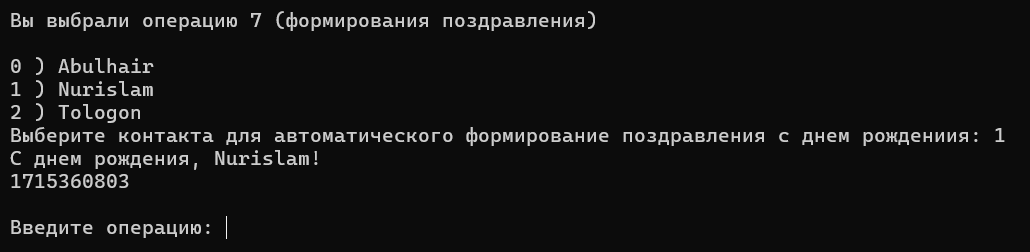
Меню программы:

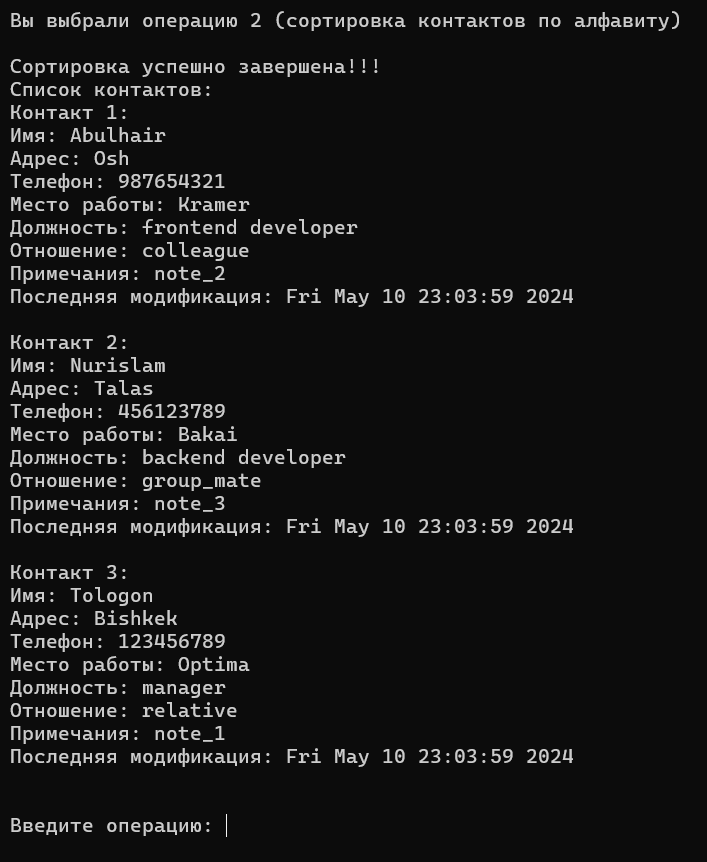


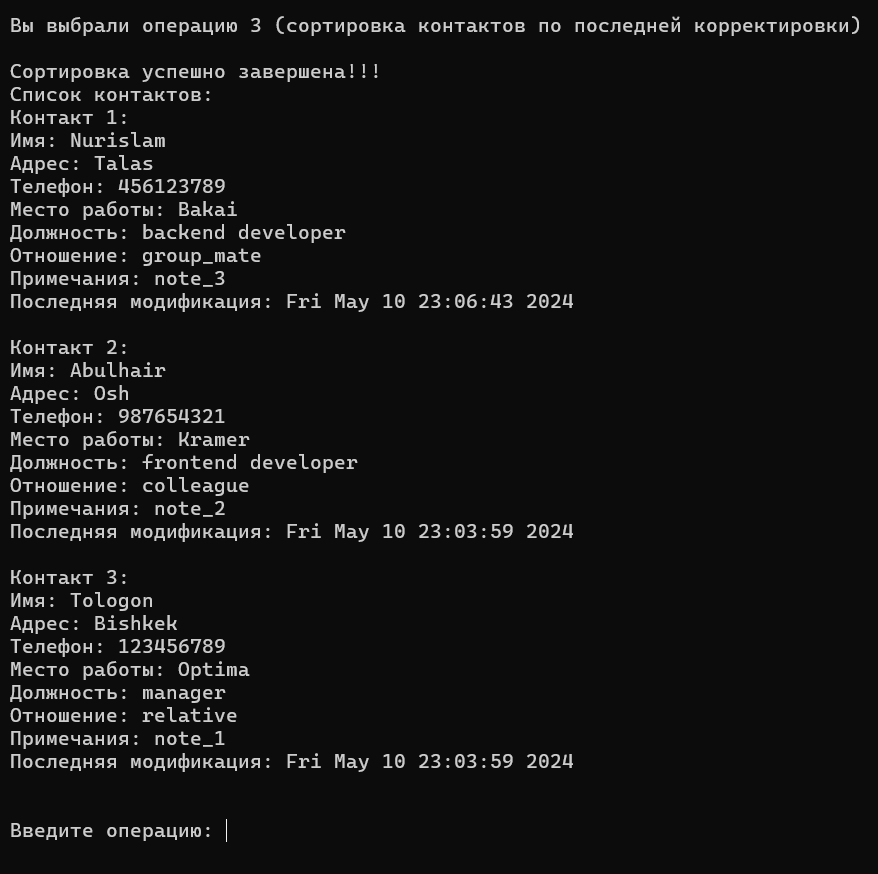












# Задача 17.30

Справочник фаната. База спортсменов: анкетные и ан-тропологические данные, гражданство, происхождение, вид спорта, клуб или команда, данные о личном рекорде или победах и так далее. Выбор по произвольному признаку. Поиск рекордсмена в заданном виде спорта.

## Описание задачи

**Структура Athlete**: Эта структура предназначена для хранения информации о спортсменах. Она содержит следующие поля и методы:

**Поля:**

string name; //имя

int age; //возраст

string nationality; //национальность

string origin; //происхождения

string sport; //вид спорта

string club; //клуб

string personalRecord; //личный рекорд спортсмена

**Методы:**

Конструктор по умолчанию: Athlete()

Конструктор: Athlete(string name, int age, string nationality, string origin, string sport, string club, string personalRecord)

**Класс** **FanGuide:** Этот класс представляет собой справочник фаната. Который содержит следующие поля и методы:

**Поля:**

Athlete athletes[MAX\_ATHLETES]; *//приватный массив хранящий список спортсменов*

int numAthletes; *//количество спортсменов*

**Методы:**

addAthlete() *//метод для добавление спортсменов*

sortByAlphabet() *//метод для сортировки спортсменов по алфавиту*

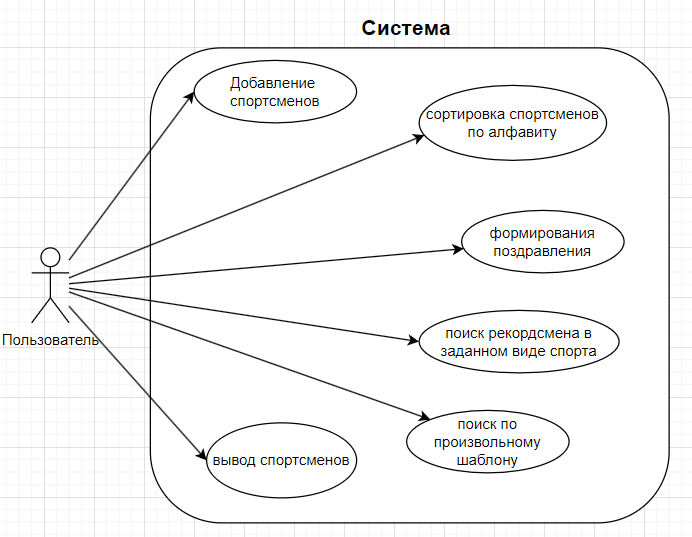
search(const string& query) *//метод для поиска по произвольному шаблону*

displayAthletes() *//метод для вывода спортсменов*

searchBySport(const string& sport) //*поиск рекордсмена в заданном виде спорта*

FanGuide() *//конструктор по умолчанию*

## Диаграмма use case

**

## Листинг кода

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <algorithm>

#include <ctime>

using namespace std;

const int MAX\_ATHLETES = 100;

struct Athlete {

string name;

int age;

string nationality;

string origin;

string sport;

string club;

string personalRecord;

Athlete() : name(""), age(0), nationality(""), origin(""), sport(""), club(""), personalRecord(""){}

Athlete(string name, int age, string nationality, string origin, string sport, string club, string personalRecord)

: name(name), age(age), nationality(nationality), origin(origin), sport(sport), club(club), personalRecord(personalRecord) {

}

};

class FanGuide {

private:

Athlete athletes[MAX\_ATHLETES];

int numAthletes;

public:

FanGuide() : numAthletes(0) {}

void addAthlete(const Athlete& newAthlete) {

if (numAthletes < MAX\_ATHLETES) {

athletes[numAthletes++] = newAthlete;

}

else {

cout << "Ошибка: превышено максимальное количество спортсменов.\n" << endl;

}

}

void addAthlete() {

if (numAthletes < MAX\_ATHLETES) {

cout << "Введите имя: ";

cin >> athletes[numAthletes].name;

cout << "Введите возраст: ";

cin >> athletes[numAthletes].age;

cout << "Введите национальность: ";

cin >> athletes[numAthletes].nationality;

cout << "Введите происхождение: ";

cin >> athletes[numAthletes].origin;

cout << "Введите вид спорта: ";

cin >> athletes[numAthletes].sport;

cout << "Введите клуб: ";

cin >> athletes[numAthletes].club;

cout << "Введите личний рекорд спортсмена: ";

cin >> athletes[numAthletes].personalRecord;

cout << "Спортсмен успешно добавлен!!!\n" << endl;

numAthletes++;

}

else {

cout << "Ошибка: превышено максимальное количество контактов.\n" << endl;

}

}

void sortByAlphabet() {

if (numAthletes == 0) {

cout << "Список спортсменов пуст.\n" << endl;

}

else {

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

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

if (athletes[j].name > athletes[j + 1].name) {

swap(athletes[j], athletes[j + 1]);

}

}

}

cout << "Сортировка успешно завершена!!!\n";

}

}

void search(const string& query) {

bool found = false;

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

if (athletes[i].name == query ||

/\*athletes[i].age == stoi(query) ||\*/

athletes[i].nationality == query ||

athletes[i].origin == query ||

athletes[i].sport == query ||

athletes[i].club == query ||

athletes[i].personalRecord == query) {

cout << "Имя: " << athletes[i].name << endl;

cout << "возраст: " << athletes[i].age << endl;

cout << "Национальность: " << athletes[i].nationality << endl;

cout << "Происхождение: " << athletes[i].origin << endl;

cout << "Вид спорта: " << athletes[i].sport << endl;

cout << "Клуб: " << athletes[i].club << endl;

cout << "Личный рекорд: " << athletes[i].personalRecord << endl;

found = true;

}

}

if (!found) {

cout << "Спортсмен не найден.\n" << endl;

}

}

void displayAthletes() {

char buffer[26];

if (numAthletes == 0) {

cout << "Список спортсменов пуст.\n" << endl;

}

else {

setlocale(LC\_ALL, "rus");

cout << "Список контактов:" << endl;

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

cout << "Имя: " << athletes[i].name << endl;

cout << "возраст: " << athletes[i].age << endl;

cout << "Национальность: " << athletes[i].nationality << endl;

cout << "Происхождение: " << athletes[i].origin << endl;

cout << "Вид спорта: " << athletes[i].sport << endl;

cout << "Клуб: " << athletes[i].club << endl;

cout << "Личный рекорд: " << athletes[i].personalRecord << endl;

cout << "\n";

}

}

}

void searchBySport(const string& sport) {

bool found = false;

int max = 0;

int num = 0;

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

if (athletes[i].sport == sport) {

if (stoi(athletes[i].personalRecord) > max) {

num = i;

max = stoi(athletes[i].personalRecord);

}

found = true;

}

}

cout << "Рекордсмен в виде спорта " << sport << " найден:" << endl;

cout << "Имя: " << athletes[num].name << endl;

cout << "Возраст: " << athletes[num].age << endl;

cout << "Национальность: " << athletes[num].nationality << endl;

cout << "Происхождение: " << athletes[num].origin << endl;

cout << "Клуб: " << athletes[num].club << endl;

cout << "Личный рекорд: " << athletes[num].personalRecord << endl;

if (!found) {

cout << "Рекордсмен в виде спорта " << sport << " не найден." << endl;

}

}

};

int main() {

setlocale(LC\_ALL, "rus");

string a;

FanGuide fanGuide;

Athlete athlete1("Tologon", 20, "kyrgyz", "kyrgyz", "tennis", "Tibhard", "100");

Athlete athlete2("Erlan", 20, "kyrgyz", "kyrgyz", "tennis", "Tibhard", "200");

Athlete athlete3("Abulhair", 20 , "kyrgyz", "kazak", "voleyball", "Politech", "400");

Athlete athlete4("Erjan", 20, "kyrgyz", "kyrgyz", "futball", "Alga", "500");

fanGuide.addAthlete(athlete1);

fanGuide.addAthlete(athlete2);

fanGuide.addAthlete(athlete3);

fanGuide.addAthlete(athlete4);

int n;

cout << "Введите операцию:\n \n1) Добавить спортсмена \n2) Сортировать спортсмена по алфавиту \n3) Вывести список контактов \n4) Меню \n5) Поиск по произвольному шаблону";

cout << "\n6) Поиск рекордсмена в заданном виде спорта \n0) Выйти из программы \n";

do {

cout << "\nВведите операцию: ";

cin >> n;

switch (n) {

case 1:

system("cls");

cout << "\nВы выбрали операцию 1 (добавления спортсмена)\n" << endl;

fanGuide.addAthlete();

fanGuide.displayAthletes();

break;

case 2:

system("cls");

cout << "\nВы выбрали операцию 2 (сортировка спортсмена по алфавиту)\n" << endl;

fanGuide.sortByAlphabet();

fanGuide.displayAthletes();

break;

case 3:

system("cls");

cout << "\nВы выбрали операцию 3 (вывод спортсменов)\n" << endl;

fanGuide.displayAthletes();

break;

case 4:

system("cls");

cout << "\nВы выбрали операцию 4 (вывод меню)\n" << endl;

cout << "Введите операцию:\n \n1) Добавить спортсмена \n2) Сортировать спортсмена по алфавиту \n3) Вывести список контактов \n4) Меню \n5) Поиск по произвольному шаблону";

cout << "\n6) Поиск рекордсмена в заданном виде спорта \n0) Выйти из программы \n";

break;

case 5:

system("cls");

cout << "\nВы выбрали операцию 5 (поиск по произвольному шаблону)\n" << endl;

cout << "Введите произвольный шаблон для поиска:";

cin >> a;

fanGuide.search(a);

break;

case 6:

system("cls");

cout << "\nВы выбрали операцию 6 (поиск рекордсмена в заданном виде спорта)\n" << endl;

cout << "Введите вид спорта для поиска рекордсмена:";

cin >> a;

fanGuide.searchBySport(a);

break;

default:

system("cls");

cout << "Программа завершена!";

break;

}

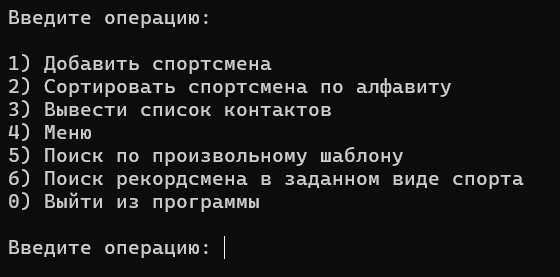
} while (n != 0);

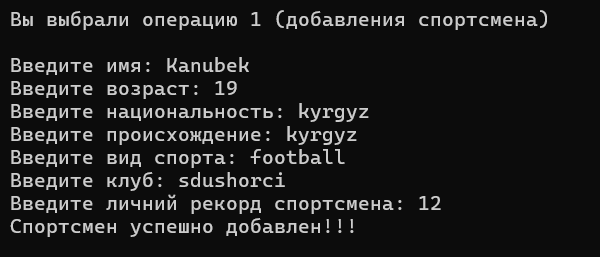
return 0;

}

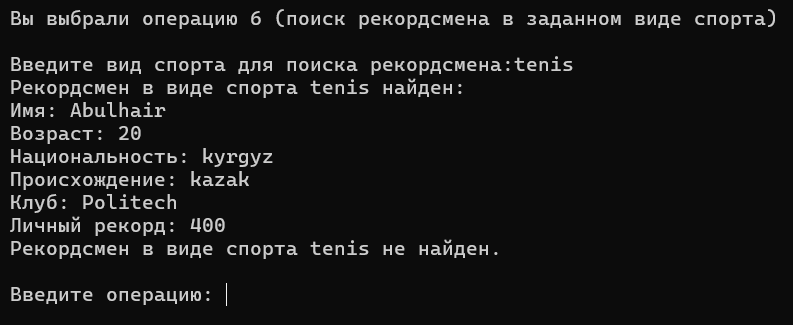
## Результат программы

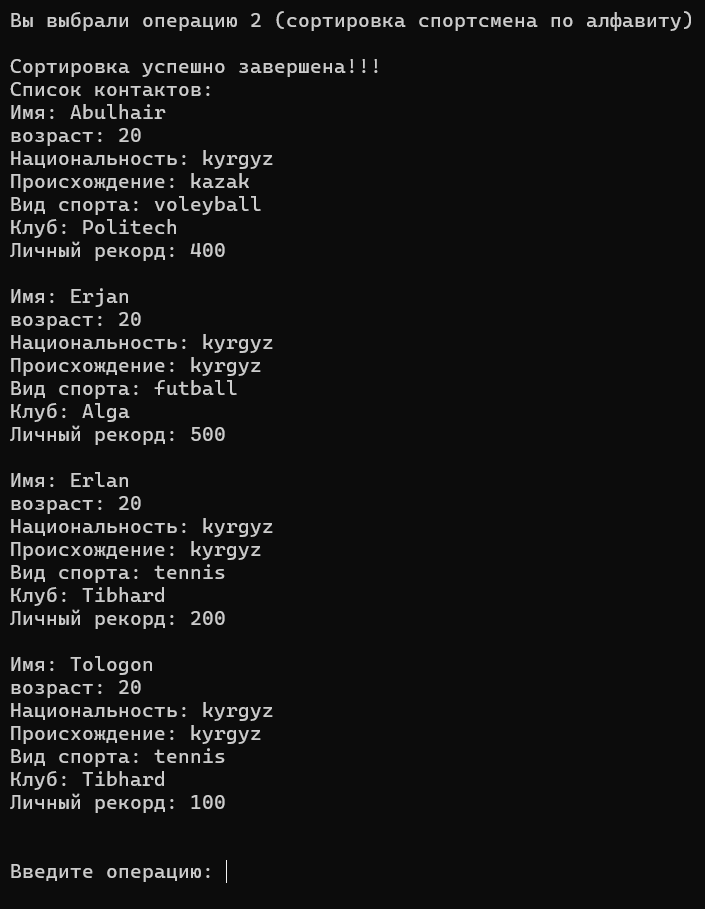
Меню программы:

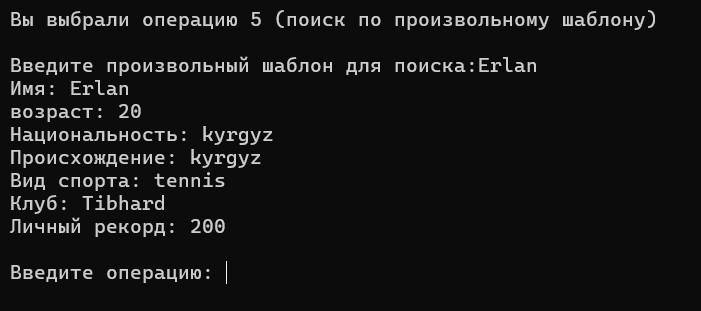












# Заключение

В результате разработки данной программы мы улучшили навыки применения принципов объектно-ориентированного программирования (ООП) для создания эффективных и структурированных решений. Кроме того, мы расширили наши знания и опыт работы с языком программирования C++, который предоставляет широкие возможности для создания производительного и надежного программного обеспечения.