**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №2

«Переход от структур к классам»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-23Б |  | преподаватель каф. ИУ5 |
| Лачина Екатерина |  | Аладин Д.В. |
| Подпись и дата:  19.03.24 |  | Подпись и дата: |



Москва, 2024 г.

1. **Задача**

Часть 1:

* Создать класс "Дроби" для выполнения арифметических операций над обыкновенными дробями. Внутреннее представление дробей (состав полей класса) должно обеспечивать эффективное выполнение операций над дробями и может отличаться от представления дробей на экране монитора, которое должно быть удобным для пользователя. Например, внутри объекта класса "Дроби" может храниться неправильная дробь.
* Разработать конструктор по умолчанию и конструктор, который преобразует строку, содержащую внешнее представление дроби, в объект класса "Дроби".
* Перегрузить операции потокового ввода дроби с клавиатуры и вывода ее на экран монитора. При вводе выполнять сокращение дроби.
* Создать многофайловый проект и отладить программу, которая создает один объект класса "Дроби" и выводит значения его полей на экран (эта программа состоит из 5 первых операторов программы, приведённой в Приложении 1).

Часть 2:

* Дополнить класс функциями-членами класса и функциями-друзьями класса, которые необходимы для выполнения программы из Приложения 1.
* Выполнить программу из Приложения 1 и сравнить результаты с тестовым примером.

1. **Листинг**

**Папка Planet**

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.2)

set(project "Planet")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

func.cpp)

set(**${project}**\_HEADERS

planet.h)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES}

${**${project}**\_HEADERS})

add\_library(**${project}**

STATIC

${**${project}**\_SOURCE\_LIST})

**func.cpp**

#include "planet.h"

int Planet1::menu() {

std::cout << std::endl;

std::cout << "Меню:" << std::endl;

std::cout << "1. Прочитать базу данных" << std::endl;

std::cout << "2. Записать в базу данных" << std::endl;

std::cout << "3. Найти планету и информацию о ней" << std::endl;

std::cout << "4. Вывести базу данных" << std::endl;

std::cout << "5. Отсортировать планеты" << std::endl;

std::cout << "6. Добавить планету" << std::endl;

std::cout << "7. Удалить планету" << std::endl;

std::cout << "8. Редактировать планету" << std::endl;

std::cout << "9.Выход" << std::endl;

std::cout << "Введите цифру: ";

int choice;

std::cin >> choice;

return choice;

}

Planet1::Planet\* Planet1::read\_db(const char\* file\_name, Planet\* planets, int& count) {

std::ifstream file(file\_name);

file >> count;

Planet\* new\_planets = new Planet[count];

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

file >> new\_planets[i];

}

file.close();

delete[] planets;

return new\_planets;

}

void Planet1::add\_planet(Planet\* planets, int& count) {

char temp[10];

float diametr;

int life;

int satellites;

std::cout << "Введите имя планеты: ";

std::cin >> temp;

planets[count].set\_name(temp);

std::cout << "Введите диаметр планеты: ";

std::cin >> diametr;

planets[count].set\_diametr(diametr);

std::cout << "Введите жизнь планеты (0 - нет, 1 - да): ";

std::cin >> life;

planets[count].set\_life\_long(life);

std::cout << "Введите количество спутников: ";

std::cin >> satellites;

planets[count].set\_kol\_satellites\_0(satellites);

count += 1;

}

void Planet1::write\_db(const char\* file\_name, Planet\* planets, int& count) {

std::ofstream file(file\_name, std::ios::out);

if (file.is\_open()) {

file << count << std::endl;

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

file << planets[i];

file << std::endl;

}

}

else {

std::cout << "Ошибка записи в файл";

}

file.close();

}

void Planet1::delete\_planet(Planet\*& planets, int& count) {

char temp[10];

std::cout << "Введите название планеты, для удаления: ";

std::cin >> temp;

char\* name = new char[strlen(temp) + 1];

strcpy\_s(name, strlen(name) + 1, temp);

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

if (strcmp(planets[i].pName, name) == 0) {

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

planets[j] = planets[j + 1];

}

count--;

std::cout << "Планета успешно удалена" << std::endl;

}

}

Planet\* pl = new Planet[count];

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

pl[i] = planets[i];

}

delete[] planets;

planets = pl;

}

void Planet1::find\_planet(Planet\* planets, int& count) {

char temp[10];

std::cout << "Введите название планеты, для поиска информации: ";

std::cin >> temp;

int g = 0;

char\* name = new char[strlen(temp) + 1];

strcpy\_s(name, strlen(name) + 1, temp);

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

if (strcmp(planets[i].get\_name(), name) == 0) {

g++;

std::cout << "Информация о планете:" << std::endl;

std::cout << "Диаметр: " << planets[i].get\_diametr() << std::endl;

std::cout << "Продолжительность жизни: " << planets[i].get\_life\_long() << std::endl;

std::cout << "Количество спутников: " << planets[i].get\_kol\_satellites\_0() << std::endl;

}

}

if (g == 0) {

std::cout << "Планета с именем " << name << " не найдена." << std::endl;

}

}

void Planet1::sort\_db(Planet\* planets, int& count) {

int minIndex;

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

for (int j = i + 1; j < count; j++) {

if (planets[j] < planets[i]) {

Planet temp = planets[j];

planets[j] = planets[i];

planets[i] = temp;

}

}

}

}

void Planet1::edit\_planet(Planet\* planets, int& count) {

char temp[10];

std::cout << "Введите имя планеты для редактирования: ";

std::cin >> temp;

char\* searchName = new char(strlen(temp) + 1);

strcpy\_s(searchName, strlen(searchName) + 1, temp);

int index = -1;

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

if (strcmp(planets[i].get\_name(), searchName) == 0) {

index = i;

break;

}

}

if (index == -1) {

std::cout << "Планета с именем " << searchName << " не найдена." << std::endl;

}

else {

std::cout << "Планета найдена. Введите новые данные:" << std::endl;

Planet newPlanet;

std::cin >> newPlanet;

planets[index] = newPlanet;

std::cout << "Планета успешно отредактирована." << std::endl;

}

}

Planet1::Planet::Planet(const Planet& p1) {

this->life\_long = p1.life\_long;

this->dimetr = p1.dimetr;

this->kol\_satellites = p1.kol\_satellites;

this->pName = new char[strlen(p1.pName) + 1];

for (int i = 0;i < strlen(p1.pName) + 1;i++) {

this->pName[i] = p1.pName[i];

}

}

bool Planet1::Planet::operator<(const Planet& p1) {

return (strcmp(this->pName, p1.pName) < 0);

}

bool Planet1::Planet::operator>(const Planet& p1) {

return (strcmp(this->pName, p1.pName) > 0);

}

Planet1::Planet& Planet1::Planet::operator=(const Planet& p1) {

if (&p1 == this) {

return \*this;

}

this->life\_long = p1.life\_long;

this->dimetr = p1.dimetr;

this->kol\_satellites = p1.kol\_satellites;

set\_name(p1.pName);

return \*this;

}

std::ostream& Planet1::operator<<(std::ostream& out, const Planet& planet) {

out << planet.pName << " " << planet.dimetr << " " << planet.life\_long << " " << planet.kol\_satellites;

return out;

}

std::istream& Planet1::operator>>(std::istream& in, Planet& planet) {

char word[Size];

in >> word;

planet.pName = new char[strlen(word) + 1];

for (int i = 0;i < strlen(word) + 1;i++) {

planet.pName[i] = word[i];

}

in >> planet.dimetr >> planet.life\_long >> planet.kol\_satellites;

return in;

}

**planet.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#ifndef PLANET

#define PLANET

#include <iostream>

#include <fstream>

#include <cstring>

namespace Planet1 {

const int Size = 9;

class Planet {

private:

char\* pName;

float dimetr;

int life\_long;

int kol\_satellites;

public:

Planet() {

pName = new char[1];

pName[0] = '\0';

dimetr = 0;

life\_long = 0;

kol\_satellites = 0;

}

Planet(char\* pName\_0, float dimetr\_0, int life\_long\_0, int kol\_satellites\_0) {

pName = new char[strlen(pName\_0) + 1];

strcpy\_s(pName, strlen(pName\_0) + 1, pName\_0);

dimetr = dimetr\_0;

life\_long = life\_long\_0;

kol\_satellites = kol\_satellites\_0;

}

void set\_name(char\* name) {

if (this->pName != nullptr) {

delete[] this->pName;

}

this->pName = new char[strlen(name) + 1];

strcpy(this->pName, name);

}

void set\_diametr(float dimetr0) { dimetr = dimetr0; }

void set\_life\_long(int life\_long0) { life\_long = life\_long0; }

void set\_kol\_satellites\_0(int kol\_satellites0) { kol\_satellites = kol\_satellites0; }

char\* get\_name() { return pName; }

float get\_diametr() { return dimetr; }

int get\_life\_long() { return life\_long; }

int get\_kol\_satellites\_0() { return kol\_satellites; }

void print\_db2() {

//for (int i = 0; i < strlen(pName) + 1; ++i) { cout << pName[i]; }

std::cout << pName;

std::cout << ", " << dimetr << ", " << life\_long << ", " << kol\_satellites << ". " << std::endl;

}

friend Planet\* read\_db(const char\* file\_name, Planet\* planets, int& count);

friend void write\_db(const char\* file\_name, Planet\* planets, int& count);

friend void sort\_db(Planet\* planets, int& count);

friend void add\_planet(Planet\* planets, int& count);

friend void delete\_planet(Planet\*& planets, int& count);

friend void find\_planet(Planet\* planets, int& count);

friend void edit\_planet(Planet\* planets, int& count);

friend std::ostream& operator<<(std::ostream& out, const Planet& planet);

friend std::istream& operator>>(std::istream& in, Planet& planet);

bool operator<(const Planet& p1);

bool operator>(const Planet& p1);

Planet& operator=(const Planet& p1);

Planet(const Planet& p1);

~Planet() { delete[] pName; }

};

void edit\_planet(Planet\* planets, int& count);

void sort\_db(Planet\* planets, int& count);

void find\_planet(Planet\* planets, int& count);

void delete\_planet(Planet\*& planets, int& count);

void write\_db(const char\* file\_name, Planet\* planets, int& count);

void add\_planet(Planet\* planets, int& count);

void add\_planet(Planet\* planets, int& count);

Planet\* read\_db(const char\* file\_name, Planet\* planets, int& count);

int menu();

}

#endif

**Папка Avia**

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.2)

set(project "Avia")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

aviafunc.cpp)

set(**${project}**\_HEADERS

avia.h)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES}

${**${project}**\_HEADERS})

add\_library(**${project}**

STATIC

${**${project}**\_SOURCE\_LIST})

**aviafunc.cpp**

#include "avia.h"

int Avia1::menu() {

std::cout << std::endl;

std::cout << "Меню:" << std::endl;

std::cout << "1. Прочитать базу данных" << std::endl;

std::cout << "2. Записать в базу данных" << std::endl;

std::cout << "3. Найти информацию о билете" << std::endl;

std::cout << "4. Вывести базу данных" << std::endl;

std::cout << "5. Отсортировать билеты" << std::endl;

std::cout << "6. Добавить билет" << std::endl;

std::cout << "7. Удалить билет" << std::endl;

std::cout << "8. Редактировать билет" << std::endl;

std::cout << "9.Выход" << std::endl;

std::cout << "Введите цифру: ";

int choice;

std::cin >> choice;

return choice;

}

Avia1::Avia\* Avia1::read\_db(const char\* file\_name, Avia\* tickets, int& count) {

std::ifstream file(file\_name);

file >> count;

Avia\* new\_tickets = new Avia[count];

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

file >> new\_tickets[i];

}

file.close();

delete[] tickets;

return new\_tickets;

}

void Avia1::add\_ticket(Avia\* tickets, int& count) {

char temp[10];

float dist;

int way;

int transfers;

std::cout << "Введите название страны для полета: ";

std::cin >> temp;

tickets[count].set\_name(temp);

std::cout << "Введите расстояние до этой страны: ";

std::cin >> dist;

tickets[count].set\_dist(dist);

std::cout << "Введите продолжительность полета (кол-во часов): ";

std::cin >> way;

tickets[count].set\_way\_long(way);

std::cout << "Введите количество пересадок: ";

std::cin >> transfers;

tickets[count].set\_kol\_transfers\_0(transfers);

count += 1;

}

void Avia1::write\_db(const char\* file\_name, Avia\* tickets, int& count) {

std::ofstream file(file\_name, std::ios::out);

if (file.is\_open()) {

file << count << std::endl;

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

file << tickets[i];

file << std::endl;

}

}

else {

std::cout << "Ошибка записи в файл";

}

file.close();

}

void Avia1::delete\_ticket(Avia\*& tickets, int& count) {

char temp[10];

std::cout << "Введите название билета по стране, для удаления: ";

std::cin >> temp;

char\* name = new char[strlen(temp) + 1];

strcpy\_s(name, strlen(name) + 1, temp);

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

if (strcmp(tickets[i].pName, name) == 0) {

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

tickets[j] = tickets[j + 1];

}

count--;

std::cout << "Билет успешно удален" << std::endl;

}

}

Avia\* av = new Avia[count];

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

av[i] = tickets[i];

}

delete[] tickets;

tickets = av;

}

void Avia1::find\_ticket(Avia\* tickets, int& count) {

char temp[10];

std::cout << "Введите название билета по стране, для поиска информации: ";

std::cin >> temp;

int g = 0;

char\* name = new char[strlen(temp) + 1];

strcpy\_s(name, strlen(name) + 1, temp);

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

if (strcmp(tickets[i].get\_name(), name) == 0) {

g++;

std::cout << "Информация о билете:" << std::endl;

std::cout << "Расстояние до страны: " << tickets[i].get\_dist() << std::endl;

std::cout << "Продолжительность полета: " << tickets[i].get\_way\_long() << std::endl;

std::cout << "Количество пересадок: " << tickets[i].get\_kol\_transfers\_0() << std::endl;

}

}

if (g == 0) {

std::cout << "Билет с именем " << name << " не найден." << std::endl;

}

}

void Avia1::sort\_db(Avia\* tickets, int& count) {

int minIndex;

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

for (int j = i + 1; j < count; j++) {

if (tickets[j] < tickets[i]) {

Avia temp = tickets[j];

tickets[j] = tickets[i];

tickets[i] = temp;

}

}

}

}

void Avia1::edit\_ticket(Avia\* tickets, int& count) {

char temp[10];

std::cout << "Введите имя билета по стране для редактирования: ";

std::cin >> temp;

char\* searchName = new char(strlen(temp) + 1);

strcpy\_s(searchName, strlen(searchName) + 1, temp);

int index = -1;

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

if (strcmp(tickets[i].get\_name(), searchName) == 0) {

index = i;

break;

}

}

if (index == -1) {

std::cout << "Билет с именем " << searchName << " не найден." << std::endl;

}

else {

std::cout << "Бител найден. Введите новые данные:" << std::endl;

Avia newTicket;

std::cin >> newTicket;

tickets[index] = newTicket;

std::cout << "Билет успешно отредактирован." << std::endl;

}

}

Avia1::Avia::Avia(const Avia& p1) {

this->way\_long = p1.way\_long;

this->dist = p1.dist;

this->kol\_transfers = p1.kol\_transfers;

this->pName = new char[strlen(p1.pName) + 1];

for (int i = 0;i < strlen(p1.pName) + 1;i++) {

this->pName[i] = p1.pName[i];

}

}

bool Avia1::Avia::operator<(const Avia& p1) {

return (strcmp(this->pName, p1.pName) < 0);

}

bool Avia1::Avia::operator>(const Avia& p1) {

return (strcmp(this->pName, p1.pName) > 0);

}

Avia1::Avia& Avia1::Avia::operator=(const Avia& p1) {

if (&p1 == this) {

return \*this;

}

this->way\_long = p1.way\_long;

this->dist = p1.dist;

this->kol\_transfers = p1.kol\_transfers;

set\_name(p1.pName);

return \*this;

}

std::ostream& Avia1::operator<<(std::ostream& out, const Avia1::Avia& ticket) {

out << ticket.pName << " " << ticket.dist << " " << ticket.way\_long << " " << ticket.kol\_transfers;

return out;

}

std::istream& Avia1::operator>>(std::istream& in, Avia1::Avia& ticket) {

char word[Size];

in >> word;

ticket.pName = new char[strlen(word) + 1];

for (int i = 0;i < strlen(word) + 1;i++) {

ticket.pName[i] = word[i];

}

in >> ticket.dist >> ticket.way\_long >> ticket.kol\_transfers;

return in;

}

**avia.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#ifndef AVIA

#define AVIA

#include <iostream>

#include <fstream>

#include <cstring>

namespace Avia1 {

const int Size = 10;

class Avia {

private:

char\* pName;

float dist;

int way\_long;

int kol\_transfers;

public:

Avia() {

pName = new char[1];

pName[0] = '\0';

dist = 0;

way\_long = 0;

kol\_transfers = 0;

}

Avia(char\* pName\_0, float dist\_0, int way\_long\_0, int kol\_transfers\_0) {

pName = new char[strlen(pName\_0) + 1];

strcpy\_s(pName, strlen(pName\_0) + 1, pName\_0);

dist = dist\_0;

way\_long = way\_long\_0;

kol\_transfers = kol\_transfers\_0;

}

void set\_name(char\* name) {

if (this->pName != nullptr) {

delete[] this->pName;

}

this->pName = new char[strlen(name) + 1];

strcpy(this->pName, name);

}

void set\_dist(float dist0) { dist = dist0; }

void set\_way\_long(int way\_long0) { way\_long = way\_long0; }

void set\_kol\_transfers\_0(int kol\_transfers0) { kol\_transfers = kol\_transfers0; }

char\* get\_name() { return pName; }

float get\_dist() { return dist; }

int get\_way\_long() { return way\_long; }

int get\_kol\_transfers\_0() { return kol\_transfers; }

void print\_db1() {

std::cout << pName;

std::cout << ", " << dist << ", " << way\_long << ", " << kol\_transfers << ". " << std::endl;

}

friend Avia\* read\_db(const char\* file\_name, Avia\* tickets, int& count);

friend void write\_db(const char\* file\_name, Avia\* tickets, int& count);

friend void sort\_db(Avia\* tickets, int& count);

friend void add\_ticket(Avia\* tickets, int& count);

friend void delete\_ticket(Avia\*& tickets, int& count);

friend void find\_ticket(Avia\* tickets, int& count);

friend void edit\_ticket(Avia\* tickets, int& count);

friend std::ostream& operator<<(std::ostream& out, const Avia& ticket);

friend std::istream& operator>>(std::istream& in, Avia& ticket);

bool operator<(const Avia& p1);

bool operator>(const Avia& p1);

Avia& operator=(const Avia& p1);

Avia(const Avia& p1);

~Avia() { delete[] pName; }

};

void edit\_ticket(Avia\* tickets, int& count);

void sort\_db(Avia\* tickets, int& count);

void find\_ticket(Avia\* tickets, int& count);

void delete\_ticket(Avia\*& tickets, int& count);

void write\_db(const char\* file\_name, Avia\* tickets, int& count);

void add\_ticket(Avia\* tickets, int& count);

void add\_ticket(Avia\* tickets, int& count);

Avia\* read\_db(const char\* file\_name, Avia\* tickets, int& count);

int menu();

}

#endif

**Главные файлы**

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.2)

set(project "Laba22Cmake")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

configure\_file(**${CMAKE\_CURRENT\_SOURCE\_DIR}**/avia.txt

**${CMAKE\_CURRENT\_BINARY\_DIR}** COPYONLY)

configure\_file(**${CMAKE\_CURRENT\_SOURCE\_DIR}**/planet.txt

**${CMAKE\_CURRENT\_BINARY\_DIR}** COPYONLY)

set(**${project}**\_SOURCES

main.cpp)

add\_subdirectory(Avia)

add\_subdirectory(Planet)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES})

add\_executable(**${project}**

${**${project}**\_SOURCE\_LIST})

target\_link\_libraries(**${project}** Avia)

target\_link\_libraries(**${project}** Planet)

**main.cpp**

#include <iostream>

#include "Avia/avia.h"

#include "Planet/planet.h"

int main(){

setlocale(LC\_ALL, "Russian");

std::cout << "С какой библиотекой Вы хотите работать (0/1) 0 - Планеты, 1 - Авиабилет: ";

int n;

std::cin >> n;

char\* file\_name;

if (n == 0) {

file\_name = "planet.txt";

int count = 0;

Planet1::Planet\* planets = new Planet1::Planet[0];

while (true) {

switch (Planet1::menu()) {

case 1:

planets = Planet1::read\_db(file\_name, planets, count);

break;

case 2:

Planet1::write\_db(file\_name, planets, count);

break;

case 3:

Planet1::find\_planet(planets, count);

break;

case 4:

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

planets[i].print\_db2();

}

break;

case 5:

Planet1::sort\_db(planets, count);

break;

case 6: {

Planet1::Planet\* new\_planets = new Planet1::Planet[count + 1];

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

new\_planets[i] = planets[i];

}

delete[] planets;

planets = new\_planets;

Planet1::add\_planet(planets, count);

break;}

case 7:

Planet1::delete\_planet(planets, count);

break;

case 8:

Planet1::edit\_planet(planets, count);

break;

case 9:

return 0;

default:

std::cout << " Неправильный ввод" << std::endl;

break;

}

}

}

else {

file\_name = "avia.txt";

int count = 0;

Avia1::Avia\* tickets = new Avia1::Avia[0];

while (true) {

switch (Avia1::menu()) {

case 1:

tickets = read\_db(file\_name, tickets, count);

break;

case 2:

Avia1::write\_db(file\_name, tickets, count);

break;

case 3:

Avia1::find\_ticket(tickets, count);

break;

case 4:{

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

tickets[i].print\_db1();

}

}

break;

case 5:

Avia1::sort\_db(tickets, count);

break;

case 6: {

Avia1::Avia\* new\_tickets = new Avia1::Avia[count + 1];

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

new\_tickets[i] = tickets[i];

}

delete[] tickets;

tickets = new\_tickets;

Avia1::add\_ticket(tickets, count);

break;}

case 7:

Avia1::delete\_ticket(tickets, count);

break;

case 8:

Avia1::edit\_ticket(tickets, count);

break;

case 9:

return 0;

default:

std::cout << " Неправильный ввод" << std::endl;

break;

}

}

}

}

**planet.txt**

9

Mercury 4878 0 0

Earth 12774 1 1

Mars 6786 1 2

Jupiter 142796 0 16

Saturn 120000 0 17

Uranus 51108 0 5

Kate 1 0 43

Arobus 1 0 43

Loopka 1 0 0

**avia.txt**

9

Bali 4878 8 2

Nigeria 12774 4 2

Moscow 6786 8 0

Sofia 142796 4 1

Afins 120000 4 1

Lalalend 51108 100 10

Tokio 1000 8 2

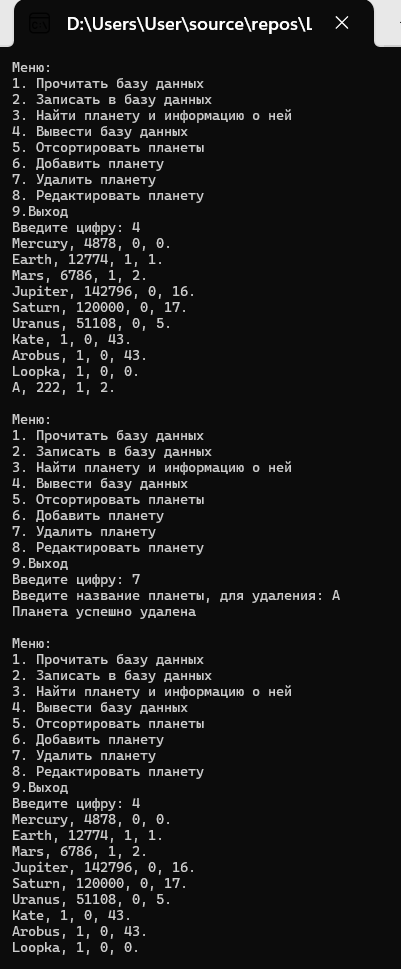
Amsterdam 20202 8 1

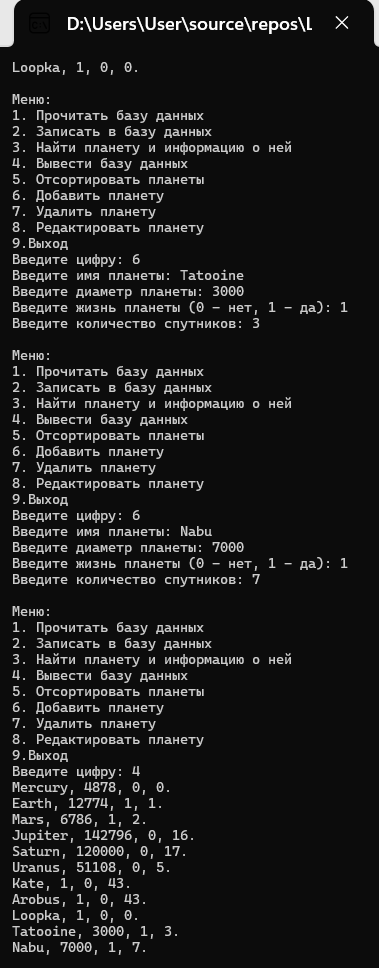
Varshava 30303 5 1

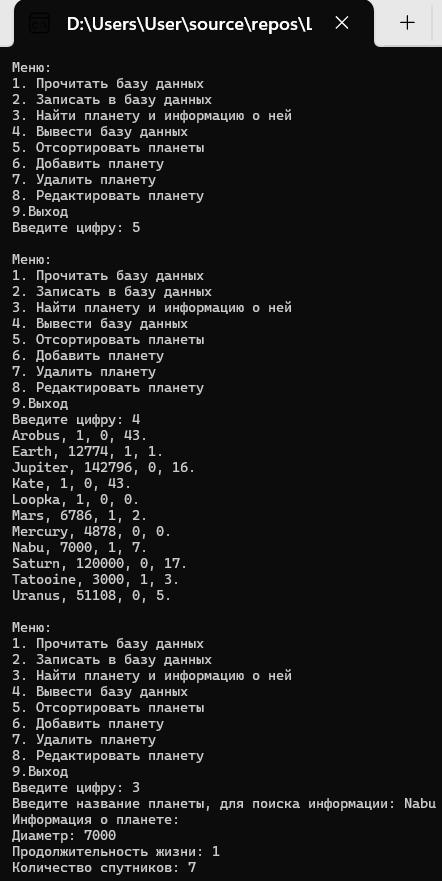
1. **Тестирование**

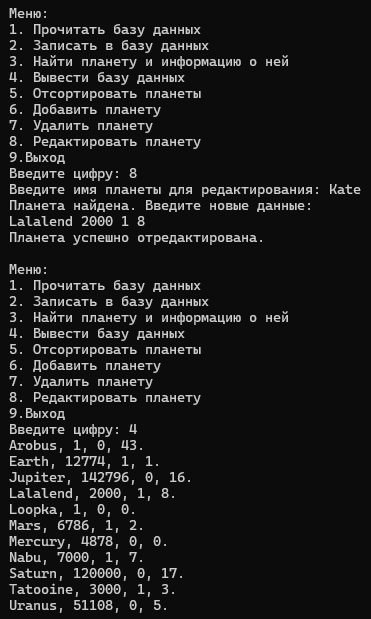
Для планет:

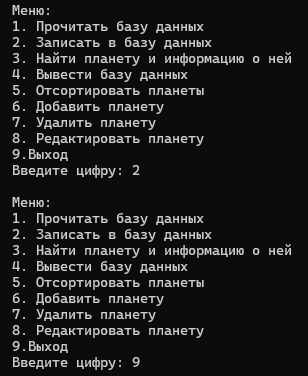




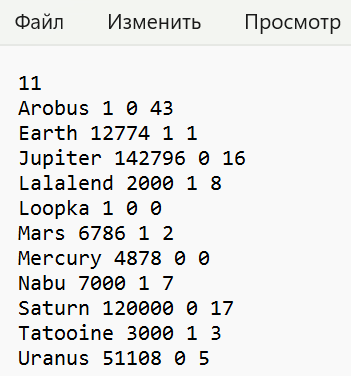




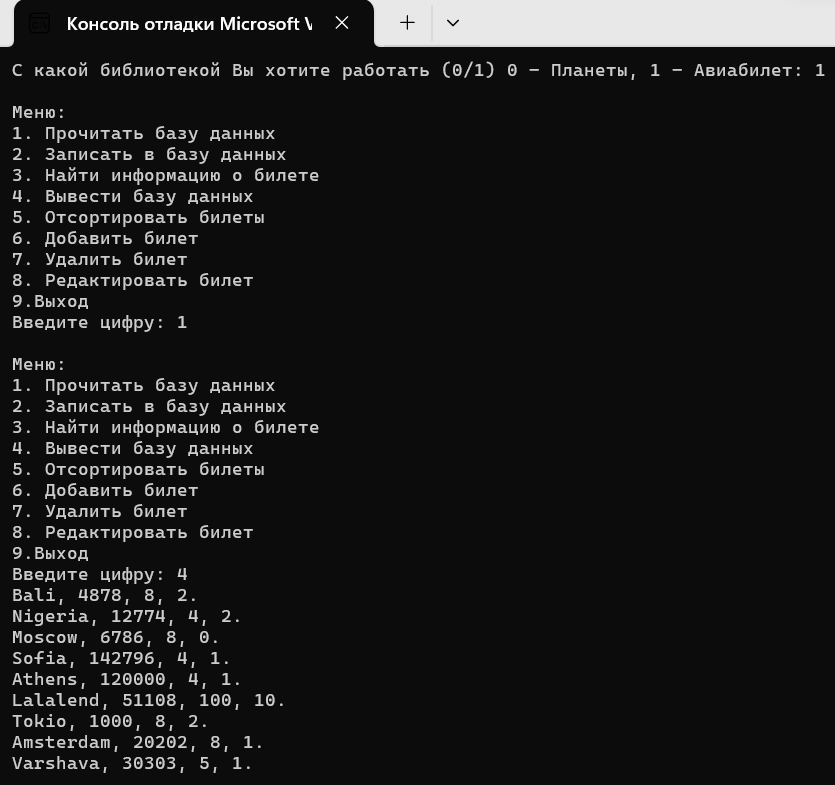


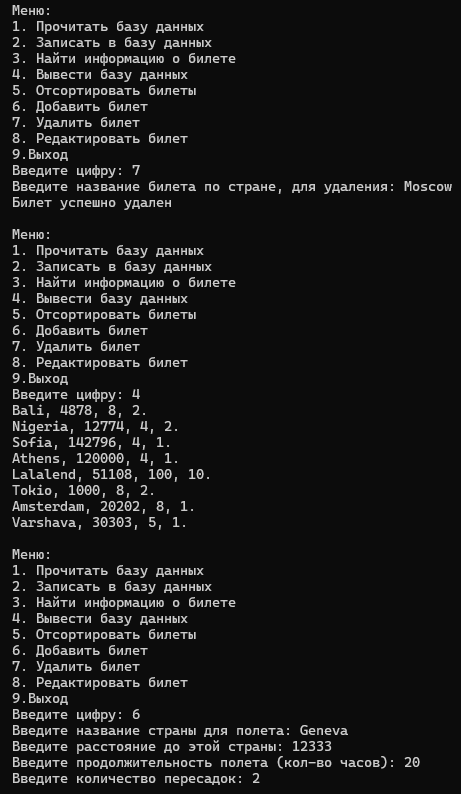


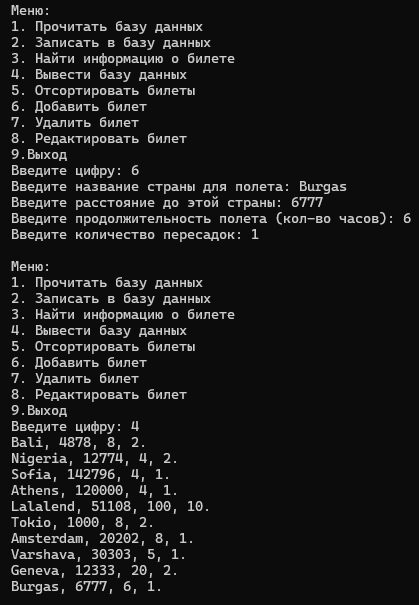
planet.txt



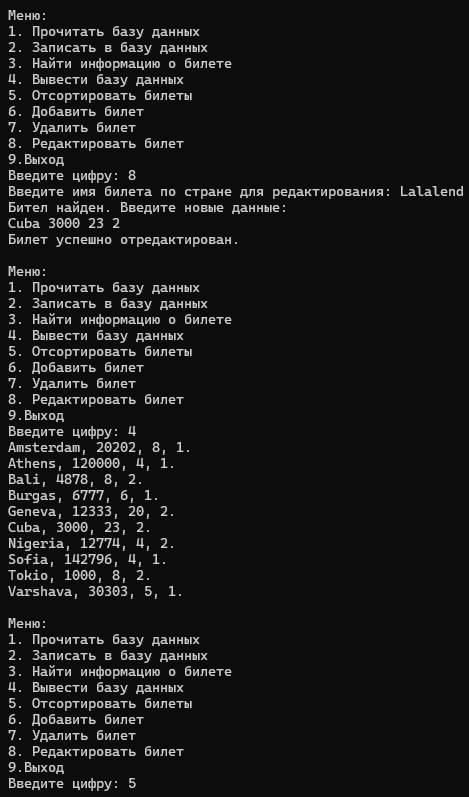
Для авиабилетов:













avia.txt

