**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 ИУ5-24б |  | преподаватель каф. ИУ5 |
| Чернев Н.А. |  |  |
| Подпись и дата: 08.03.2024 |  | Подпись и дата: |
|  |  |  |

Москва, 2024 г.

**Постановка задачи**

**Часть 1:**

1. Изучить материал о переходе от структуры к классу.

2. Разработать класс "Планета" с четырьмя характеристиками, используя **char\*** для строк.

3. Динамически выделять память для строк.

4. Использовать деструктор для освобождения памяти.

5. Создать статическую библиотеку для класса "Планета" и тестовую программу для вывода данных объекта на экран.

6. Реализовать функции для работы с БД: чтение из файла, запись в файл, сортировка, добавление, удаление, редактирование и вывод на экран.

7. Создать текстовый файл в кодировке ASCII с данными о планетах для отладки.

**Часть 2:**

1. Изучить перегрузку операции **<<**.

2. Перегрузить конструктор копирования, деструктор и оператор присваивания с выводом информации о создании/удалении объектов.

3. Перегрузить операцию **>>** для чтения данных о планетах из файла.

4. Перегрузить операцию **<<** для вывода данных на экран.

**Часть 3:**

1. Перегрузить операции сравнения **<** и **==** для класса "Планета".

2. Отсортировать массив планет, используя перегруженные операции.

**Часть 4:**

1. На основе класса "Планета" выполнить индивидуальное задание с четырьмя разными характеристиками.

2. Упаковать класс в статическую библиотеку.

3. Создать пользовательский интерфейс с меню для взаимодействия с БД планет: чтение, запись, сортировка, добавление, удаление, редактирование, вывод на экран.

Каждый этап разработки требует внимательной реализации и тестирования, чтобы убедиться, что все части работают корректно и вместе составляют работоспособную программу.

**Разработка алгоритма**

Библиотека planet\_lib:

* Класс Planet:
  + Поля:
    - char \*name; - имя планеты
    - int diameter; - диаметр планеты
    - bool life; - есть ли жизнь
    - int satellites\_count; - кол-во спутников
    - static unsigned total; - общее кол-во планет
    - unsigned id = 0; - id планеты
  + Методы:
    - Planet(); - конструктор без параметров
    - Planet(const Planet&) - конструктор копирования
    - ~Planet(); - деструктор
    - Planet(const char \*, int, bool, int); - конструктор с параметрами
    - char \*getName(); - возвращает имя
    - int getDiameter(); - возвращает диаметр
    - bool getLife(); - возвращает life
    - int getSatellitesCount(); - возвращает кол-во спутников
    - void setName(const char \*); - принимает строку и записывает в name эту строку
    - void setDiameter(int); - принимает целое число и записывает в diameter это число
    - void setLife(bool); - принимает bool и записывает его в life
    - void setSatellitesCount(int); - принимает целое число и записывает в satellites\_count это число
    - void edit(); - изменяет значения полей планеты на введенные
    - bool operator < (const Planet&); - сравнивает планеты по диаметру
    - bool operator == (const Planet&); - сравнивает планеты по диаметру
    - Planet& operator = (const Planet&); - оператор присваивания
  + Статические функции:
    - static int read\_db(const char \*, Planet \*, int); - принимает имя файла, массив планет и макс. размер массива; заполняет массив планет, забирая данные из файла
    - static void print\_db(Planet \*, int); - принимает массив планет и кол-во планет; выводит информацию о планетах на экран
    - static int write\_db(const char \*, Planet \*, int); - принимает имя файла, массив планет и кол-во планет; выводит в файл данные об этих планетах, возвращает -1, если не удалось открыть файл, иначе 0
    - static int find(Planet \*, int, const char \*); - принимает массив планет, кол-во планет и название планеты, которую мы хотим найти; ищет эту планету и возвращает ее индекс, если такой планеты нет в массиве, то возвращает -1
    - static void sort\_db(Planet \*, int); - принимает массив планет и кол-во планет; сортирует массив по возрастанию диаметра
    - static void add\_planet(Planet \*, int&, Planet&); - принимает массив планет, их кол-во и новую планету; добавляет новую планету в конец массива
    - static void delete\_planet(Planet \*, int&, const char \*); - принимает массив планет, их кол-во и название планеты, которую хотим удалить; удаляет эту планету
* std::ostream& operator << (std::ostream&, Planet&); - перегрузка оператора вывода планеты
* std::istream& operator >> (std::istream&, Planet&); - перегрузка оператора ввода планеты

Библиотека goods аналогична planet\_lib

Функции в main.cpp:

* int menu() - выводит информацию о командах, считывает число с клавиатуры и возвращает его
* int demo() - демонстрационный режим
* int interactive() - интерактивный режим

**Текст программы**

Main.cpp

#include "planet\_lib/planet.h"

#include "goods/goods.h"

const int SIZE = 12;

unsigned Planet::total = 0;

unsigned Goods::total = 0;

int menu() {

int k;

std::cout << "Введите номер команды\n";

std::cout << "1 - Взять значения из базы данных\n";

std::cout << "2 - Записать значения в базу данных\n";

std::cout << "3 - Найти товар по названию\n";

std::cout << "4 - Вывести информацию о товарах на экран\n";

std::cout << "5 - Отсортировать товары по возрастанию цены\n";

std::cout << "6 - Добавить товар\n";

std::cout << "7 - Удалить товар\n";

std::cout << "8 - Изменить значения товара\n";

std::cout << "9 - Выход\n";

std::cin >> k;

return k;

}

int demo() {

Planet planets[SIZE];

char file\_name[15] = "sunsys.txt";

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

int planets\_count = Planet::read\_db(file\_name, planets, SIZE);

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

Planet::print\_db(planets, planets\_count);

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

Planet::sort\_db(planets, planets\_count);

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

Planet::print\_db(planets, planets\_count);

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

Planet::delete\_planet(planets, planets\_count, "Venus");

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

Planet::print\_db(planets, planets\_count);

std::cout << "\tadding planet 'zxc'\n";

Planet zxc;

zxc.setName("zxc");

zxc.setDiameter(123);

zxc.setLife(true);

zxc.setSatellitesCount(11);

Planet::add\_planet(planets, planets\_count, zxc);

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

Planet::print\_db(planets, planets\_count);

std::cout << "\tsearching Earth index and printing: ";

std::cout << Planet::find(planets, planets\_count, "Earth");

std::cout << "\n\twriting db to file\n";

Planet::print\_db(planets, planets\_count);

return 0;

}

int interactive() {

char file\_name[15] = "goods.txt";

Goods goods[SIZE];

int goods\_count;

int ind;

while (true) {

switch (menu()) {

case 1:

goods\_count = Goods::read\_db(file\_name, goods, SIZE);

break;

case 2:

if (Goods::write\_db(file\_name, goods, goods\_count)) {

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

}

break;

case 3:

if ((ind = Goods::find(goods, goods\_count)) >= 0)

std::cout << "Искомая планета имеет индекс " << ind << '\n';

else

std::cout << "Такой планеты нет" << std::endl;

break;

case 4:

Goods::print\_db(goods, goods\_count);

break;

case 5:

Goods::sort\_db(goods, goods\_count);

break;

case 6:

Goods::add\_goods(goods, goods\_count);

break;

case 7:

Goods::delete\_goods(goods, goods\_count);

break;

case 8:

if ((ind = Goods::find(goods, goods\_count)) >= 0)

goods[ind].edit();

else

std::cout << "Такой планеты нет" << std::endl;

break;

case 9:

return 0;

default:

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

break;

}

}

}

int main(int argc, char \*\*argv) {

if (argc != 2) return 0;

if (!strcmp(argv[1], "demo")) {

demo();

}

else if (!strcmp(argv[1], "int")) {

interactive();

}

return 0;

}

CMakeLists.txt проекта

cmake\_minimum\_required(VERSION 3.23)

set(project "lab02")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(${project}\_SOURCES

main.cpp)

add\_subdirectory(planet\_lib)

add\_subdirectory(goods)

set(${project}\_SOURCE\_LIST

${${project}\_SOURCES})

add\_executable(${project}

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

target\_link\_libraries(${project} planet\_lib)

target\_link\_libraries(${project} goods)

configure\_file(

${CMAKE\_CURRENT\_SOURCE\_DIR}/sunsys.txt

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

configure\_file(

${CMAKE\_CURRENT\_SOURCE\_DIR}/goods.txt

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

planet.h

#pragma once

#include <iostream>

#include <fstream>

class Planet {

char \*name;

int diameter;

bool life;

int satellites\_count;

static unsigned total;

unsigned id = 0;

public:

Planet();

Planet(const Planet&);

~Planet();

Planet(const char \*, int, bool, int);

char \*getName();

int getDiameter();

bool getLife();

int getSatellitesCount();

void setName(const char \*);

void setDiameter(int);

void setLife(bool);

void setSatellitesCount(int);

void edit();

bool operator < (const Planet&);

bool operator == (const Planet&);

Planet& operator = (const Planet&);

static int read\_db(const char \*, Planet \*, int);

static void print\_db(Planet \*, int);

static int write\_db(const char \*, Planet \*, int);

static int find(Planet \*, int, const char \*);

static void sort\_db(Planet \*, int);

static void add\_planet(Planet \*, int&, const Planet&);

static void delete\_planet(Planet \*, int&, const char \*);

};

std::ostream& operator << (std::ostream&, Planet&);

std::istream& operator >> (std::istream&, Planet&);

planet.cpp

#include "planet.h"

Planet::Planet() {

name = new char[1];

name[0] = '\0';

diameter = -1;

life = false;

satellites\_count = -1;

total++;

id = total;

*//std::cout << "Created id: " << id << '\n';*

}

Planet::Planet(const Planet &pl) {

name = new char[strlen(pl.name) + 1];

strcpy(name, pl.name);

diameter = pl.diameter;

life = pl.life;

satellites\_count = pl.satellites\_count;

total++;

id = total;

*//std::cout << "Created id: " << id << '\n';*

}

Planet::~Planet() {

delete[] name;

total--;

*//std::cout << "Deleted id: " << id << '\n';*

}

Planet::Planet(const char \*\_name, int \_diameter, bool \_life, int \_satellites\_count) {

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

strcpy(name, \_name);

diameter = \_diameter;

life = \_life;

satellites\_count = \_satellites\_count;

total++;

id = total;

*//std::cout << "Created id: " << id << '\n';*

}

char \*Planet::getName() {

return name;

}

int Planet::getDiameter() {

return diameter;

}

bool Planet::getLife() {

return int(life);

}

int Planet::getSatellitesCount() {

return satellites\_count;

}

void Planet::setName(const char \*\_name) {

delete[] name;

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

strcpy(name, \_name);

}

void Planet::setDiameter(int \_diameter) {

diameter = \_diameter;

}

void Planet::setLife(bool \_life) {

life = \_life;

}

void Planet::setSatellitesCount(int \_satellites\_count) {

satellites\_count = \_satellites\_count;

}

void Planet::edit() {

std::cout << "Введите новые значения для данной планеты\n";

std::cin >> \*this;

}

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

out << planet.getName() << ' ' << planet.getDiameter() << ' ' << planet.getLife() << ' ' << planet.getSatellitesCount();

return out;

}

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

char \*name = new char[15];

int diameter;

bool life;

int satellites\_count;

in >> name >> diameter >> life >> satellites\_count;

planet.setName(name);

planet.setDiameter(diameter);

planet.setLife(life);

planet.setSatellitesCount(satellites\_count);

return in;

}

bool Planet::operator < (const Planet &pl) {

return diameter < pl.diameter;

}

bool Planet::operator == (const Planet &pl) {

return ((!strcmp(name, pl.name)) && (diameter == pl.diameter) && (life == pl.life) && (satellites\_count == pl.satellites\_count));

}

Planet& Planet::operator = (const Planet &pl) {

delete[] name;

name = new char[strlen(pl.name) + 1];

strcpy(name, pl.name);

diameter = pl.diameter;

life = pl.life;

satellites\_count = pl.satellites\_count;

return \*this;

}

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

std::ifstream fin(file\_name);

int i = 0;

while (fin >> planets[i++]);

return i - 1;

}

void Planet::print\_db(Planet \*planets, int planets\_count) {

for (int i = 0; i < planets\_count; i++) {

std::cout << planets[i] << '\n';

}

}

int Planet::write\_db(const char \*file\_name, Planet \*planets, int planets\_count) {

std::ofstream fout(file\_name);

if (!fout) return -1;

for (int i = 0; i < planets\_count; i++) {

fout << planets[i] << '\n';

}

return 0;

}

int Planet::find(Planet \*planets, int planets\_count, const char \*search\_name) {

for (int i = 0; i < planets\_count; i++) {

if (!strcmp(search\_name, planets[i].getName())) {

return i;

}

}

return -1;

}

void Planet::sort\_db(Planet \*planets, int planets\_count) {

for (int i = 0; i < planets\_count; i++) {

for (int j = i + 1; j < planets\_count; j++) {

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

Planet tmp(planets[i]);

planets[i] = planets[j];

planets[j] = tmp;

}

}

}

}

void Planet::add\_planet(Planet \*planets, int &planets\_count, const Planet &new\_planet) {

planets[planets\_count++] = new\_planet;

}

void Planet::delete\_planet(Planet \*planets, int &planets\_count, const char \*name\_to\_delete) {

int ind = Planet::find(planets, planets\_count, name\_to\_delete);

if (ind == -1) {

std::cout << "Такой планеты нет\n";

return;

}

planets\_count-–;

for (int i = ind; i < planets\_count; i++) {

planets[i] = planets[i + 1];

}

}

CMakeLists.txt библиотеки planet\_lib

cmake\_minimum\_required(VERSION 3.23)

set(project "planet\_lib")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(${project}\_SOURCES

planet.cpp)

set(${project}\_HEADERS

planet.h)

set(${project}\_SOURCE\_LIST

${${project}\_SOURCES}

${${project}\_HEADERS})

add\_library(${project}

STATIC

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

goods.h

#pragma once

#include <iostream>

#include <fstream>

class Goods {

char \*name;

int price;

int count\_in\_stock;

char \*category;

static unsigned total;

unsigned id = 0;

public:

Goods();

Goods(const Goods&);

~Goods();

Goods(const char \*, int, int, char \*);

char \*getName();

int getPrice();

int getCountInStock();

char \*getCategory();

void setName(const char \*);

void setPrice(int);

void setCountInStock(int);

void setCategory(const char \*);

void edit();

bool operator < (const Goods&);

bool operator == (const Goods&);

Goods& operator = (const Goods&);

static int read\_db(const char \*, Goods \*, int);

static void print\_db(Goods \*, int);

static int write\_db(const char \*, Goods \*, int);

static int find(Goods \*, int);

static void sort\_db(Goods \*, int);

static void add\_goods(Goods \*, int&);

static void delete\_goods(Goods \*, int&);

};

std::ostream& operator << (std::ostream& out, Goods &planet);

std::istream& operator >> (std::istream& out, Goods &planet);

goods.cpp

#include "goods.h"

Goods::Goods() {

name = new char[1];

name[0] = '\0';

price = 0;

count\_in\_stock = 0;

category = new char[1];

category[0] = '\0';

total++;

id = total;

*//std::cout << "Created id: " << id << '\n';*

}

Goods::Goods(const Goods &gd) {

name = new char[strlen(gd.name) + 1];

strcpy(name, gd.name);

price = gd.price;

count\_in\_stock = gd.count\_in\_stock;

category = new char[strlen(gd.category) + 1];

strcpy(category, gd.category);

total++;

id = total;

*//std::cout << "Created id: " << id << '\n';*

}

Goods::~Goods() {

delete[] name;

delete[] category;

total--;

*//std::cout << "Deleted id: " << id << '\n';*

}

Goods::Goods(const char \*\_name, int \_price, int \_count\_in\_stock, char \*\_category) {

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

strcpy(name, \_name);

price = \_price;

count\_in\_stock = \_count\_in\_stock;

category = new char[strlen(\_category) + 1];

strcpy(category, \_category);

total++;

id = total;

*//std::cout << "Created id: " << id << '\n';*

}

char \*Goods::getName() {

return name;

}

int Goods::getPrice() {

return price;

}

int Goods::getCountInStock() {

return count\_in\_stock;

}

char \*Goods::getCategory() {

return category;

}

void Goods::setName(const char \*\_name) {

delete[] name;

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

strcpy(name, \_name);

}

void Goods::setPrice(int \_price) {

price = \_price;

}

void Goods::setCountInStock(int \_count\_in\_stock) {

count\_in\_stock = \_count\_in\_stock;

}

void Goods::setCategory(const char \*\_category) {

delete[] category;

category = new char[strlen(\_category) + 1];

strcpy(category, \_category);

}

void Goods::edit() {

std::cout << "Введите новые значения для данного товара\n";

std::cin >> (\*this);

}

std::ostream& operator << (std::ostream& out, Goods &gd) {

out << gd.getName() << ' ' << gd.getPrice() << ' ' << gd.getCountInStock() << ' ' << gd.getCategory();

return out;

}

std::istream& operator >> (std::istream& in, Goods &gd) {

char name[20];

int price, count\_in\_stock;

char \*category = new char[30];

in >> name >> price >> count\_in\_stock >> category;

gd.setName(name);

gd.setPrice(price);

gd.setCountInStock(count\_in\_stock);

gd.setCategory(category);

return in;

}

bool Goods::operator < (const Goods &gd) {

return price < gd.price;

}

bool Goods::operator == (const Goods &gd) {

return ((!strcmp(name, gd.name)) && (price == gd.price) && (count\_in\_stock == gd.count\_in\_stock) && (!strcmp(category, gd.category)));

}

Goods& Goods::operator = (const Goods &gd) {

delete[] name;

name = new char[strlen(gd.name) + 1];

strcpy(name, gd.name);

price = gd.price;

count\_in\_stock = gd.count\_in\_stock;

delete[] category;

category = new char[strlen(gd.category) + 1];

strcpy(category, gd.category);

return \*this;

}

int Goods::read\_db(const char \*file\_name, Goods \*goods, int size) {

std::ifstream fin(file\_name);

int i = 0;

while (fin >> goods[i++]);

return i - 1;

}

void Goods::print\_db(Goods \*goods, int goods\_count) {

for (int i = 0; i < goods\_count; i++) {

std::cout << goods[i] << '\n';

}

}

int Goods::write\_db(const char \*file\_name, Goods \*goods, int goods\_count) {

std::ofstream fout(file\_name);

if (!fout) return -1;

for (int i = 0; i < goods\_count; i++) {

fout << goods[i] << '\n';

}

return 0;

}

int Goods::find(Goods \*goods, int goods\_count) {

char search\_name[15];

std::cout << "Введите название товара, который хотите найти/удалить: \n";

std::cin >> search\_name;

for (int i = 0; i < goods\_count; i++) {

if (!strcmp(search\_name, goods[i].getName())) {

return i;

}

}

return -1;

}

void Goods::sort\_db(Goods \*goods, int goods\_count) {

for (int i = 0; i < goods\_count; i++) {

for (int j = i + 1; j < goods\_count; j++) {

if (!(goods[i] < goods[j])) {

Goods tmp(goods[i]);

goods[i] = goods[j];

goods[j] = tmp;

}

}

}

}

void Goods::add\_goods(Goods \*goods, int &goods\_count) {

std::cout << "Введите данные нового товара\n";

Goods gd;

std::cin >> gd;

goods[goods\_count++] = gd;

}

void Goods::delete\_goods(Goods \*goods, int &goods\_count) {

int ind = Goods::find(goods, goods\_count);

if (ind == -1) {

std::cout << "Такого товара нет\n";

return;

}

goods\_count--;

for (int i = ind; i < goods\_count; i++) {

goods[i] = goods[i + 1];

}

}

CMakeLists.txt библиотеки goods

cmake\_minimum\_required(VERSION 3.23)

set(project "goods")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(${project}\_SOURCES

goods.cpp)

set(${project}\_HEADERS

goods.h)

set(${project}\_SOURCE\_LIST

${${project}\_SOURCES}

${${project}\_HEADERS})

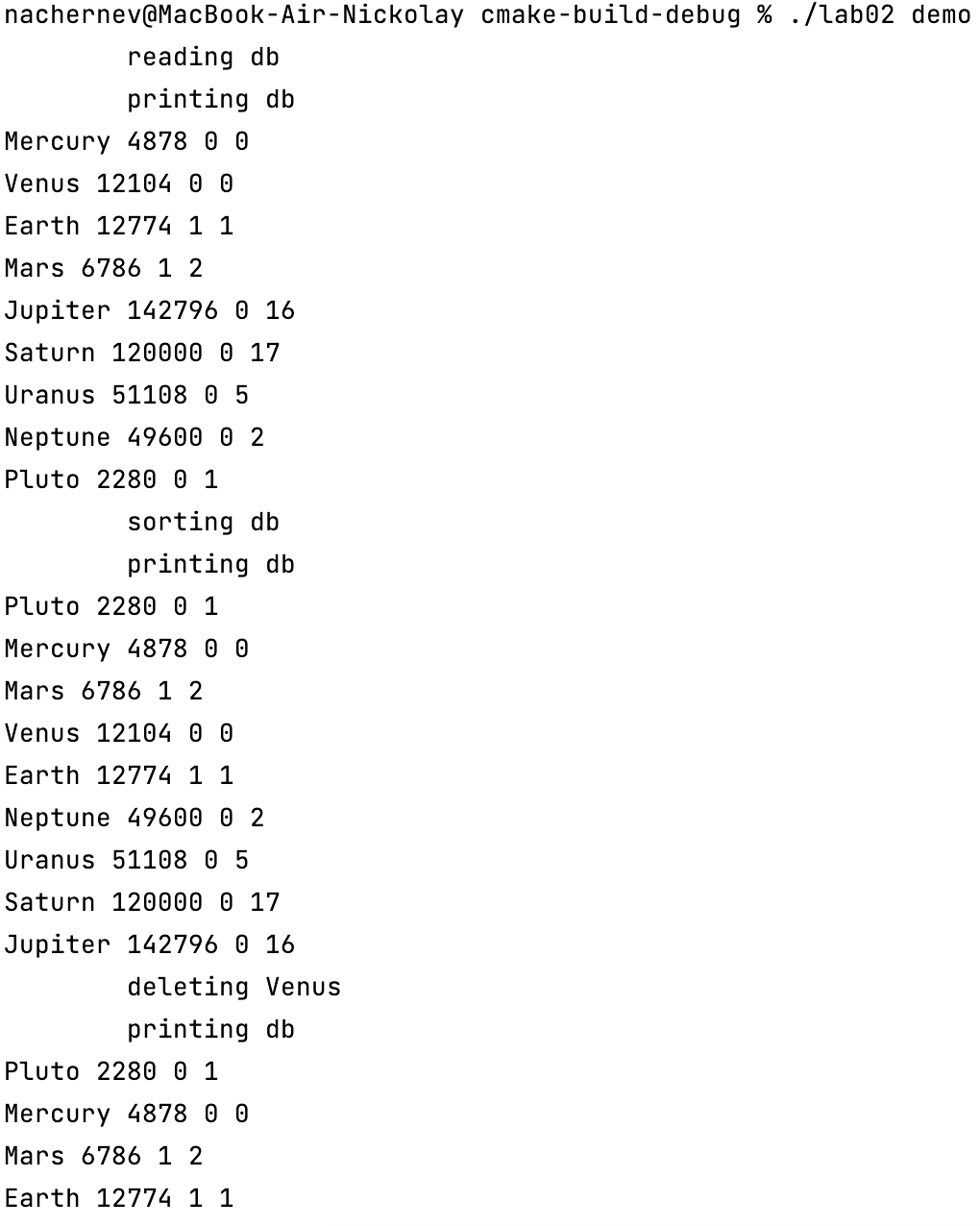
add\_library(${project}

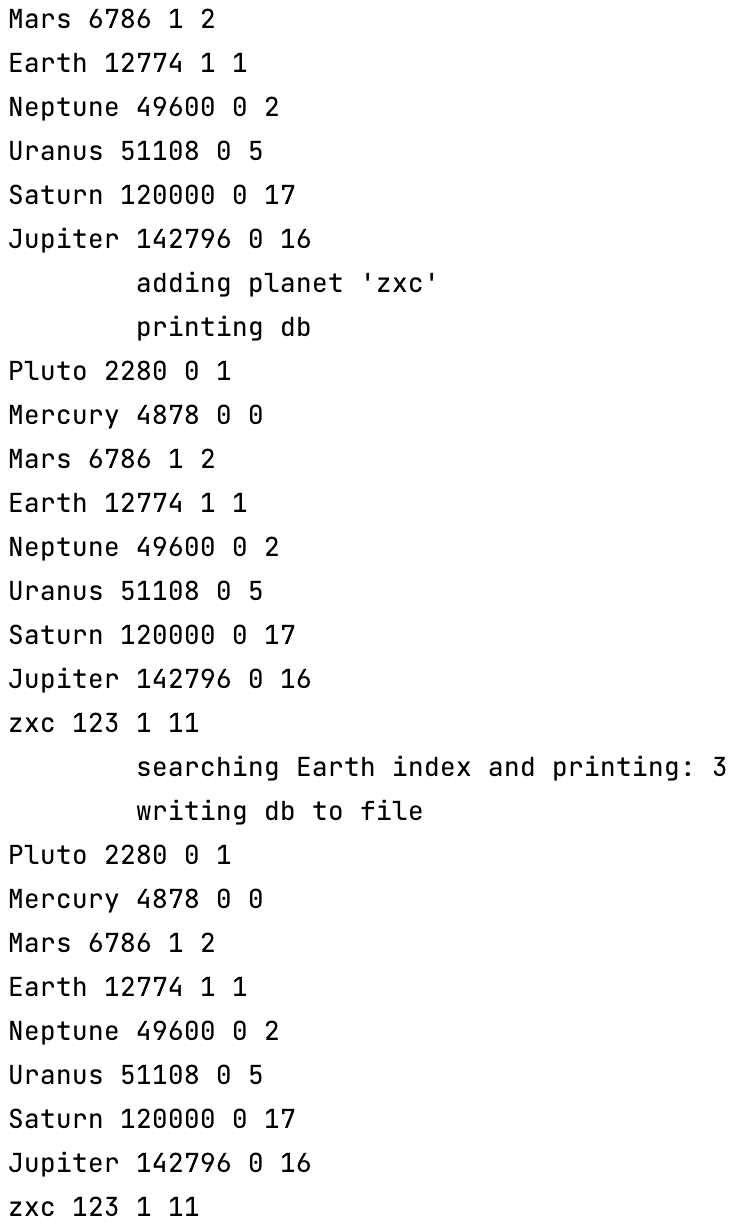
STATIC

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

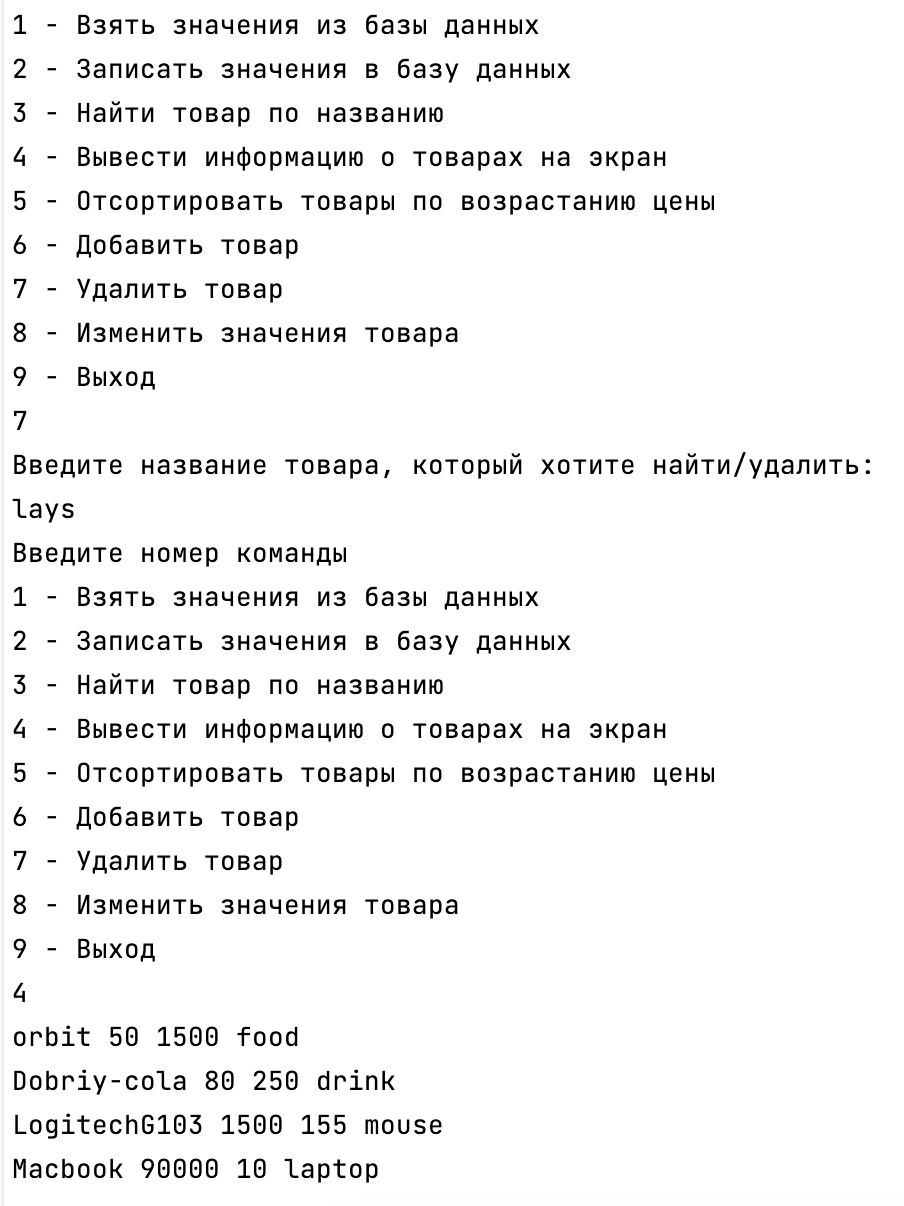
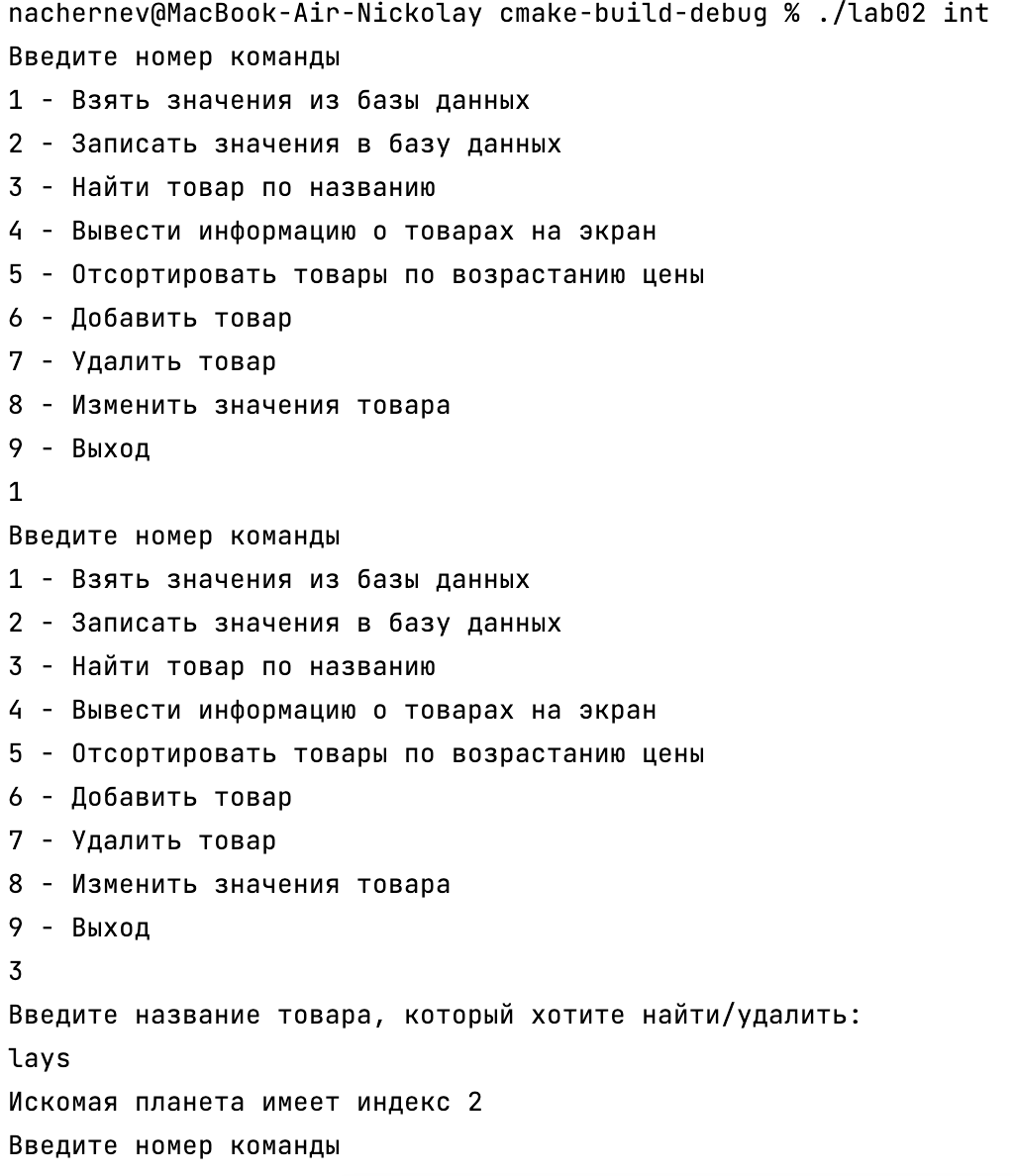
**Анализ результатов**

Демонстрационный режим:

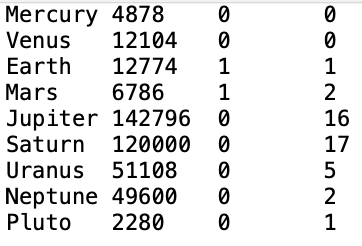




Интерактивный режим:



sunsys.txt



goods.txt

