**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №2

«Создание и использование статической библиотеки классов»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-21 |  | преподаватель каф. ИУ5 |
| Агличеев Михаил |  | Кротов Ю.Н. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020г.

1. Постановка задачи

**Задание.**

**1.1**

* 1. Ознакомиться с приведенным ниже материалом (Введение и Пример 1). Перейти в примере 1 от структуры к классу.
  2. Разработать класс «Планета» для планет солнечной системы (4 характеристики планет разного типа приведены в Приложении 1). Имя планеты должно иметь тип char\*.
  3. Создать много-файловый проект и отладить программу, которая создает один объект класса «Планета» и выводит значения его полей на экран. (Пример программы приведен в Приложении 3).
  4. Организовать интерфейс пользователя с программой в виде меню, позволяющего выполнять следующие действия:

- чтение БД «Солнечная система» из файла;

- запись БД «Солнечная система» в файл;

- сортировка БД;

- редактирование БД;

- вывод БД на экран.

* 1. Создать текстовый файл (в блокноте) с данными о планетах солнечной системы и сохранить его в папке проекта. Первоначально, для отладки, введите две записи.
  2. Добавить в программу ввод – вывод объекта класса «Планета» в текстовый файл.

**1.2**

1. Ознакомиться с содержанием раздела «Перегрузка операций» и выполнить приведенный в нем пример 2.

2. Перегрузить конструктор копирования, деструктор и оператор присваивания.

3. Вставьте в конструкторы и деструктор печать типа «Создание (Удаление) ID ***n*** » , где ***n*** - номер объекта, для которого они вызываются. (Для реализации этого пункта задания разберите и выполните пример из Приложения 2.).

4. Перегрузить операцию “ >> “ для класса «Планета» и ***ifstream*** и прочитать данные о планетах из файла в массив «Солнечная система» из объектов класса «Планета».

5. Перегрузить операцию “ << “ для классов «Планета» и ***ofstream*** и вывести на экран данные из массива.

**1.3**

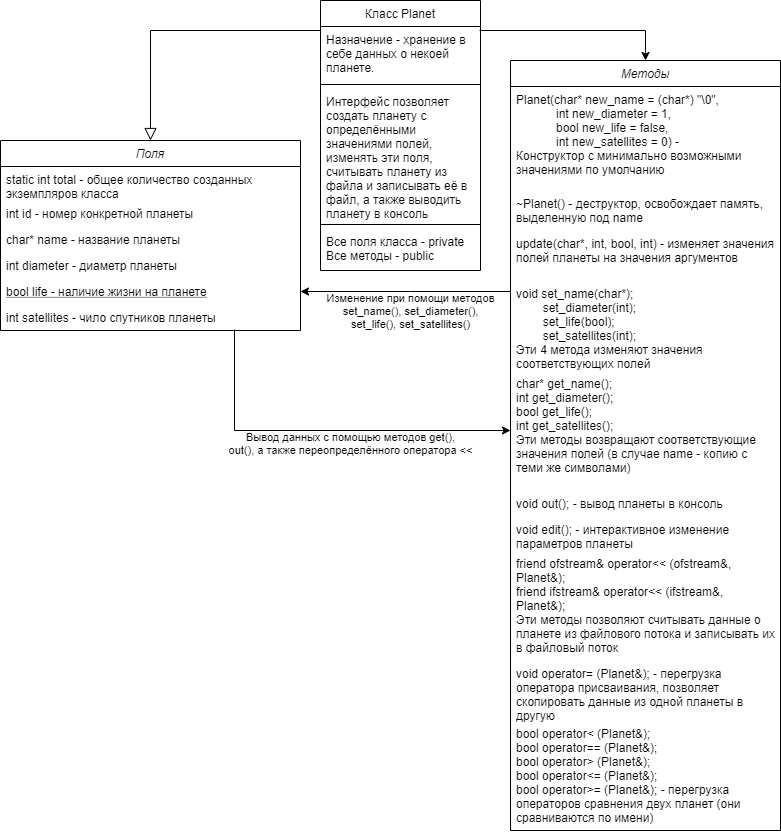
1. Перегрузить операции сравнения “ < и ==” для класса «Планета», использовав для этого значение одного из полей.

2. Отсортировать массив планет солнечной системы, хранящийся в файле, с использованием перегруженных операций.

**1.4**

На основе разработанного класса Планета выполнить задание по варианту (не менее 4 характеристик в классе разного типа).

2. Разработка интерфейса класса



3. Текст программы

Planet.h

#ifndef \_\_PLANET\_H\_\_

#define \_\_PLANET\_H\_\_

#include <fstream>

char\* cut(char\*);

class Planet {

private:

static int total;

int id;

char\* name;

int diameter;

bool life;

int satellites;

public:

Planet(char\* new\_name = (char\*)"\0", int new\_diameter = 1,

bool new\_life = false, int new\_satellites = 0);

~Planet();

void update(char\*, int, bool, int);

void set\_name(char\*);

void set\_diameter(int);

void set\_life(bool);

void set\_satellites(int);

char\* get\_name();

int get\_diameter();

bool get\_life();

int get\_satellites();

void out();

void edit();

friend ofstream& operator<< (ofstream&, Planet&);

friend ifstream& operator>> (ifstream&, Planet&);

void operator= (Planet&);

bool operator< (Planet&);

bool operator== (Planet&);

bool operator> (Planet&);

bool operator<= (Planet&);

bool operator>= (Planet&);

};

#endif //\_\_PLANET\_H\_\_

Planet.cpp

#include <iostream>

#include "Planet.h"

int Planet::total = 0;

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

diameter = 1;

satellites = 0;

total++;

id = total;

cout << "Создание ID " << id << endl;

update(new\_name, new\_diameter, new\_life, new\_satellites);

}

Planet::~Planet() {

total--;

cout << "Удаление ID " << id << endl;

if (!name) {

delete[] name;

}

}

void Planet::update(char\* new\_name, int new\_diameter, bool new\_life, int new\_satellites) {

set\_name(new\_name);

set\_diameter(new\_diameter);

set\_life(new\_life);

set\_satellites(new\_satellites);

}

void Planet::set\_name(char new\_name[]) {

if (!name) {

delete[] name;

}

name = cut(new\_name);

}

void Planet::set\_diameter(int new\_diameter) {

if (new\_diameter > 0) {

diameter = new\_diameter;

}

else {

cout << "Некорректный диаметр: " << new\_diameter

<< " Сохраняется прежнее значение (1 по умолчанию)" << endl;

}

}

void Planet::set\_life(bool new\_life) {

life = new\_life;

}

void Planet::set\_satellites(int new\_satellites) {

if (new\_satellites >= 0) {

satellites = new\_satellites;

}

else {

cout << "Некорректное число спутников: " << new\_satellites

<< " Сохраняется прежнее значение (1 по умолчанию)" << endl;

}

}

void Planet::edit() {

const int name\_len = 100;

int answer;

cout << "Что бы вы хотели изменить?" << endl

<< "Название: 1" << endl

<< "Диаметр: 2" << endl

<< "Жизнь: 3" << endl

<< "Количество спутников: 4" << endl

<< "Введите номер желаемой опции: ";

cin >> answer;

cout << endl;

switch (answer) {

case 1:

char new\_name[name\_len];

cout << "Введите новое название: ";

cin >> new\_name;

cout << endl;

set\_name(new\_name);

break;

case 2:

int new\_diameter;

cout << "Введите новый диаметр: ";

cin >> new\_diameter;

cout << endl;

set\_diameter(new\_diameter);

break;

case 3:

bool new\_life;

cout << "Введите новый статус жизни (0, если жизни нет и 1, если есть): ";

cin >> new\_life;

cout << endl;

set\_life(new\_life);

break;

case 4:

int new\_satellites;

cout << "Введите новое количество спутников: ";

cin >> new\_satellites;

cout << endl;

set\_satellites(new\_satellites);

break;

default:

cout << "Такой опции нет! Планета останется неизменной" << endl;

}

}

void Planet::out() {

cout << "Planet: " << name << endl

<< "Diameter: " << diameter << endl

<< "Has it life?: " << life << endl

<< "Sattelites count: " << satellites << endl << endl;

}

ofstream& operator<< (ofstream& out, Planet& planet) {

out << planet.name << " "

<< planet.diameter << " "

<< planet.life << " "

<< planet.satellites << endl;

return out;

}

ifstream& operator>> (ifstream& in, Planet& planet) {

const int name\_len = 100;

char new\_name[name\_len];

int new\_diameter, new\_life, new\_satellites;

in >> new\_name >> new\_diameter >> new\_life >> new\_satellites;

planet.update(new\_name, new\_diameter, new\_life, new\_satellites);

return in;

}

void Planet::operator= (Planet& planet) {

if (!name) {

delete[] name;

}

name = cut(planet.name);

diameter = planet.diameter;

life = planet.life;

satellites = planet.satellites;

id = planet.id;

}

bool Planet::operator< (Planet& other) {

int i;

char\* other\_name = other.name;

for (i = 0; name[i] && other\_name[i] && name[i] == other\_name[i]; i++);

return name[i] < other\_name[i];

}

bool Planet::operator== (Planet& other) {

int i;

char\* other\_name = other.name;

for (i = 0; name[i] && other\_name[i] && name[i] == other\_name[i]; i++);

return name[i] == other\_name[i];

}

bool Planet::operator> (Planet& other) {

int i;

char\* other\_name = other.name;

for (i = 0; name[i] && other\_name[i] && name[i] == other\_name[i]; i++);

return name[i] > other\_name[i];

}

bool Planet::operator<= (Planet& other) {

int i;

char\* other\_name = other.name;

for (i = 0; name[i] && other\_name[i] && name[i] <= other\_name[i]; i++);

return name[i] <= other\_name[i];

}

bool Planet::operator>= (Planet& other) {

int i;

char\* other\_name = other.name;

for (i = 0; name[i] && other\_name[i] && name[i] >= other\_name[i]; i++);

return name[i] >= other\_name[i];

}

char\* Planet::get\_name() {

return cut(name);

}

int Planet::get\_diameter() {

return diameter;

}

bool Planet::get\_life() {

return life;

}

int Planet::get\_satellites() {

return satellites;

}

char\* cut(char\* old\_str) {

char\* new\_str = new char[strlen(old\_str) + 1];

int i;

for (i = 0; old\_str[i]; i++)

{

new\_str[i] = old\_str[i];

}

new\_str[i] = '\0';

return new\_str;

}

PlanetTesting.h

#ifndef \_\_PLANETTESTING\_H\_\_

#define \_\_PLANETTESTING\_H\_\_

#include "Planet.h"

using namespace std;

int main();

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

void write\_db(char\*,Planet\*, int);

int find(Planet\*,int);

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

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

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

int menu();

void qsort(Planet\*, int, int);

void swap(Planet&, Planet&);

#endif

PlanetTesting.cpp

#include <iostream>

#include <iomanip>

#include "PlanetTesting.h"

#include "windows.h"

const int Size = 12;

int main() {

setlocale(LC\_CTYPE, ".1251");

char\* file\_name = (char \*)"database.txt";

Planet solar\_system[Size];

int n\_planet = 0;

int ind;

while (true) {

switch (menu())

{

case 1:

n\_planet = read\_db(file\_name, solar\_system, Size);

break;

case 2:

write\_db(file\_name, solar\_system, n\_planet);

break;

case 3:

if ((ind = find(solar\_system, n\_planet)) >= 0) {

cout << "Такая планета есть" << endl;

solar\_system[ind].edit();

}

else {

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

}

break;

case 4:

if ((ind = find(solar\_system, n\_planet)) >= 0) {

cout << "Такая планета есть и будет удалена" << endl;

delete\_planet(solar\_system, ind, n\_planet);

cout << endl;

}

else {

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

}

break;

case 5:

add\_planet(solar\_system, n\_planet, Size);

cout << endl;

break;

case 6:

print\_db(solar\_system, n\_planet);

break;

case 7:

qsort(solar\_system, 0, n\_planet - 1);

break;

case 8:

return 0;

default:

cout << " Неправильный ввод" << endl;

break;

}

}

return 0;

}

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

int n\_planet;

int i;

ifstream infile(file\_name);

if (infile.fail()) {

cout << "Ошибка при чтении файла " << file\_name << endl;

system("pause");

infile.clear();

infile.close();

return 0;

}

infile >> n\_planet;

if (n\_planet > planets\_size) {

n\_planet = planets\_size - 1;

}

for (i = 0; i < n\_planet; i++) {

infile >> planets[i];

}

infile.close();

return n\_planet;

}

void write\_db(char\* file\_name, Planet\* planets, int n\_planet) {

ofstream outfile(file\_name);

int i;

outfile << n\_planet << endl;

for (i = 0; i < n\_planet; i++) {

outfile << planets[i];

}

outfile.close();

}

int find(Planet\* planets, int n\_planet) {

int i, a;

const int word\_len = 100;

char name[word\_len], \* planet\_name;

cout << "Введите имя планеты: ";

cin >> name;

for (i = 0; i < n\_planet; i++) {

planet\_name = planets[i].get\_name();

for (a = 0; planet\_name[a] && name[a] && planet\_name[a] == name[a]; a++);

if (planet\_name[a] == name[a]) {

return i;

}

}

return -1;

}

void delete\_planet(Planet\* planets, int index, int& n\_planet) {

planets[index] = planets[n\_planet - 1];

planets[n\_planet - 1].update((char\*)"", 1, 0, 0);

n\_planet--;

}

void add\_planet(Planet\* planets, int& n\_planet, const int planets\_size) {

const int name\_len = 100;

char new\_name[name\_len];

int new\_diameter, new\_life, new\_satellites;

if (n\_planet == planets\_size) {

cout << "Невозможно добавить планету - массив заполнен" << endl;

}

else {

cout << "Введите название, диаметр, наличие жизни (0 либо 1), количество её спутников через пробел" << endl;

cin >> new\_name >> new\_diameter >> new\_life >> new\_satellites;

cout << endl;

planets[n\_planet].update(new\_name, new\_diameter, new\_life, new\_satellites);

n\_planet++;

}

}

void print\_db(Planet\* planets, int n\_planet) {

cout << setw(20) << "Название планеты"

<< setw(15) << "Диаметр"

<< setw(15) << "Наличие жизни"

<< setw(25) << "Количество спутников" << endl;

for (int i = 0; i < n\_planet; i++) {

cout << setw(20) << left << planets[i].get\_name()

<< setw(15) << right << planets[i].get\_diameter()

<< setw(15) << planets[i].get\_life()

<< setw(25) << planets[i].get\_satellites() << endl;

}

}

int menu() {

int result;

cout << " Меню:" << endl

<< "Чтение БД из файла: 1" << endl

<< "Запись БД в файл: 2" << endl

<< "Изменение существующей планеты: 3" << endl

<< "Удаление планеты: 4" << endl

<< "Добавление планеты: 5" << endl

<< "Печать БД: 6" << endl

<< "Сортировка БД: 7" << endl

<< "Выход: 8" << endl

<< "Введите номер желаемой опции: ";

cin >> result;

cout << endl;

return result;

}

void qsort(Planet\* arr, int start, int end) { // Quick sort from start to end including both.

Planet\* pivot = arr + end; // Element to be placed at the right position

int i = start - 1, j;

for (j = start; j < end; j++) {

if (arr[j] < \*pivot) {

i++;

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[end]);

if (i > start) {

qsort(arr, start, i); // Sorting "less than pivot" and "more than pivot" arrays.

}

if (end > i + 2) {

qsort(arr, i + 2, end);

}

}

void swap(Planet& first, Planet& second) {

Planet temp;

temp = first;

first = second;

second = temp;

}

4. Анализ результатов

Содержимое файла database.txt до изменения:

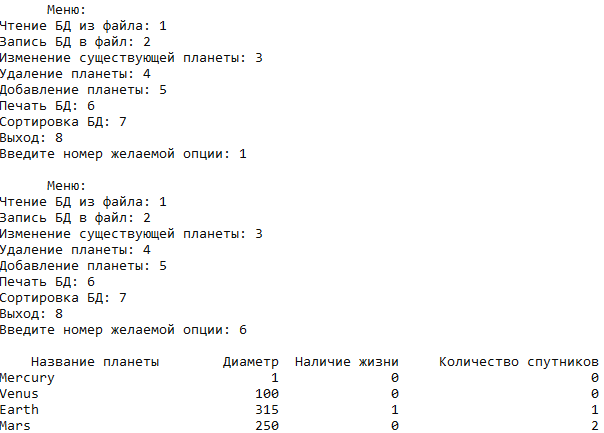
4

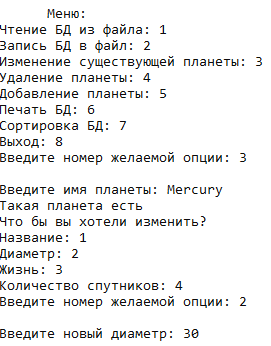
Mercury 1 0 0

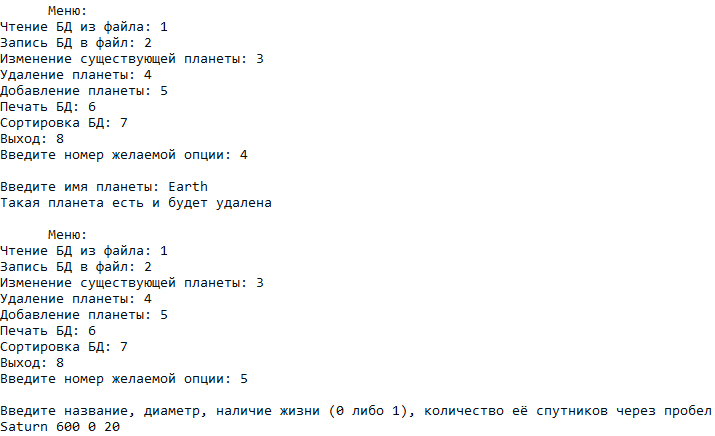
Venus 100 0 0

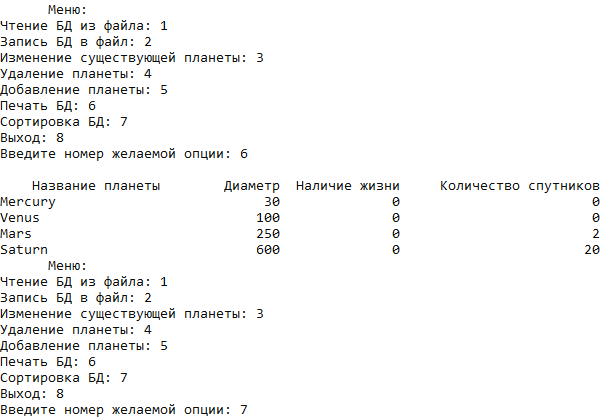
Earth 315 1 1

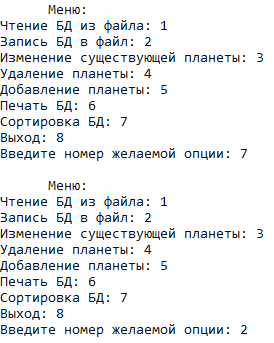
Mars 250 0 2

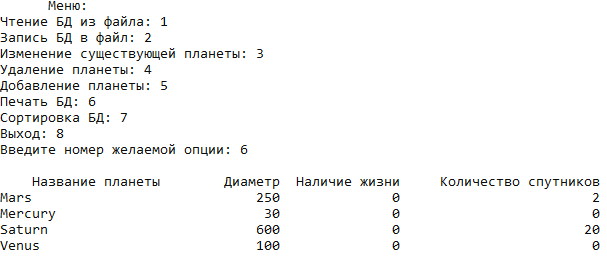












Содержимое файла database.txt после изменения:

4

Mars 250 0 2

Mercury 30 0 0

Saturn 600 0 20

Venus 100 0 0