**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 |
| Байдаков В.М. |  |  |
| Подпись и дата: |  | 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

Члены класса:

char\* planetName – название планеты

unsigned int Diameter – диаметр планеты

bool haveLife – присутствие жизни

unsigned int satellites – количество спутников

static unsigned total – количество инициализированных планет

unsigned id – id планеты

Методы класса:

void SetPlanetName(char \*planetNameToSet) – сеттер имени планеты

void SetDiameter(unsigned int DiameterToSet) – сеттер диаметра

void SetHaveLife(bool haveLifeValue) – сеттер присутствия жизни

void SetSatellites(unsigned int satellitesToSet) – сеттер количества спутников

char \*GetPlanetName() – геттер имени планеты

unsigned int GetDiameter() – геттер диаметра

bool GetHaveLife() – геттер присутствия жизни

unsigned int GetSatellites() – геттер количества спутников

unsigned GetID() – геттер idшника

Planet() – конструктор без параметров

Planet(char \*p, unsigned d = 0, bool h = 0, unsigned s = 0) – геттер с параметрами

~Planet() - деструктор

void operator=(Planet &planet\_) – перегрузка оператор =

bool operator==(Planet &planetForEqual) – перегрузка оператора ==

bool operator<(Planet &planetForCompare) – перегрузка оператора <

Planet(Planet &planetToCopy) – перегрзка конструктора копирования

static void AddPlanet(Planet \*db, size\_t &size, Planet &planetToAdd) – фуннкция добавляет планету в массив db

static void PrintDB(Planet \*mas, size\_t size) – выводит массив db в консоль

static void DBtoFile(Planet \*mas, size\_t size, const char \*file1) – выводит массив db в файл

static void Sort(Planet \*mas, size\_t size) – сортирует массив db

static unsigned FindID(char \*planetToFind, Planet \*mas, size\_t size) – находит id планеты по ее имени

static void DeletePlanet(Planet \*mas, size\_t &size, char \*planetToDelete) – удаляет планету из массива db

static void EditPlanetValues(Planet \*mas, size\_t size) – редактирует значения планеты.

Прочее:

std::istream &operator>>(std::istream &in, Planet &planet) – перегрузка оператора >> для ввода значений класса из потока in

std::ostream &operator<<(std::ostream &out, Planet &planet) – перегрузка оператора << для вывода значений класса в поток out

unsigned menu() – меню для интерактивного режима

Библиотека bus аналогична библиотеке planet

Функции в main.cpp:

int interactive() – интерактивный режим выполнения программы для класса по варианту (в данном случае – автобус в автопарке)

int demo() – демонстрационный режим выполнения программы для класса планеты

Текст программы

Main.cpp

#include "planet.h"

#include "bus.h"

int interactive()

{

Bus db[15];

size\_t size = 0;

std::fstream file("busdata.txt");

while (true)

{

switch (menuBus())

{

case 1:

while (file)

{

file >> db[size++];

}

size--;

break;

case 2:

Bus::DBtoFile(db, size, "busdata.txt");

break;

case 3:

Bus::EditBusValues(db, size);

break;

case 4:

Bus::PrintDB(db, size);

break;

case 5:

Bus::Sort(db, size);

break;

case 6:

Bus::AddBus(db, size);

break;

case 7:

Bus::DeleteBus(db, size);

break;

case 8:

return 0;

break;

default:

std::cout << "\nWrong input, try again)\n";

break;

}

}

return 0;

}

int demo()

{

Planet db[15];

size\_t size = 0;

std::fstream file("data.txt");

while (file)

{

file >> db[size++];

}

size--;

std::cout << "\nprinting database:\n";

Planet::PrintDB(db, size);

std::cout << "\nsorting database:\n";

Planet::Sort(db, size);

std::cout << "\nprinting database:\n";

Planet::PrintDB(db, size);

std::cout << "\nDeleting mercury...\n";

char planetName[30];

strcpy(planetName, "Mercury");

Planet::DeletePlanet(db, size, planetName);

std::cout << "\nprinting database:\n";

Planet::PrintDB(db, size);

std::cout << "\nadding moon with 780 as diameter, false value of having life and 10 satellites :) :\n";

Planet moon;

strcpy(planetName, "Moon");

moon.SetPlanetName(planetName);

moon.SetDiameter(780);

moon.SetHaveLife(0);

moon.SetSatellites(10);

Planet::AddPlanet(db, size, moon);

std::cout << "\nprinting database:\n";

Planet::PrintDB(db, size);

std::cout << "\nmerging to file:\n";

Planet::DBtoFile(db, size, "data.txt");

return 0;

}

int main(int argc, char \*argv[])

{

system("chcp 1251");

bool isInteractive = false;

if ((argc == 2) && strcmp(argv[1], "i") == 0)

{

isInteractive = true;

}

if (isInteractive)

{

return interactive();

}

else

{

return demo();

}

return 0;

}

CMakeLists.txt лабораторной работы(основной, для main.cpp)

cmake\_minimum\_required(VERSION 3.23)

set(project "lab2")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

add\_subdirectory(buslib)

add\_subdirectory(planet)

set(SOURCES

main.cpp

)

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

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

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

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

add\_executable(${project} ${SOURCES})

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

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

target\_include\_directories(

${PROJECT\_NAME} PUBLIC

${CMAKE\_SOURCE\_DIR}/planet

${CMAKE\_SOURCE\_DIR}/buslib

)

**Библиотека Planet**

CMakeLists.txt

cmake\_minimum\_required(VERSION 3.23)

set(project "planet")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(SOURCES

planet.cpp

)

set(HEADERS

planet.h

)

add\_library(${project} STATIC ${SOURCES} ${HEADERS})

**planet.cpp**

#include "planet.h"

unsigned Planet::total = 0;

void Planet::AddPlanet(Planet \*db, size\_t &size, Planet &planetToAdd)

{

    if (size == CAPACITY)

    {

        std::cout << "FULL OF CAPACITY";

        return;

    }

    db[size++] = planetToAdd;

}

void Planet::PrintDB(Planet \*mas, size\_t size)

{

    for (size\_t i = 0; i < size; i++)

    {

        std::cout << mas[i];

    }

}

void Planet::DBtoFile(Planet \*mas, size\_t size, const char \*file1)

{

    std::fstream file(file1);

    for (size\_t i = 0; i < size; i++)

    {

        file << mas[i];

    }

    file.close();

}

void Planet::Sort(Planet \*mas, size\_t size)

{

    Planet k;

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

    {

        int min = i;

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

        {

            if (mas[j] < mas[min])

            {

                min = j;

            }

        }

        if (min != i)

        {

            k = mas[i];

            mas[i] = mas[min];

            mas[min] = k;

        }

    }

}

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

{

    char buf[30];

    in >> buf;

    planet.SetPlanetName(buf);

    unsigned Diameter\_;

    bool haveLife\_;

    unsigned satellites\_;

    in >> Diameter\_ >> haveLife\_ >> satellites\_;

    planet.SetDiameter(Diameter\_);

    planet.SetHaveLife(haveLife\_);

    planet.SetSatellites(satellites\_);

    return in;

}

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

{

    out << planet.GetPlanetName() << " " << planet.GetDiameter() << " " << planet.GetHaveLife() << " " << planet.GetSatellites() << "\n";

    return out;

}

unsigned Planet::FindID(char \*planetToFind, Planet \*mas, size\_t size)

{

    for (size\_t i = 0; i < size; i++)

    {

        if (strcmp(mas[i].planetName, planetToFind) == 0)

        {

            return mas[i].id;

        }

    }

    std::cout << "CAN'T FIND PLANET";

    return 0;

}

void Planet::DeletePlanet(Planet \*mas, size\_t &size, char \*planetToDelete)

{

    int curIndex = -1;

    for (size\_t i = 0; i < size; i++)

    {

        if (strcmp(planetToDelete, mas[i].GetPlanetName()) == 0)

        {

            curIndex = i;

            break;

        }

    }

    if (curIndex == -1)

    {

        std::cout << "Planet not found";

        return;

    }

    size--;

    for (size\_t i = curIndex; i < size; i++)

    {

        mas[i] = mas[i + 1];

    }

    std::cout << "planet deleted";

    return;

}

void Planet::EditPlanetValues(Planet \*mas, size\_t size)

{

    char planetName[30];

    std::cout << "\nEnter planet name to edit:\n";

    std::cin >> planetName;

    int curIndex = -1;

    for (size\_t i = 0; i < size; i++)

    {

        if (strcmp(planetName, mas[i].GetPlanetName()) == 0)

        {

            curIndex = i;

            break;

        }

    }

    if (curIndex == -1)

    {

        std::cout << "Planet not found.\n";

        return;

    }

    std::cout << "which one of the values you want to edit: Name of the planet(press 1), Diameter(press 2), Having of life(press 3), Number of satellites(press 4)";

    int ans;

    std::cin >> ans;

    switch (ans)

    {

    case 1:

        std::cout << "enter new name of the planet:";

        char ansName[30];

        std::cin >> ansName;

        mas[curIndex].SetPlanetName(ansName);

        std::cout << "\n";

        break;

    case 2:

        std::cout << "enter new diameter:";

        unsigned newDiameter;

        std::cin >> newDiameter;

        mas[curIndex].SetDiameter(newDiameter);

        std::cout << "\n";

        break;

    case 3:

        std::cout << "New 'having of life' value?(1/0)";

        bool newValue;

        std::cin >> newValue;

        mas[curIndex].SetHaveLife(newValue);

        std::cout << "\n";

        break;

    case 4:

        std::cout << "set new number of satellites:";

        unsigned satellites;

        std::cin >> satellites;

        mas[curIndex].SetSatellites(satellites);

        std::cout << "\n";

        break;

    default:

        break;

    }

    return;

}

unsigned menu()

{

    unsigned ans;

    std::cout << "\nPress 1 to read Database\n";

    std::cout << "Press 2 to write to Database\n";

    std::cout << "Press 3 to edit planet values\n";

    std::cout << "Press 4 to print Database\n";

    std::cout << "Press 5 to sort Database\n";

    std::cout << "Press 6 to add planet\n";

    std::cout << "Press 7 to delete planet\n";

    std::cout << "Press 8 to exit\n";

    std::cin >> ans;

    return ans;

}

void Planet::SetPlanetName(char \*planetNameToSet)

{

    if (planetName!=nullptr)

    {

        delete[] planetName;

    }

    planetName = new char[strlen(planetNameToSet) + 1];

    strcpy(planetName, planetNameToSet);

}

void Planet::SetDiameter(unsigned int DiameterToSet)

{

    Diameter = DiameterToSet;

}

void Planet::SetHaveLife(bool haveLifeValue)

{

    haveLife = haveLifeValue;

}

void Planet::SetSatellites(unsigned int satellitesToSet)

{

    satellites = satellitesToSet;

}

char\* Planet::GetPlanetName()

{

    return planetName;

}

unsigned int Planet::GetDiameter()

{

    return Diameter;

}

bool Planet::GetHaveLife()

{

    return haveLife;

}

unsigned int Planet::GetSatellites()

{

    return satellites;

}

unsigned Planet::GetID()

{

    return id;

}

Planet::Planet()

{

    planetName = new char[1];

    planetName[0] = '\0';

    total++;

    id = total;

*// std::cout << "created ID " << id << "\n";*

}

Planet::Planet(char \*p, unsigned d, bool h, unsigned s)

{

    planetName = new char[strlen(p) + 1];

    strcpy(planetName, p);

    total++;

    id = total;

*// std::cout << "created ID " << id << "\n";*

}

Planet::~Planet()

{

    total--;

*// std::cout << "deleted ID " << id << "\n";*

    delete[] planetName;

}

Planet& Planet::operator=(Planet &planet\_)

{

    if (this == &planet\_)

    {

        return \*this;

    }

    char newName[30];

    strcpy(newName, planet\_.GetPlanetName());

    SetPlanetName(newName);

    Diameter = planet\_.Diameter;

    haveLife = planet\_.haveLife;

    satellites = planet\_.satellites;

    return \*this;

}

bool Planet::operator==(Planet &planetForEqual)

{

    return Diameter == planetForEqual.Diameter;

}

bool Planet::operator<(Planet &planetForCompare)

{

    return Diameter < planetForCompare.Diameter;

}

Planet::Planet(Planet &planetToCopy)

{

    Diameter = planetToCopy.Diameter;

    haveLife = planetToCopy.haveLife;

    satellites = planetToCopy.satellites;

    SetPlanetName(planetToCopy.GetPlanetName());

}

**Planet.h**

#pragma once

#define CAPACITY 15

#include <iostream>

#include <fstream>

#include <cstring>

class Planet

{

private:

    char \*planetName;

    unsigned int Diameter;

    bool haveLife;

    unsigned int satellites;

private:

    static unsigned total;

    unsigned id = 0;

public:

    void SetPlanetName(char \*planetNameToSet);

    void SetDiameter(unsigned int DiameterToSet);

    void SetHaveLife(bool haveLifeValue);

    void SetSatellites(unsigned int satellitesToSet);

    char \*GetPlanetName();

    unsigned int GetDiameter();

    bool GetHaveLife();

    unsigned int GetSatellites();

    unsigned GetID();

    Planet();

    Planet(char \*p, unsigned d = 0, bool h = 0, unsigned s = 0);

    ~Planet();

    void operator=(Planet &planet\_);

    bool operator==(Planet &planetForEqual);

    bool operator<(Planet &planetForCompare);

    Planet(Planet &planetToCopy);

    static void AddPlanet(Planet \*db, size\_t &size, Planet &planetToAdd);

    static void PrintDB(Planet \*mas, size\_t size);

    static void DBtoFile(Planet \*mas, size\_t size, const char \*file1);

    static void Sort(Planet \*mas, size\_t size);

    static unsigned FindID(char \*planetToFind, Planet \*mas, size\_t size);

    static void DeletePlanet(Planet \*mas, size\_t &size, char \*planetToDelete);

    static void EditPlanetValues(Planet \*mas, size\_t size);

};

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

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

unsigned menu();

**Библиотека Bus**

CMakeLists.txt

cmake\_minimum\_required(VERSION 3.23)

set(project "bus")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(SOURCES

bus.cpp

)

set(HEADERS

bus.h

)

add\_library(${project} STATIC ${SOURCES} ${HEADERS})

**bus.cpp**

#include "bus.h"

unsigned Bus::total = 0;

void Bus::AddBus(Bus \*db, size\_t &size)

{

    if (size == CAPACITY)

    {

        std::cout << "FULL OF CAPACITY\n";

        return;

    }

    std::cout << "enter values of new bus:\n";

    Bus newBus;

    std::cin >> newBus;

    db[size++] = newBus;

}

void Bus::PrintDB(Bus \*mas, size\_t size)

{

    for (size\_t i = 0; i < size; i++)

    {

        std::cout << mas[i];

    }

}

void Bus::DBtoFile(Bus \*mas, size\_t size, const char \*file1)

{

    std::ofstream file(file1);

    for (size\_t i = 0; i < size; i++)

    {

        file << mas[i];

    }

    file.close();

}

void Bus::Sort(Bus \*mas, size\_t size)

{

    Bus k;

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

    {

        int min = i;

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

        {

            if (mas[j] < mas[min])

            {

                min = j;

            }

        }

        if (min != i)

        {

            k = mas[i];

            mas[i] = mas[min];

            mas[min] = k;

        }

    }

}

std::istream &operator>>(std::istream &in, Bus &bus)

{

    char model[30];

    in >> model;

    bus.SetBusModel(model);

    unsigned year;

    unsigned mileage;

    bool availability;

    in >> year >> mileage >> availability;

    bus.SetYearOfIssue(year);

    bus.SetMileage(mileage);

    bus.SetAvailability(availability);

    return in;

}

std::ostream &operator<<(std::ostream &out, Bus &bus)

{

    out << bus.GetBusModel() << " " << bus.GetYearOfIssue() << " " << bus.GetMileage() << " " << bus.GetAvailability() << "\n";

    return out;

}

unsigned Bus::FindID(char \*modelToFind, Bus \*mas, size\_t size)

{

    for (size\_t i = 0; i < size; i++)

    {

        if (strcmp(mas[i].GetBusModel(), modelToFind) == 0)

        {

            return mas[i].id;

        }

    }

    std::cout << "CAN'T FIND BUS";

    return 0;

}

void Bus::DeleteBus(Bus \*mas, size\_t &size)

{

    char modelName[30];

    std::cout << "\nEnter bus model to delete:\n";

    std::cin >> modelName;

    int curIndex = -1;

    for (size\_t i = 0; i < size; i++)

    {

        if (strcmp(modelName, mas[i].GetBusModel()) == 0)

        {

            curIndex = i;

            break;

        }

    }

    if (curIndex == -1)

    {

        std::cout << "Bus not found.";

        return;

    }

    size--;

    for (size\_t i = curIndex; i < size; i++)

    {

        mas[i] = mas[i + 1];

    }

    return;

}

void Bus::EditBusValues(Bus \*mas, size\_t size)

{

    char modelName[30];

    std::cout << "\nEnter bus model to edit:\n";

    std::cin >> modelName;

    int curIndex = -1;

    for (size\_t i = 0; i < size; i++)

    {

        if (strcmp(modelName, mas[i].GetBusModel()) == 0)

        {

            curIndex = i;

            break;

        }

    }

    if (curIndex == -1)

    {

        std::cout << "Bus not found.\n";

        return;

    }

    std::cout << "which value do you want to edit: Model (press 1), Year of issue (press 2), Mileage (press 3), Availability (press 4)";

    int ans;

    std::cin >> ans;

    switch (ans)

    {

    case 1:

        std::cout << "enter new model name:";

        char newModel[30];

        std::cin >> newModel;

        mas[curIndex].SetBusModel(newModel);

        std::cout << "\n";

        break;

    case 2:

        std::cout << "enter new year of issue:";

        unsigned newYear;

        std::cin >> newYear;

        mas[curIndex].SetYearOfIssue(newYear);

        std::cout << "\n";

        break;

    case 3:

        std::cout << "enter new mileage:";

        unsigned newMileage;

        std::cin >> newMileage;

        mas[curIndex].SetMileage(newMileage);

        std::cout << "\n";

        break;

    case 4:

        std::cout << "New availability value? (1/0):";

        bool newAvailability;

        std::cin >> newAvailability;

        mas[curIndex].SetAvailability(newAvailability);

        std::cout << "\n";

        break;

    default:

        break;

    }

    return;

}

unsigned menuBus()

{

    unsigned ans;

    std::cout << "\nPress 1 to read Database\n";

    std::cout << "Press 2 to write to Database\n";

    std::cout << "Press 3 to edit bus values\n";

    std::cout << "Press 4 to print Database\n";

    std::cout << "Press 5 to sort Database\n";

    std::cout << "Press 6 to add bus\n";

    std::cout << "Press 7 to delete bus\n";

    std::cout << "Press 8 to exit\n";

    std::cin >> ans;

    return ans;

}

void Bus::SetBusModel(char \*busModelToSet)

{

    if (busModel!=nullptr)

    {

        delete[] busModel;

    }

    busModel = new char[strlen(busModelToSet) + 1];

    strcpy(busModel, busModelToSet);

}

void Bus::SetYearOfIssue(unsigned yearOfIssueToSet)

{

    yearOfIssue = yearOfIssueToSet;

}

void Bus::SetMileage(unsigned mileageToSet)

{

    mileage = mileageToSet;

}

void Bus::SetAvailability(bool availabilityValue)

{

    availability = availabilityValue;

}

char \*Bus::GetBusModel()

{

    return busModel;

}

unsigned Bus::GetYearOfIssue()

{

    return yearOfIssue;

}

unsigned Bus::GetMileage()

{

    return mileage;

}

bool Bus::GetAvailability()

{

    return availability;

}

Bus::Bus()

{

    busModel = new char[1];

    busModel[0] = '\0';

    total++;

    id = total;

*// std::cout << "created ID " << id << "\n";*

}

Bus::Bus(char \*m, unsigned y, unsigned mile, bool av)

{

    busModel = new char[strlen(m) + 1];

    strcpy(busModel, m);

    total++;

    id = total;

*// std::cout << "created ID " << id << "\n";*

}

Bus::~Bus()

{

    total--;

*// std::cout << "deleted ID " << id << "\n";*

    delete[] busModel;

}

Bus& Bus::operator=(Bus &bus\_)

{

    if (this == &bus\_)

    {

        return \*this;

    }

    char newModel[15];

    strcpy(newModel, bus\_.busModel);

    SetBusModel(newModel);

    yearOfIssue = bus\_.yearOfIssue;

    mileage = bus\_.mileage;

    availability = bus\_.availability;

    return \*this;

}

bool Bus::operator==(Bus &busForEqual)

{

    return mileage == busForEqual.mileage;

}

bool Bus::operator<(Bus &busForCompare)

{

    return mileage < busForCompare.mileage;

}

Bus::Bus(Bus &busToCopy)

{

    SetBusModel(busToCopy.GetBusModel());

    yearOfIssue = busToCopy.yearOfIssue;

    mileage = busToCopy.mileage;

    availability = busToCopy.availability;

}

**Bus.h**

#pragma once

#ifndef CAPACITY

#define CAPACITY 12

#endif

#include <iostream>

#include <fstream>

#include <cstring>

class Bus

{

    char \*busModel;

    unsigned yearOfIssue = 0;

    unsigned mileage = 0;

    bool availability = 0;

    static unsigned total;

    unsigned id = 0;

public:

    void SetBusModel(char \*busModelToSet);

    void SetYearOfIssue(unsigned yearOfIssueToSet);

    void SetMileage(unsigned mileageToSet);

    void SetAvailability(bool availabilityValue);

    char \*GetBusModel();

    unsigned GetYearOfIssue();

    unsigned GetMileage();

    bool GetAvailability();

    Bus();

    Bus(char \*m, unsigned y, unsigned mile, bool av);

    ~Bus();

    void operator=(Bus &bus\_);

    bool operator==(Bus &busForEqual);

    bool operator<(Bus &busForCompare);

    Bus(Bus &busToCopy);

    static void AddBus(Bus \*db, size\_t &size);

    static void PrintDB(Bus \*mas, size\_t size);

    static void DBtoFile(Bus \*mas, size\_t size, const char \*file1);

    static void Sort(Bus \*mas, size\_t size);

    static unsigned FindID(char \*busModelToFind, Bus \*mas, size\_t size);

    static void DeleteBus(Bus \*mas, size\_t &size);

    static void EditBusValues(Bus \*mas, size\_t size);

};

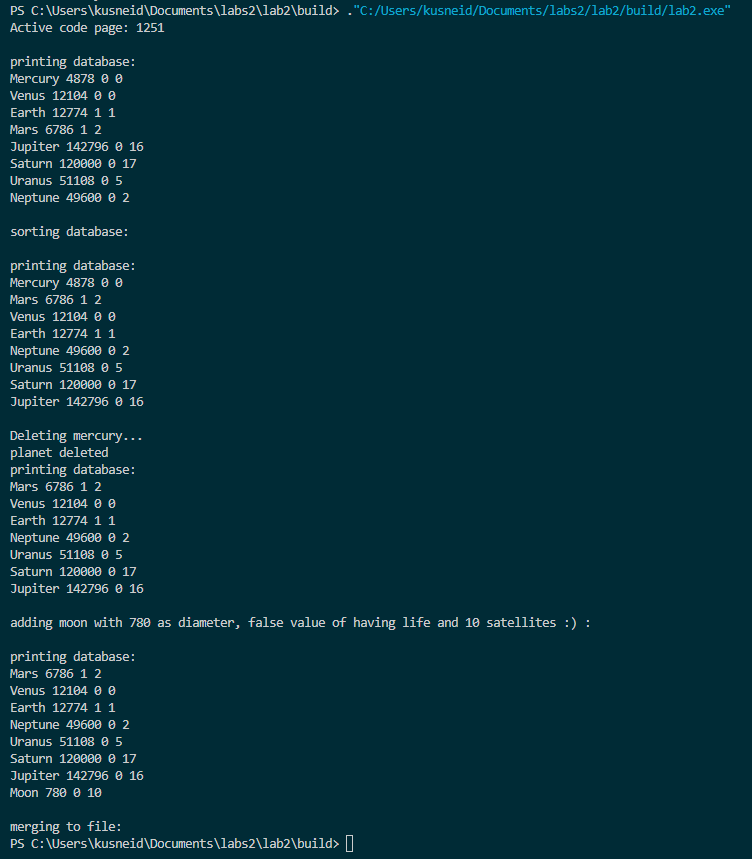
std::istream &operator>>(std::istream &in, Bus &bus);

std::ostream &operator<<(std::ostream &out, Bus &bus);

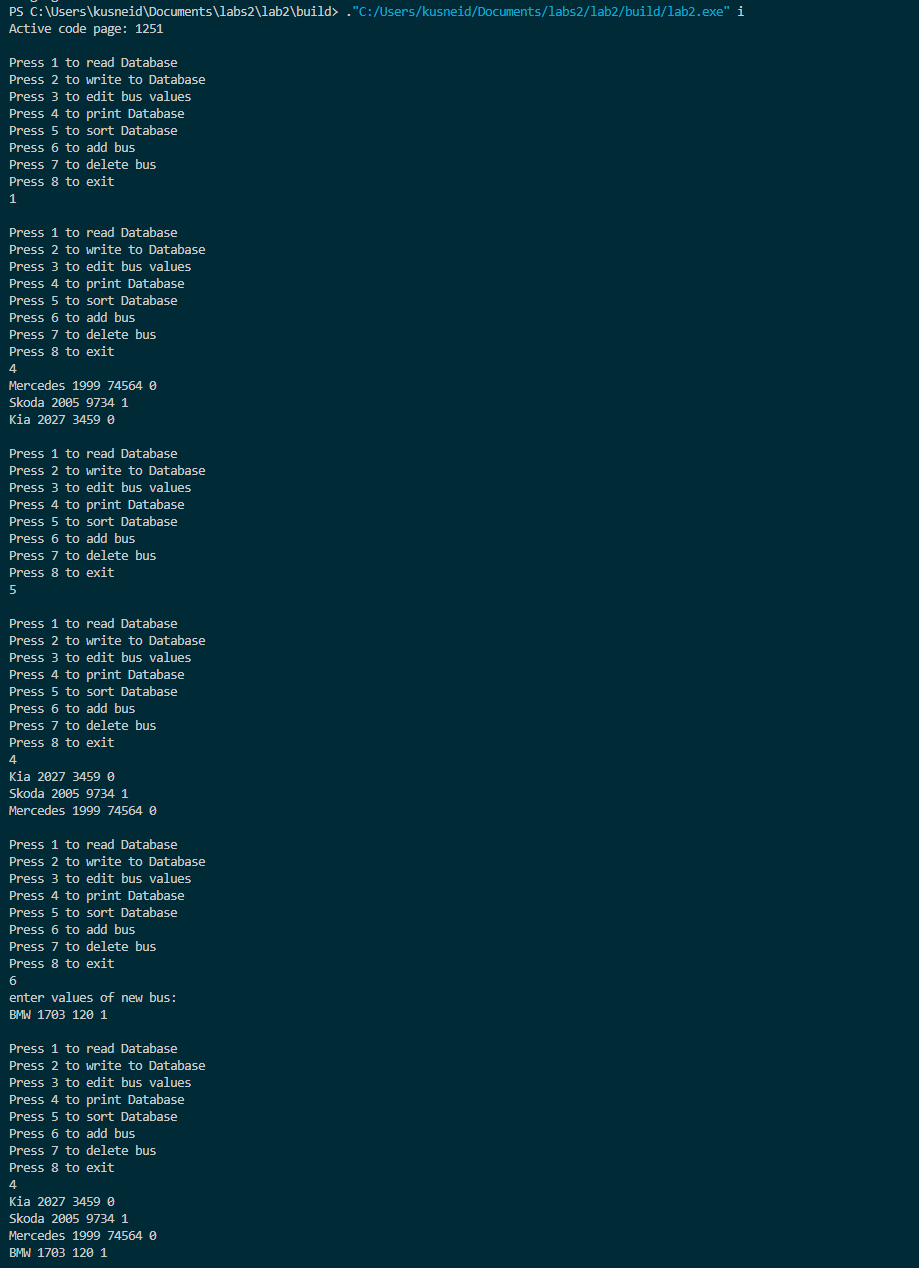
unsigned menuBus();

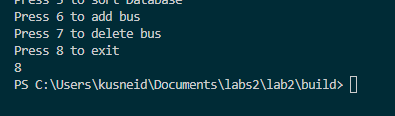
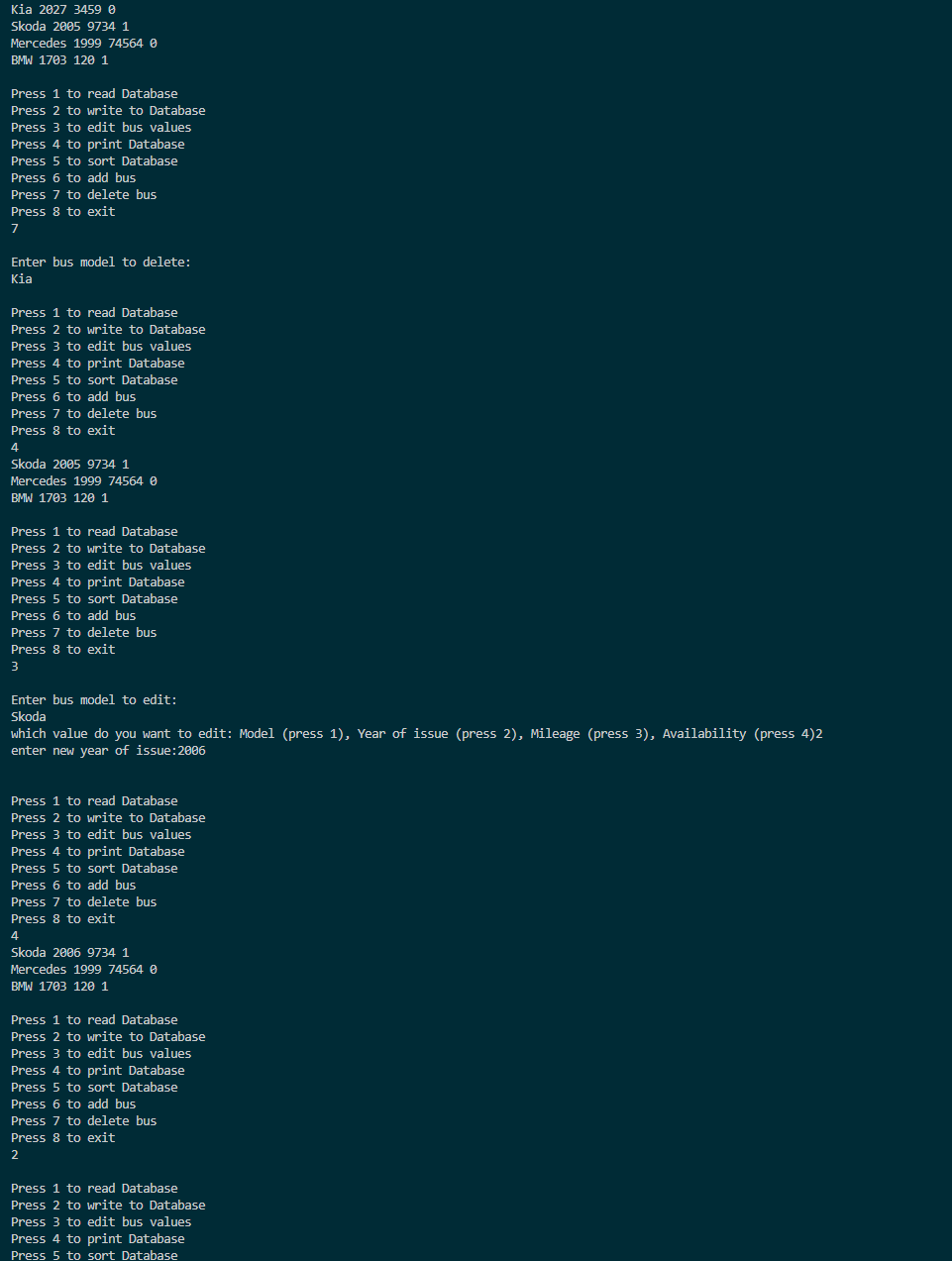
**Анализ результатов**

**Демонстрационный режим**



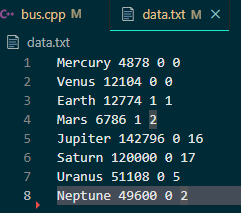
**Интерактивный режим**

****

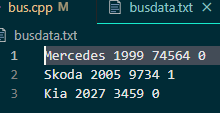
****

**Изначальные данные txt файлов до выполнения программы**

Data.txt



Busdata.txt



После выполнения:

