

ГУАП

КАФЕДРА № 14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

ассистент

должность, уч. степень, звание

подпись, дата

А.Ю. Петров

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

7041

подпись, дата

Н.А. Тыртышный

инициалы, фамилия

Санкт-Петербург 2022

Оглавление

	по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ.....	1
1	Постановка задачи	3
2	Формализация задачи.....	4
3	Таблица описания классов	5
4	Исходный код	Error! Bookmark not defined.
5	Результаты работы программы.....	6
6	Выводы	8
	Приложение 1	9
	Листинг программы	9

1 Постановка задачи

Задания выполняются в соответствии с вариантом №4.

Разработать класс Кеерет, который реализует контейнер для хранения и обработки объектов. Класс Кеерет должен обеспечивать следующие функции:

- добавление и удаление производных объектов абстрактного класса Base (базовый класс определяется согласно варианту задания);
- полное сохранение себя в файле;
- полное восстановление себя из файла.

Класс Гараж хранит данные об автомобилях, мотоциклах и автобусах. Для каждого автомобиля определено: марка, модель, объем двигателя, цвет, тип КПП. Для каждого мотоцикла определено: марка, модель, объем двигателя, мощность двигателя, для какой местности мотоцикл предназначен. Для каждого автобуса определено: марка, модель, количество сидячих пассажирских мест, общее количество пассажирских мест, конечный пункт.

Важно учитывать при реализации

- Работать необходимо с динамическим выделением памяти.
- Не использовать STL контейнеры для реализации программы.
- Реализовать полное пользовательское меню согласно заданию. Не должно быть аргументов, которые явно задаются в исходном коде, пользователь имеет максимальные возможности при работе с программой, все параметры, которые могут быть введены пользователем, вводятся с клавиатуры.
- У пользователя должен быть доступ для взаимодействия с каждым из объектов-наследников: добавлять, удалять, выводить на экран, изменять данные и пр.
- Создать все конструкторы (с параметрами, без параметров и копирования) и деструктор, использовать переопределение операторов. Каждый вызов конструктора и деструктора должен сопровождаться выводом этой информации пользователю (какой объект, какой конструктор или деструктор вызван).
- Определить в классе методы для просмотра и установки значений (set и get функции).
- Определить исключения (exceptions) и применить в программе. Каждое исключение должно сопровождаться текстом, поясняющим возникшую исключительную ситуацию.

- Классы-наследники от базового класса должны определять свою сущность по переменным и методам для работы с ними. Например, у животного есть тип, пол, возраст, имя, хозяин, описание и т.д. В задании определено некоторое описание характеристик.

- По списку студент определяет свой порядковый номер и этому номеру соответствует номер задания в таблице. Общее задание расположено перед таблицей.

- Необходимо выполнить разделение на h и cpp файлы для каждого класса. h файлы содержат определение, cpp файлы содержат реализацию. Функция main обязана располагаться в отдельном cpp файле.

2 Формализация задачи

По заданию необходимо разделения на h и cpp файлы для каждого класса. Функция main должна располагаться в отдельном cpp файле. Поэтому в начале программы создаем отдельные файлы “main.cpp”, “menu.h”, “keeper.h” – реализует контейнер для хранения и обработки объектов, “garage.h”, “garage.cpp” – родительский класс гаража, “bus.h”, “bus.cpp” – дочерний класс автобус, “moto.h”, “moto.cpp” – дочерний класс мотоцикл, “car.h”, “car.cpp” – дочерний класс машина.

В файл main подключаем все заголовочные файлы: “menu.h”, “keeper.h”, “garage.h”, “bus.h”, “moto.h”, “car.h”. Также в main мы реализовываем пользовательское меню, чтобы с помощью клавиатуры выбирать необходимый класс (Автобус, Мотоцикл или Машина), а затем действия, которые мы хотим сделать с этим объектом (Добавление объекта, Удаление объекта, Изменение данных объекта, Вывод на экран, Запись объекта в файл, Чтение объекта из файла и Выход из программы).

Объявляем три экземпляра по каждому из трех классов (Keeper <Bus> bus; Keeper <Car> car; Keeper <Moto> moto;). Также с помощью цикла while и операторов switch и case организовываем работу с пользовательским меню.

Класс keeper реализует контейнер для хранения и обработки объектов. Он обеспечивать следующие функции: добавление и удаление производных объектов класса garage, полное сохранение себя в файле, полное восстановление себя из файла, печать данных на консоль, а также редактирование производных объектов класса garage.

Все три класса bus, moto, car имеют тип наследования public от родительского класса garage.

Также во всех дочерних классах объявлены private переменные, которые отвечают за характеристику объекта (тип, марка, номер, цвет и т.д.) и public функции. В данной лабораторной работе в трех классах bus, moto, car использовалась перегрузка оператора с помощью дружественной функции и перегрузка оператора с помощью метода класса. Перегрузка оператора с помощью дружественной функции использовалась для ввода

данных объекта с помощью клавиатуры, вывода данных объектов на консоль, Запись данных объекта в файл и чтения данных объекта из файла. Перегрузка оператора с помощью метода класса использовалась для указателя `this`, чтобы дать компилятору понять, что именно и куда надо скопировать. Это использовалось для характеристик каждого объекта.

В классе `keeper` использовались исключения (`exceptions`). Каждое исключение должно сопровождаться текстом, поясняющим возникшую исключительную ситуацию, поэтому если пользователь вводит неправильные данные на экран могут выводиться такие сообщения, как "Пусто", "Неверный номер".

Чтобы не засорять консоль программы пользовательским меню, использовали функцию `system "cls"`, которая как раз очищает экран. Также использовалась функция `system "pause"`, чтобы наоборот остались данные на экране до нажатия любой клавиши. При этом на экране появляется надпись «Для продолжения нажмите любую клавишу . . .».

3 Таблица описания классов

На Рисунок 1 продемонстрирована иерархия связи классов, а именно наследование (`garage` – родительский класс гараж, `bus`– дочерний класс автобус, `work` – дочерний класс мотоцикл, `car` – дочерний класс машина).

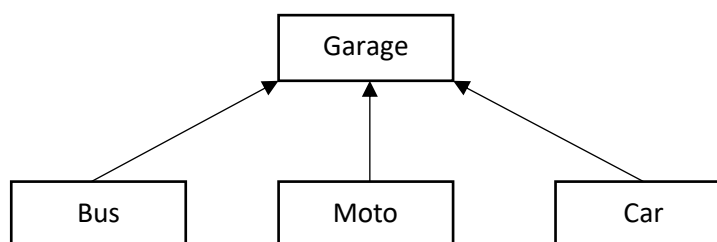


Рисунок 1 – Иерархия классов

В Таблица 1 представлены все классы с полным описанием (имя, назначение, наследование и его тип).

Таблица 1 – Описания классов

Класс	Описание
Keeper	Хранитель. Предназначен, чтобы реализовывать контейнер для хранения и обработки объектов. В нем содержатся такие функции, как: добавление объекта, удаление объекта, редактирование объекта, вывод объекта на консоль, запись объекта в файл и чтение объекта из файла. Не наследуется. Используется в функции меню (<code>menu</code>).
Garage	Гараж. Базовый класс (класс-родитель). В нем находятся

	конструктор, деструктор и виртуальная функция с модификатором доступа в классе: public. Этот класс хранит данные о мебели, мотоциклах и машинах.
Bus	Автобус. Производный класс (класс-наследник). Является наследником для базового класса Bus. Имеет тип наследования public. В этом классе для автобуса определено: марка, модель, количество сидячих пассажирских мест, общее количество пассажирских мест, конечный пункт. С помощью перегрузки операторов с дружественными функциями и методом класса реализовывает ввод данных объекта с клавиатуры, вывод их на экран, запись в файл и чтение этих данных из файла.
Moto	Мотоцикл. Производный класс (класс-наследник). Является наследником для базового класса Bus. Имеет тип наследования public. В этом классе для мотоцикла определено: марка, модель, объем двигателя, мощность двигателя, для какой местности мотоцикл предназначен. С помощью перегрузки операторов с дружественными функциями и методом класса реализовывает ввод данных объекта с клавиатуры, вывод их на экран, запись в файл и чтение этих данных из файла.
Car	Машина. Производный класс (класс-наследник). Является наследником для базового класса Bus. Имеет тип наследования public. В этом классе для машины определено: марка, модель, объем двигателя, цвет, тип КПП. С помощью перегрузки операторов с дружественными функциями и методом класса реализовывает ввод данных объекта с клавиатуры, вывод их на экран, запись в файл и чтение этих данных из файла.

4 Результаты работы программы

Приведем пример работы программы. После запуска программы на экран выводится меню (Рисунок 2), которое предлагает выбрать необходимый объект. Например, выберем пункт с автобусом, для этого введем число 1. После нажатия на enter функция стирает данное меню и теперь появляется другое (Рисунок 2), которое состоит из 7 пунктов. Введем цифру 1, чтобы ввести новый объект со своими данными.

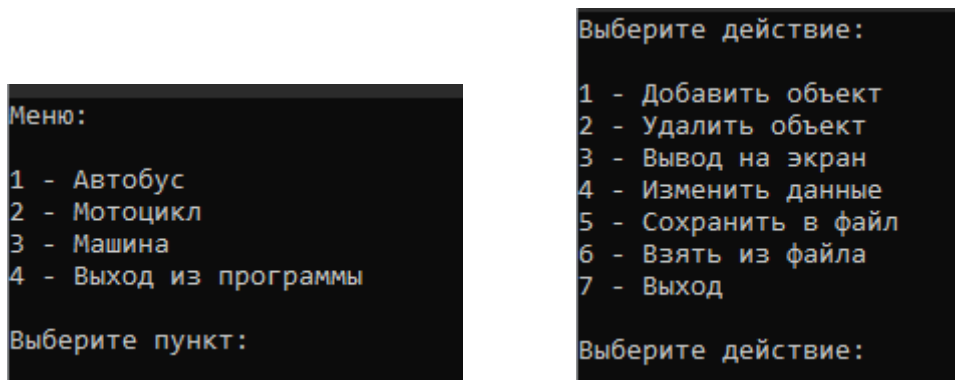


Рисунок 2 – Результат работы программы 1

Прошное меню снова стирается с консоли, на экране показываются сообщения, какие конструкторы были вызваны, а также предлагается ввести данные объекта Автобус (Рисунок 3). На этом же Рисунок 3 справа показано, что мы ввели с клавиатуры. Также выводится сообщение о том, что объект добавлен.

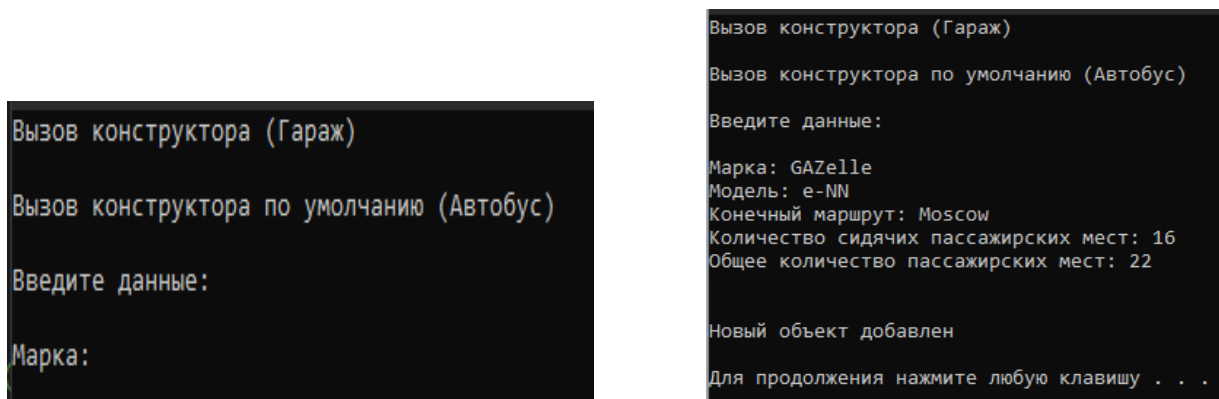


Рисунок 3 – Результат работы программы 2

Снова нажимаем кнопку enter и выводится на консоль меню для объекта (Рисунок 4). Попробуем вывести данные прошлого объекта на экран. Для этого вводим пункт 3. Как видно по Рисунок 4 все данные вывелись на консоль.

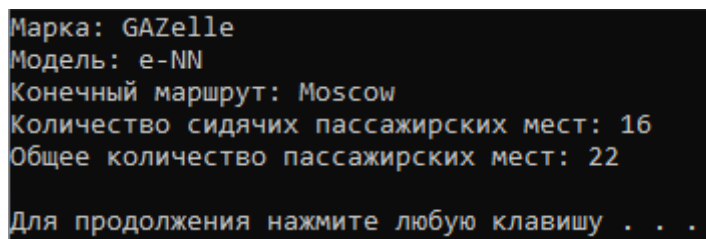
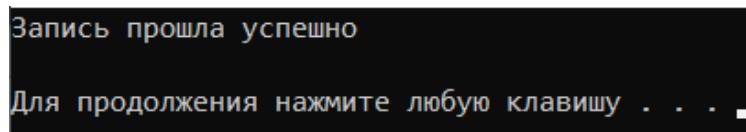


Рисунок 4 – Результат работы программы 3

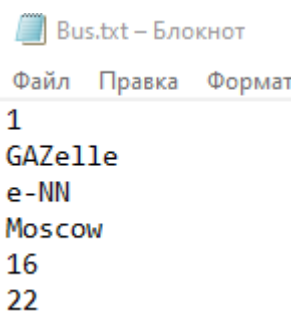
После этого попробуем вызвать пункт 5 меню, чтобы записать данные в файл. На консоль выводится сообщение о том, что запись прошла успешно (Рисунок 5).



```
Запись прошла успешно  
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5 – Результат работы программы 4

Проверим это, открыв созданный программой файл. Он имеет название “bus.txt”. Как видно по Рисунок 6 данные сохранились в файле.



```
Bus.txt – Блокнот  
Файл  Правка  Формат  
1  
GAZelle  
e-NN  
Moscow  
16  
22
```

Рисунок 6 – Результат работы программы 5

5 Выводы

В ходе лабораторной работы были получены навыки по созданию базового и наследных классов. Было реализовано пользовательское меню, благодаря которому все данные об объектах вводятся пользователем с клавиатуры. Было реализовано разделение на h и cpp файлы для каждого класса. Также были определены исключения (exceptions), чтобы пользователь знал, что допустил ошибку, а классы-наследники от базового класса определяли свою сущность по переменным и методам для работы с ними.

Приложение 1

Листинг программы

Ссылка кода на GitHub: <https://github.com/RadioactiveFrogSouse/Lab1.git>

Файл "main.cpp":

```
#include <iostream>
#include "keeper.h"
#include "garage.h"
#include "bus.h"
#include "car.h"
#include "menu.h"
#include "moto.h"

using namespace std;

int main() {
    setlocale(LC_ALL, "Russian");
    Keeper <Bus> bus;
    Keeper <Car> car;
    Keeper <Moto> moto;
    int button;
    bool flag = true; // Для проверки логических условий (флаг состояний), как
    выключатель - ВКЛ и ВЫКЛ
    while (flag) {
        system("cls"); // Ждет ввода, чтобы окно консоли не закрылось
        cout << "Меню: " << endl << endl;
        cout << "1 - Автобус" << endl;
        cout << "2 - Мотоцикл" << endl;
        cout << "3 - Машина" << endl;
        cout << "4 - Выход из программы" << endl << endl;
        cout << "Выберите пункт: ";
        cin >> button;
        if (cin.fail()) { // Проверка на правильный ввод
            button = -1;
            cin.clear();
            cin.ignore(10000, '\n');
        }
    }
```

```

switch (button) {
case 1:
    menu(bus, "Bus.txt"); // Вызов меню (Автобус)
    break;
case 2:
    menu(moto, "Moto.txt"); // Вызов меню (Мотоцикл)
    break;
case 3:
    menu(car, "Car.txt"); // Вызов меню (Машина)
    break;
case 4:
    flag = false; // Выход из программы
    cout << "\nКонец" << endl;
    break;
default:
    cout << endl << "Некорректные данные, введите заново!" <<
endl << endl; // Проверка на правильный ввод
    system("pause");
    break;
}
}
return 0;
}

```

Файл "menu.h":

```

#pragma once
#include "keeper.h"
#include "garage.h"
#include "bus.h"
#include "car.h"
#include "moto.h"
#include <iostream>
using namespace std;
template <typename T> // T - тип, указанный в параметре шаблона
void menu(T& obj, string line) {
    int c;

```

```

int i;
bool flag = true;
setlocale(LC_ALL, "Russian");
while (flag) {
    system("cls");
    cout << "Выберите действие: " << endl << endl;
    cout << "1 - Добавить объект" << endl;
    cout << "2 - Удалить объект" << endl;
    cout << "3 - Вывод на экран" << endl;
    cout << "4 - Изменить данные" << endl;
    cout << "5 - Сохранить в файл" << endl;
    cout << "6 - Взять из файла" << endl;
    cout << "7 - Выход" << endl << endl;
    cout << "Выберите действие: ";
    cin >> c;
    if (cin.fail() || c < 0 || c > 7) { // Проверка на правильный ввод
        c = -1;
        cin.clear();
        cin.ignore(10000, '\n');
    }
    system("cls");
    switch (c) {
    case 1:
        obj.push(); // Добавление объекта
        system("pause");
        break;

    case 2: obj.display();
        cout << "Выберите для удаления: ";
        cin >> i;
        obj.pop(i); // Удаление объекта
        system("pause");
        break;

    case 3:
        obj.display(); // Вывод объекта на экран

```

```

        system("pause");
        break;
    case 4:
        obj.display();
        cout << "Выберите для изменения: ";
        cin >> i;
        obj.edit(i); // Редактирование объекта
        system("pause");
        break;
    case 5:
        obj.write(line); // Запись объекта в файл
        system("pause");
        break;
    case 6:
        obj.read(line); // Считывание объекта из файла
        system("pause");
        break;
    case 7:
        flag = false; // Выход
        break;
    default:
        cout << "Введите от 1 до 7:" << endl << endl;
        system("pause");
        break;
    }
}
}

```

Файл “keeper.h”:

```

#pragma once
#include <iostream>
#include <fstream>
#include <string>
#include <exception>
using namespace std;
template <class T>

```

```

class Keeper {
    T* ptr;
    int size;
public:
    Keeper();
    ~Keeper();
    void push(); // Функция добавления нового объекта
    void pop(int); // Функция удаления объекта
    void write(string); // Функция записи объекта в файл
    void read(string); // Функция считывания объекта из файла
    void display(); // Функция вывода объекта на экран
    void edit(int); // Функция редактирования объекта
};

template<class T>
Keeper<T>::Keeper() { // Конструктор
    cout << "Вызов конструктора по умолчанию (Хранитель)" << endl << endl;
    ptr = nullptr;
    size = 0;
}

template<class T>
Keeper<T>::~Keeper() { // Деструктор
    cout << "Вызов деструктора (Хранитель)" << endl << endl;
    delete[] ptr;
}

template<class T>
void Keeper<T>::push() { // Функция добавления нового объекта
    T* tmp = new T[size + 1];
    for (int i = 0; i < size; ++i) {
        tmp[i] = ptr[i];
    }
    delete[] ptr;
    ptr = tmp;
    cin >> ptr[size];
    ++size;
    cout << endl << endl << "Новый объект добавлен" << endl << endl;
}

```

```

}

template<class T>
void Keeper<T>::pop(int num) { // Функция удаления объекта
    try {
        if (size == 0) {
            exception error("Пусто");
            throw error;
        }
        if (num < 0 || num >= size) {
            exception bug("Неверный номер");
            throw bug;
        }
        T* tmp = new T[size - 1];
        ptr[num] = ptr[size - 1];
        for (int i = 0; i < size - 1; ++i) {
            tmp[i] = ptr[i];
        }
        delete[] ptr;
        ptr = tmp;
        --size;
        cout << endl << endl << "Объект добавлен" << endl << endl;
    }
    catch (exception& error) {
        cout << error.what() << endl << endl;
    }
}

template<class T>
void Keeper<T>::display() { // Функция вывода объекта на экран
    if (size == 0) {
        cout << "Пусто" << endl << endl;
    }
    else {
        for (int i = 0; i < size; ++i) {
            cout << ptr[i] << endl;
        }
    }
}

```

```

    }
}

template<class T>
void Keeper<T>::edit(int change) { // Функция редактирования объекта
    try {
        if (size == 0) {
            exception error("Пусто");
            throw error;
        }
        if (change < 0 || change >= size) {
            exception bug("Неверный номер");
            throw bug;
        }
        cin >> ptr[change];
    }
    catch (exception& bug) {
        cout << bug.what() << endl << endl;
    }
}

template<class T>
void Keeper<T>::write(string s) { // Функция записи объекта в файл
    ofstream fout(s, ios::out);
    try {
        if (size == 0) {
            exception error("Пусто");
            throw error;
        }
        fout << size << endl;
        for (int i = 0; i < size; ++i) {
            fout << ptr[i] << endl;
        }
        fout.close();

        cout << "Запись прошла успешно" << endl << endl;
    }
}

```

```

        catch (exception& error) {
            cout << error.what() << endl;
        }
    }
}

template<class T>
void Keeper<T>::read(string s) { // Функция считывания объекта из файла
    delete[] ptr;
    ifstream fin(s, ios::in);
    fin >> size;
    ptr = new T[size];
    for (int i = 0; i < size; ++i) {
        fin >> ptr[i];
    }
    fin.close();
    cout << "Прочитано успешно" << endl << endl;
}

```

Файл “garage.h”:

```

#pragma once
#include <fstream>
#include "garage.h"
#include "keeper.h"
using namespace std;
class Garage
{
public:
    Garage();
    virtual ~Garage();
    virtual void setVal() = 0;
};

```

Файл “garage.cpp”:

```

#include "garage.h"
#include <iostream>
using namespace std;

```



```

Garage::Garage()
{
    cout << "Вызов конструктора (Фабрика)" << endl << endl;
}
Garage::~~Garage()
{
    cout << "Вызов деструктора (Фабрика)" << endl << endl;
}

```

Файл "bus.h":

```

#pragma once
#include <fstream>
#include "keeper.h"
#include "garage.h"
using namespace std;
class Bus : public Garage
{
private:
    string marka;
    string model;
    double pas_sid;
    double pas_all;
    string end_p;
public:
    Bus();
    Bus(const Bus& other);
    ~Bus();
    void setVal() override;
    friend ostream& operator << (ostream& fout, Bus& obj);
    friend ifstream& operator >> (ifstream& fin, Bus& obj);
    friend ostream& operator << (ostream& out, Bus& obj);
    friend istream& operator >> (istream& in, Bus& obj);
    Bus& operator =(const Bus& other);
};

```

Файл "bus.cpp":

```

#include "bus.h"

```

```

#include <iostream>
#include <string>
using namespace std;
Bus::Bus() {
    setlocale(LC_ALL, "Russian");
    cout << "Вызов конструктора по умолчанию (Автобус)" << endl << endl;
    marka = "";
    model = "";
    end_p = "";
    pas_sid = 0;
    pas_all = 0;
}
Bus::Bus(const Bus& other) {
    setlocale(LC_ALL, "Russian");
    cout << "Вызов конструктора копирования (Автобус)" << endl << endl;
    *this = other;
}
Bus::~Bus() {
    setlocale(LC_ALL, "Russian");
    cout << "Вызов деструктора (Автобус)" << endl << endl;
}
void Bus::setVal() {
    cin >> *this;
}
Bus& Bus::operator=(const Bus& other) {
    this->marka = other.marka;
    this->model = other.model;
    this->end_p = other.end_p;
    this->pas_sid = other.pas_sid;
    this->pas_all = other.pas_all;
    return *this;
}
ostream& operator<<(ostream& fout, Bus& obj) { // Функция записи в файл объекта
(Автобус)
    fout << obj.marka << " " << endl;

```

```

        fout << obj.model << " " << endl;
        fout << obj.end_p << " " << endl;
        fout << obj.pas_sid << " " << endl;
        fout << obj.pas_all << " " << endl;
        fout << endl;
        return fout;
    }

    ifstream& operator>>(ifstream& fin, Bus& obj) { // Функция чтения файла объекта
(Автобус)

        fin >> obj.marka >> obj.model >> obj.end_p >> obj.pas_sid >> obj.pas_all;
        return fin;
    }

    ostream& operator<<(ostream& out, Bus& obj) { // Функция вывода на экран данных
объекта (Автобус)

        setlocale(LC_ALL, "Russian");
        out << "Марка: " << obj.marka << endl;
        out << "Модель: " << obj.model << endl;
        out << "Конечный маршрут: " << obj.end_p << endl;
        out << "Количество сидячих пассажирских мест: " << obj.pas_sid << endl;
        out << "Общее количество пассажирских мест: " << obj.pas_all << endl;
        return out;
    }

    istream& operator>>(istream& in, Bus& obj) { // Функция ввода данных объекта
(Автобус)

        setlocale(LC_ALL, "Russian");
        cout << "Введите данные:" << endl << endl;
        cout << "Марка: ";
        cin >> obj.marka;
        cout << "Модель: ";
        cin >> obj.model;
        cout << "Конечный маршрут: ";
        cin >> obj.end_p;
        while (1) {
            cout << "Количество сидячих пассажирских мест: ";
            cin >> obj.pas_sid;

```

```

        if (cin.fail() || obj.pas_sid < 0) {
            cout << "Некорректные данные, введите заново!" << endl;
            cin.clear();
            cin.ignore(10000, '\n');
            continue;
        }
        break;
    }
    while (1) {
        cout << "Общее количество пассажирских мест: ";
        cin >> obj.pas_all;
        if (cin.fail() || obj.pas_all < 0) {
            cout << "Некорректные данные, введите заново!" << endl;
            cin.clear();
            cin.ignore(10000, '\n');
            continue;
        }
        break;
    }
    return in;
} Файл "moto.h":
#pragma once
#include <fstream>
#include "garage.h"
#include "keeper.h"
using namespace std;
class Moto : public Garage
{
private:
    string marka;
    string model;
    string eng_vol;
    string power;
    string specal;

```

public:

```
Moto();  
~Moto();  
Moto(const Moto&);  
void setVal() override;  
friend ostream& operator << (ostream& fout, Moto& obj);  
friend ifstream& operator >> (ifstream& fin, Moto& obj);  
friend ostream& operator << (ostream& out, Moto& obj);  
friend ifstream& operator >> (ifstream& in, Moto& obj);  
Moto& operator =(const Moto& other);
```

```
};
```

Файл “moto.cpp”:

```
#include "moto.h"
```

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
Moto::Moto() {
```

```
    setlocale(LC_ALL, "Russian");
```

```
    cout << "Вызов конструктора по умолчанию (Мотоцикл)" << endl << endl;
```

```
    marka = "";
```

```
    model = "";
```

```
    eng_vol = "";
```

```
    power = "";
```

```
    specal = "";
```

```
}
```

```
Moto::Moto(const Moto& other) {
```

```
    setlocale(LC_ALL, "Russian");
```

```
    cout << "Вызов конструктора копирования (Мотоцикл)" << endl << endl;
```

```
    *this = other;
```

```
}
```

```
Moto::~Moto() {
```

```
    setlocale(LC_ALL, "Russian");
```

```
    cout << "Вызов деструктора (Мотоцикл)" << endl << endl;
```

```
}
```

```
void Moto::setVal() {
```

```

        cin >> *this;
    }

    Moto& Moto::operator=(const Moto& other) {
        this->marka = other.marka;
        this->model = other.model;
        this->eng_vol = other.eng_vol;
        this->power = other.power;
        this->specal = other.specal;
        return *this;
    }

    ofstream& operator<<(ofstream& fout, Moto& obj) { // Функция записи в файл
    объекта (Мотоцикл)
        fout << obj.marka << " " << endl;
        fout << obj.model << " " << endl;
        fout << obj.eng_vol << " " << endl;
        fout << obj.power << " " << endl;
        fout << obj.specal << " " << endl;
        fout << endl;
        return fout;
    }

    ifstream& operator>>(ifstream& fin, Moto& obj) { // Функция чтения файла объекта
    (Мотоцикл)
        fin >> obj.marka >> obj.model >> obj.eng_vol >> obj.power >> obj.specal;
        return fin;
    }

    ostream& operator<<(ostream& out, Moto& obj) { // Функция вывода на экран
    данных объекта (Мотоцикл)
        setlocale(LC_ALL, "Russian");
        out << "Марка: " << obj.marka << endl;
        out << "Модель: " << obj.model << endl;
        out << "Объём двигателя: " << obj.eng_vol << endl;
        out << "Мощность двигателя: " << obj.power << endl;
        out << "Местность: " << obj.specal << endl << endl;
        return out;
    }

```

```
istream& operator>>(istream& in, Moto& obj) { // Функция ввода данных объекта  
(Мотоцикл)
```

```
    setlocale(LC_ALL, "Russian");  
    cout << "Введите данные:" << endl << endl;  
    cout << "Марка: ";  
    cin >> obj.marka;  
    cout << "Модель: ";  
    cin >> obj.model;  
    cout << "Объём двигателя: ";  
    cin >> obj.eng_vol;  
    cout << "Мощность двигателя: ";  
    cin >> obj.power;  
    cout << "Местность: ";  
    cin >> obj.specal;  
    return in;
```

```
}
```

Файл “car.h”:

```
#pragma once
```

```
#include <fstream>
```

```
#include "garage.h"
```

```
#include "keeper.h"
```

```
using namespace std;
```

```
class Car : public Garage {
```

```
private:
```

```
    string marka;
```

```
    string model;
```

```
    string eng_vol;
```

```
    string color;
```

```
    string transm;
```

```
public:
```

```
    Car();
```

```
    ~Car();
```

```
    Car(const Car&);
```

```
    void setVal() override;
```

```
    friend ostream& operator << (ostream& fout, Car& obj);
```

```

friend ifstream& operator >> (ifstream& fin, Car& obj);
friend ostream& operator << (ostream& out, Car& obj);
friend istream& operator >> (istream& in, Car& obj);
Car& operator =(const Car& other);

};

Файл "car.cpp":
#include "car.h"
#include <iostream>
#include <string>
using namespace std;

Car::Car() {
    setlocale(LC_ALL, "Russian");
    cout << "Вызов конструктора по умолчанию (Машина)" << endl << endl;
    marka = "";
    model = "";
    eng_vol = "";
    color = "";
    transm = "";
}

Car::Car(const Car& other) {
    setlocale(LC_ALL, "Russian");
    cout << "Вызов конструктора копирования (Машина)" << endl << endl;
    *this = other;
}

Car::~Car() {
    setlocale(LC_ALL, "Russian");
    cout << "Вызов деструктора (Машина)" << endl << endl;
}

void Car::setVal() {
    cin >> *this;
}

Car& Car::operator=(const Car& other) {
    this->marka = other.marka;
    this->model = other.model;
    this->eng_vol = other.eng_vol;

```



```

        this->color = other.color;
        this->transm = other.transm;
        return *this;
    }

    ofstream& operator<<(ofstream& fout, Car& obj) { // Функция записи в файл объекта
(Машина)

        fout << obj.marka << " " << endl;
        fout << obj.model << " " << endl;
        fout << obj.eng_vol << " " << endl;
        fout << obj.color << " " << endl;
        fout << obj.transm << " " << endl;
        fout << endl;
        return fout;
    }

    ifstream& operator>>(ifstream& fin, Car& obj) { // Функция чтения файла объекта
(Машина)

        fin >> obj.marka >> obj.model >> obj.eng_vol >> obj.color >> obj.transm;
        return fin;
    }

    ostream& operator<<(ostream& out, Car& obj) { // Функция вывода на экран данных
объекта (Машина)

        setlocale(LC_ALL, "Russian");
        out << "Марка: " << obj.marka << endl;
        out << "Модель: " << obj.model << endl;
        out << "Объём двигателя: " << obj.eng_vol << endl;
        out << "Цвет: " << obj.color << endl;
        out << "Тип КПП: " << obj.transm << endl;
        return out;
    }

    istream& operator>>(istream& in, Car& obj) { // Функция ввода данных объекта
(Машина)

        setlocale(LC_ALL, "Russian");
        cout << "Введите данные:" << endl << endl;
        cout << "Марка: ";
        cin >> obj.marka;

```

```
    cout << "Модель: ";  
    cin >> obj.model;  
    cout << "Объём двигателя: ";  
    cin >> obj.eng_vol;  
    cout << "Цвет: ";  
    cin >> obj.color;  
    cout << "Тип КПП: ";  
    cin >> obj.transm;  
    return in;  
}
```