РУСЕНСКИ УНИВЕРСИТЕТ КАТЕДРА

„АНГЕЛ КЪНЧЕВ“ „КСТ“

**КУРСОВА ЗАДАЧА**

по „Обектно-ориентирано програмиране“

на Диан Лъчезаров Лазаров

студент I курс, специалност КСТ, група 26А, фак. № 233078

**Русе** **Ръководител:** гл. ас. д-р Ц. Кънева

**17.05.2024 г.**

Задание №7:

Дадена е следната класификация:

**Hypercar**

**Supercar**

**Performance**

**Luxury**

**Exotic**

**Sports**

**Truck**

**Car**

**Motorcycle**

**Vehicle**

Класификацията да се продължи поне на още две нива. Да се състави йерархия от класове, отразяваща създадената класификация. Да се декларират съответните класове, като всеки клас, с излючение на базовия, да имат поне по 2 собствени атрибута.

Да се дефинира виртуална функция, която извежда характеристиките на обект от всеки клас на йерархията. Във функцията ***main*** да се изгради едносвързан списък от обекти от произволни класове в йерархията. Да се разработи функция, която обхожда едносвързания списък и извежда информация за признаците на включените в него обекти.

Декларациите на всеки клас от йерархията да бъдат оформени в отделни заглавни (.h) файлове. Дефинициите на всеки клас и функцията ***“main”*** да бъдат оформени в отделни модули (.cpp файлове). Във всички файлове, съдържащи дефинициите на класовете и функцията ***“main”***, чрез директивата ***#include*** да се включат съответните заглавни файлове, съдържащи декларациите на класовете. Да се създаде проект, състоящ се от създадените модули.

1. **Описание на декларираните класове**
2. Клас **Vehicle**

* Тип на класа: базов
* Наследява от: няма
* Базов за: Car, Motorcycle, Truck
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* vehicle – име на класа
* Vehicle\* next – нужен за съставяне на едносвързан списък
* Конструктори:
* Vehicle(const char\* v = "class Vehicle") – показва съответния клас
* Деструктор:
* ~Vehicle() { delete vehicle; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* void link(Vehicle\*) – нужен за съставяне на едносвързан списък
* void printlist(Vehicle\*) – нужен за съставяне на едносвързан списък

1. Клас **Car**

* Тип на класа: базов (производен за Vehicle)
* Наследява от: Vehicle
* Базов за: Sports, Performance
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* car – име на класа
* int speed – максимална скорост на превозното средство (km/h)
* int passengers – максимален брой пътници, който превозното средство може да побере
* Конструктори:
* void initialize(int, int) – инициализиращ конструктор
* Car(const char\* c = "class Car") – показва съответния клас
* Деструктор:
* ~Car() { delete car; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getSpeed(void) – виртуален метод, който взема и извежда зададената скорост
* virtual int getPassengers(void) – виртуален метод, който взема и извежда зададения брой пътници

1. Клас **Sports**

* Тип на класа: базов (производен за Car)
* Наследява от: Car
* Базов за: Luxury, Exotic
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* sports – име на класа
* int speed – максимална скорост на превозното средство (km/h)
* int passengers – максимален брой пътници, който превозното средство може да побере
* Конструктори:
* void initialize(int, int) – инициализиращ конструктор
* Sports(const char\* s = "class Sports") – показва съответния клас
* Деструктор:
* ~Sports() { delete sports; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getSpeed(void) – виртуален метод, който взема и извежда зададената скорост
* virtual int getPassengers(void) – виртуален метод, който взема и извежда зададения брой пътници

1. Клас **Luxury**

* Тип на класа: производен
* Наследява от: Sports
* Базов за: няма
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* luxury – име на класа
* int speed – максимална скорост на превозното средство (km/h)
* int passengers – максимален брой пътници, който превозното средство може да побере
* Конструктори:
* void initialize(int, int) – инициализиращ конструктор
* Luxury(const char\* l = "class Luxury") – показва съответния клас
* Деструктор:
* ~Luxury() { delete luxury; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getSpeed(void) – виртуален метод, който взема и извежда зададената скорост
* virtual int getPassengers(void) – виртуален метод, който взема и извежда зададения брой пътници

1. Клас **Exotic**

* Тип на класа: производен
* Наследява от: Sports
* Базов за: няма
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* exotic – име на класа
* int speed – максимална скорост на превозното средство (km/h)
* int passengers – максимален брой пътници, който превозното средство може да побере
* Конструктори:
* void initialize(int, int) – инициализиращ конструктор
* Exotic(const char\* e = "class Exotic") – показва съответния клас
* Деструктор:
* ~Exotic() { delete exotic; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getSpeed(void) – виртуален метод, който взема и извежда зададената скорост
* virtual int getPassengers(void) – виртуален метод, който взема и извежда зададения брой пътници

1. Клас **Performance**

* Тип на класа: базов (производен за Car)
* Наследява от: Car
* Базов за: Supercar, Hypercar
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* performance – име на класа
* int speed – максимална скорост на превозното средство (km/h)
* int passengers – максималнен брой пътници, който превозното средство може да побере
* Конструктори:
* void initialize(int, int) – инициализиращ конструктор
* Performance(const char\* p = "class Performance") – показва съответния клас
* Деструктор:
* ~Performance() { delete performance; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getSpeed(void) – виртуален метод, който взема и извежда зададената скорост
* virtual int getPassengers(void) – виртуален метод, който взема и извежда зададения брой пътници

1. Клас **Supercar**

* Тип на класа: производен
* Наследява от: Performance
* Базов за: няма
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* supercar – име на класа
* int speed – максимална скорост на превозното средство (km/h)
* int passengers – максимален брой пътници, който превозното средство може да побере
* Конструктори:
* void initialize(int, int) – инициализиращ конструктор
* Supercar(const char\* s = "class Supercar") – показва съответния клас
* Деструктор:
* ~Supercar() { delete supercar; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getSpeed(void) – виртуален метод, който взема и извежда зададената скорост
* virtual int getPassengers(void) – виртуален метод, който взема и извежда зададения брой пътници

1. Клас **Hypercar**

* Тип на класа: производен
* Наследява от: Performance
* Базов за: няма
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* hypercar – име на класа
* int speed – максимална скорост на превозното средство (km/h)
* int passengers – максимален брой пътници, който превозното средство може да побере
* Конструктори:
* void initialize(int, int) – инициализиращ конструктор
* Hypercar(const char\* h = "class Hypercar") – показва съответния клас
* Деструктор:
* ~Hypercar() { delete hypercar; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getSpeed(void) – виртуален метод, който взема и извежда зададената скорост
* virtual int getPassengers(void) – виртуален метод, който взема и извежда зададения брой пътници

1. Клас **Motorcycle**

* Тип на класа: производен
* Наследява от: Vehicle
* Базов за: няма
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* motorcycle – име на класа
* int wheels – брой колела на превозното средство
* float weight – тегло на превозното средство (kg)
* Конструктори:
* void initialize(int, float) – инициализиращ конструктор
* Motorcycle(const char\* m = "class Motorcycle") – показва съответния клас
* Деструктор:
* ~Motorcycle() { delete motorcycle; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getWheels(void) – виртуален метод, който взема и извежда зададения брой колела
* virtual float getWeight(void) – виртуален метод, който взема и извежда зададеното тегло

1. Клас **Truck**

* Тип на класа: производен
* Наследява от: Vehicle
* Базов за: няма
* Приятелски на: няма
* Има приятелски: няма

**Описание на характерните особености на класа:**

* Членове-данни (атрибути):
* char\* truck – име на класа
* int wheels – брой колела на превозното средство
* float weight – тегло на превозното средство (kg)
* Конструктори:
* void initialize(int, float) – инициализиращ конструктор
* Truck(const char\* t = "class Truck") – показва съответния клас
* Деструктор:
* ~Truck() { delete truck; } – освобождава динамичната памет
* Членове-функции (методи):
* virtual void print(void) – виртуален метод, връщащ името на класа
* virtual int getWheels(void) – виртуален метод, който взема и извежда зададения брой колела
* virtual float getWeight(void) – виртуален метод, който взема и извежда зададеното тегло

1. **Код на програмата**
2. Заглавни файлове:
   1. **===vehicle.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef VEHICLE\_H

#define VEHICLE\_H

// Vehicle

class Vehicle {

public:

Vehicle(const char\* v = "class Vehicle");

~Vehicle() { delete vehicle; }

virtual void print(void);

void link(Vehicle\*);

void printlist(Vehicle\*);

protected:

char\* vehicle;

Vehicle\* next;

};

// end Vehicle

#endif // !VEHICLE\_H

* 1. **===car.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef CAR\_H

#define CAR\_H

// Car

class Car : public Vehicle {

public:

void initialize(int, int);

virtual int getSpeed(void);

virtual int getPassengers(void);

Car(const char\* c = "class Car");

~Car() { delete car; }

virtual void print(void);

private:

char\* car;

int speed;

int passengers;

};

// end Car

#endif // !CAR\_H

* 1. **===sports.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef SPORTS\_H

#define SPORTS\_H

// Sports

class Sports : public Car {

public:

void initialize(int, int);

virtual int getSpeed(void);

virtual int getPassengers(void);

Sports(const char\* s = "class Sports");

~Sports() { delete sports; }

virtual void print(void);

private:

char\* sports;

int speed;

int passengers;

};

// end Sports

#endif // !SPORTS\_H

* 1. **===luxury.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef LUXURY\_H

#define LUXURY\_H

// Luxury

class Luxury : public Sports {

public:

void initialize(int, int);

virtual int getSpeed(void);

virtual int getPassengers(void);

Luxury(const char\* l = "class Luxury");

~Luxury() { delete luxury; }

virtual void print(void);

private:

char\* luxury;

int speed;

int passengers;

};

// end Luxury

#endif // !LUXURY\_H

* 1. **===exotic.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#ifndef EXOTIC\_H

#define EXOTIC\_H

// Exotic

class Exotic : public Sports {

public:

void initialize(int, int);

virtual int getSpeed(void);

virtual int getPassengers(void);

Exotic(const char\* e = "class Exotic");

~Exotic() { delete exotic; }

virtual void print(void);

private:

char\* exotic;

int speed;

int passengers;

};

// end Exotic

#endif // !EXOTIC\_H

* 1. **===performance.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef PERFORMANCE\_H

#define PERFORMANCE\_H

// Performance

class Performance : public Car {

public:

void initialize(int, int);

virtual int getSpeed(void);

virtual int getPassengers(void);

Performance(const char\* p = "class Performance");

~Performance() { delete performance; }

virtual void print(void);

private:

char\* performance;

int speed;

int passengers;

};

// end Performance

#endif // !PERFORMANCE\_H

* 1. **===supercar.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef SUPERCAR\_H

#define SUPERCAR\_H

// Supercar

class Supercar : public Performance {

public:

void initialize(int, int);

virtual int getSpeed(void);

virtual int getPassengers(void);

Supercar(const char\* s = "class Supercar");

~Supercar() { delete supercar; }

virtual void print(void);

private:

char\* supercar;

int speed;

int passengers;

};

// end Supercar

#endif // !SUPERCAR\_H

* 1. **===hypercar.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef HYPERCAR\_H

#define HYPERCAR\_H

// Hypercar

class Hypercar : public Performance {

public:

void initialize(int, int);

virtual int getSpeed(void);

virtual int getPassengers(void);

Hypercar(const char\* h = "class Hypercar");

~Hypercar() { delete hypercar; }

virtual void print(void);

private:

char\* hypercar;

int speed;

int passengers;

};

// end Hypercar

#endif // !HYPERCAR\_H

* 1. **===motorcycle.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef MOTORCYCLE\_H

#define MOTORCYCLE\_H

// Motorcycle

class Motorcycle : public Vehicle {

public:

void initialize(int, float);

virtual int getWheels(void);

virtual float getWeight(void);

Motorcycle(const char\* m = "class Motorcycle");

~Motorcycle() { delete motorcycle; }

virtual void print(void);

private:

char\* motorcycle;

int wheels;

float weight;

};

// end Motorcycle

#endif // !MOTORCYCLE\_H

* 1. **===truck.h===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#ifndef TRUCK\_H

#define TRUCK\_H

// Truck

class Truck : public Vehicle {

public:

void initialize(int, float);

virtual int getWheels(void);

virtual float getWeight(void);

Truck(const char\* t = "class Truck");

~Truck() { delete truck; }

virtual void print(void);

private:

char\* truck;

int wheels;

float weight;

};

// end Truck

#endif // !TRUCK\_H

1. Модули:
   1. **===vehicle.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include <iostream>

using namespace std;

// Vehicle

Vehicle::Vehicle(const char\* v) {

next = 0;

vehicle = new char[strlen(v) + 1];

strcpy(vehicle, v);

}

void Vehicle::print(void) {

cout << vehicle << endl;

}

void Vehicle::link(Vehicle\* pt) {

next = pt;

}

void Vehicle::printlist(Vehicle\* begin) {

for (Vehicle\* np = begin; np; np = np->next) {

np->print();

}

}

// end Vehicle

* 1. **===car.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "car.h"

#include <iostream>

using namespace std;

// Car

void Car::initialize(int inSpeed, int inPassengers) {

speed = inSpeed;

passengers = inPassengers;

}

int Car::getSpeed(void) {

return speed;

}

int Car::getPassengers(void) {

return passengers;

}

Car::Car(const char\* c) {

car = new char[strlen(c) + 1];

strcpy(car, c);

}

void Car::print(void) {

cout << car << endl;

}

// end Car

* 1. **===sports.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "car.h"

#include "sports.h"

#include <iostream>

using namespace std;

// Sports

void Sports::initialize(int inSpeed, int inPassengers) {

speed = inSpeed;

passengers = inPassengers;

}

int Sports::getSpeed(void) {

return speed;

}

int Sports::getPassengers(void) {

return passengers;

}

Sports::Sports(const char\* s) {

sports = new char[strlen(s) + 1];

strcpy(sports, s);

}

void Sports::print(void) {

cout << sports << endl;

}

// end Sports

* 1. **===luxury.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "car.h"

#include "sports.h"

#include "luxury.h"

#include <iostream>

using namespace std;

// Luxury

void Luxury::initialize(int inSpeed, int inPassengers) {

speed = inSpeed;

passengers = inPassengers;

}

int Luxury::getSpeed(void) {

return speed;

}

int Luxury::getPassengers(void) {

return passengers;

}

Luxury::Luxury(const char\* l) {

luxury = new char[strlen(l) + 1];

strcpy(luxury, l);

}

void Luxury::print(void) {

cout << luxury << endl;

}

// end Luxury

* 1. **===exotic.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "car.h"

#include "sports.h"

#include "exotic.h"

#include <iostream>

using namespace std;

// Exotic

void Exotic::initialize(int inSpeed, int inPassengers) {

speed = inSpeed;

passengers = inPassengers;

}

int Exotic::getSpeed(void) {

return speed;

}

int Exotic::getPassengers(void) {

return passengers;

}

Exotic::Exotic(const char\* e) {

exotic = new char[strlen(e) + 1];

strcpy(exotic, e);

}

void Exotic::print(void) {

cout << exotic << endl;

}

// end Exotic

* 1. **===performance.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "car.h"

#include "performance.h"

#include <iostream>

using namespace std;

// Performance

void Performance::initialize(int inSpeed, int inPassengers) {

speed = inSpeed;

passengers = inPassengers;

}

int Performance::getSpeed(void) {

return speed;

}

int Performance::getPassengers(void) {

return passengers;

}

Performance::Performance(const char\* p) {

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

strcpy(performance, p);

}

void Performance::print(void) {

cout << performance << endl;

}

// end Performance

* 1. **===supercar.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "car.h"

#include "performance.h"

#include "supercar.h"

#include <iostream>

using namespace std;

// Supercar

void Supercar::initialize(int inSpeed, int inPassengers) {

speed = inSpeed;

passengers = inPassengers;

}

int Supercar::getSpeed(void) {

return speed;

}

int Supercar::getPassengers(void) {

return passengers;

}

Supercar::Supercar(const char\* s) {

supercar = new char[strlen(s) + 1];

strcpy(supercar, s);

}

void Supercar::print(void) {

cout << supercar << endl;

}

// end Supercar

* 1. **===hypercar.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "car.h"

#include "performance.h"

#include "hypercar.h"

#include <iostream>

using namespace std;

// Hypercar

void Hypercar::initialize(int inSpeed, int inPassengers) {

speed = inSpeed;

passengers = inPassengers;

}

int Hypercar::getSpeed(void) {

return speed;

}

int Hypercar::getPassengers(void) {

return passengers;

}

Hypercar::Hypercar(const char\* h) {

hypercar = new char[strlen(h) + 1];

strcpy(hypercar, h);

}

void Hypercar::print(void) {

cout << hypercar << endl;

}

// end Hypercar

* 1. **===motorcycle.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "motorcycle.h"

#include <iostream>

using namespace std;

// Motorcycle

void Motorcycle::initialize(int inWheels, float inWeight) {

wheels = inWheels;

weight = inWeight;

}

int Motorcycle::getWheels(void) {

return wheels;

}

float Motorcycle::getWeight(void) {

return weight;

}

Motorcycle::Motorcycle(const char\* m) {

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

strcpy(motorcycle, m);

}

void Motorcycle::print(void) {

cout << motorcycle << endl;

}

// end Motorcycle

* 1. **===truck.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "truck.h"

#include <iostream>

using namespace std;

// Truck

void Truck::initialize(int inWheels, float inWeight) {

wheels = inWheels;

weight = inWeight;

}

int Truck::getWheels(void) {

return wheels;

}

float Truck::getWeight(void) {

return weight;

}

Truck::Truck(const char\* t) {

truck = new char[strlen(t) + 1];

strcpy(truck, t);

}

void Truck::print(void) {

cout << truck << endl;

}

// end Truck

* 1. **===main.cpp===**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "vehicle.h"

#include "car.h"

#include "sports.h"

#include "luxury.h"

#include "exotic.h"

#include "performance.h"

#include "supercar.h"

#include "hypercar.h"

#include "motorcycle.h"

#include "truck.h"

#include <iostream>

using namespace std;

// main

void main() {

Vehicle\* vehicle = new Vehicle;

Vehicle\* car = new Car;

Vehicle\* sports = new Sports;

Vehicle\* luxury = new Luxury;

Vehicle\* exotic = new Exotic;

Vehicle\* performance = new Performance;

Vehicle\* supercar = new Supercar;

Vehicle\* hypercar = new Hypercar;

Vehicle\* motorcycle = new Motorcycle;

Vehicle\* truck = new Truck;

Motorcycle scooter;

scooter.initialize(2, 45.5);

cout << "Skutera e ot "; motorcycle->print();

cout << "Skutera ima " << scooter.getWheels() << " kolela" << endl;

cout << "Skutera teji " << scooter.getWeight() << " kg\n\n";

Truck scania;

scania.initialize(6, 9645.18);

cout << "Scania e ot "; truck->print();

cout << "Scania ima " << scania.getWheels() << " kolela" << endl;

cout << "Scania teji " << scania.getWeight() << " kg\n\n";

Exotic aston\_martin;

aston\_martin.initialize(340, 2);

cout << "Aston Martin e ot "; exotic->print();

cout << "Aston Martin vdiga " << aston\_martin.getSpeed() << " km/h" << endl;

cout << "Aston Martin pobira " << aston\_martin.getPassengers() << " dushi\n\n";

cout << "Spisuk ot klasovete v yerarhiyata:" << endl;

vehicle->link(car);

car->link(sports);

sports->link(luxury);

luxury->link(exotic);

exotic->link(performance);

performance->link(supercar);

supercar->link(hypercar);

hypercar->link(motorcycle);

motorcycle->link(truck);

vehicle->printlist(vehicle);

delete vehicle;

delete car;

delete sports;

delete luxury;

delete exotic;

delete performance;

delete supercar;

delete hypercar;

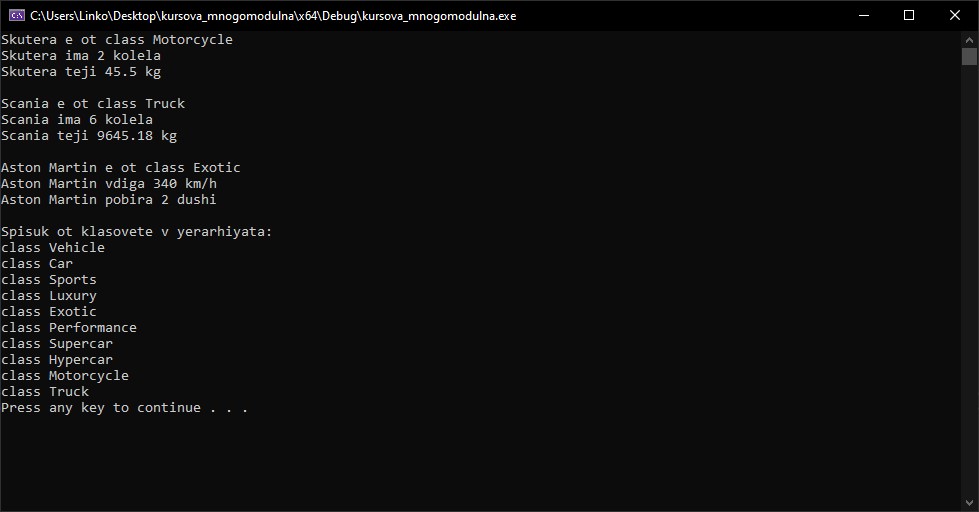
delete motorcycle;

delete truck;

system("pause");

}

// end main

1. **Тестване на програмата**