МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Объектно-ориентированное программирование

Лабораторная работа 9

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОТНОШЕНИЯ

МНОЖЕСТВЕННОГО НАСЛЕДОВАНИЯ

Выполнил:

студент гр. 1ПИб-01-21оп Викторов Д.А.

Проверил:

зав. кафедрой Ершов Е.В.

Череповец, 2020 г.

Цель работы: изучить принципы и механизмы множественного наследования, правила доступа к базовым классам; приобрести практические навыки работы с базовыми и производными классами при множественном наследовании.

Задания

1. Создайте производный класс для АТД, реализованного по заданию лабораторной работы 7, используя одиночное наследование.

2. Проверьте работоспособность АТД и производного класса на тестовом наборе данных.

Контрольные вопросы

1. С какой целью и в каких случаях используется множественное наследование?

Множественное наследование возможным получение производного класса от нескольких базовых классов. Синтаксис заголовка класса расширяется, чтобы можно было использовать список базовых классов с атрибутами доступа.  свойство, поддерживаемое частью [объектно-ориентированных](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [языков программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), когда [класс](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) может иметь более одного класса-родителя, интерфейсы поддерживают множественное наследование. Эта концепция является расширением простого [наследования](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), при котором класс может наследоваться только от одного суперкласса.

1. Опишите синтаксис заголовка производного класса при множественном наследовании.

Class {имя производного класса} : {тип наследования} {имя класса родителя}, {тип наследования} {имя класса родителя}…

1. Дайте определение ориентированного ациклического графа.

Граф, узлы которого являются классами, а ориентированные ребра направлены от производных классов к базовым.

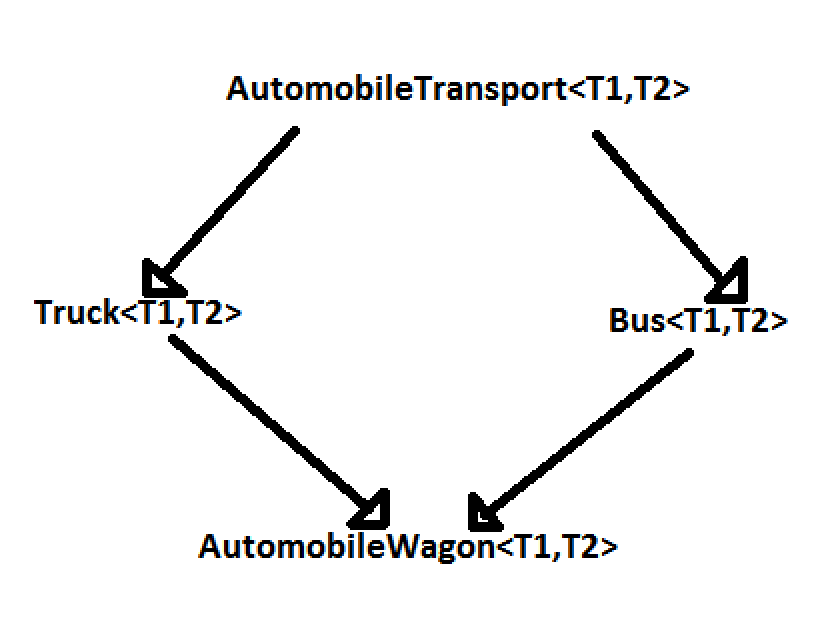
1. Где инициируются виртуальные базовые классы?

Виртуальные базовые классы инициализируются (вызывается void-конструктор) перед любыми невиртуальными базовыми классами и в том порядке, в котором они появляются в ПАГе наследования при просмотре его снизу-вверх и слева направо.

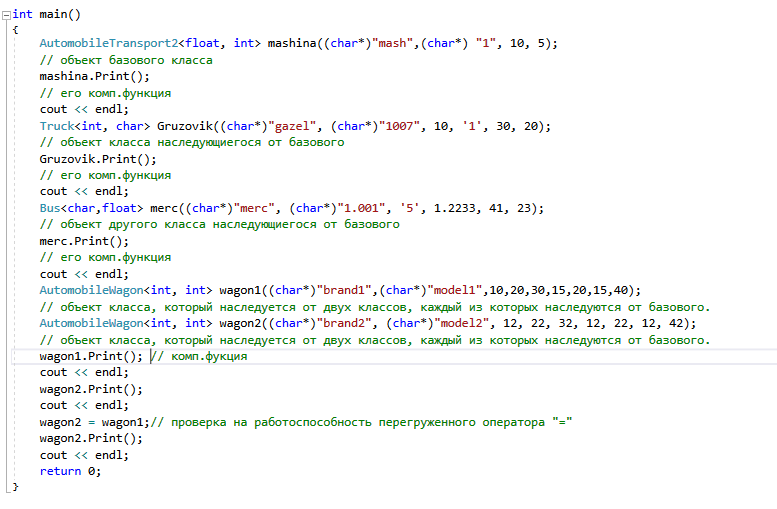
1. Что будет, если из объявлений классов Cow и Buffalo убрать ключевое слово virtual (см. последний пример в теоретических положениях)?

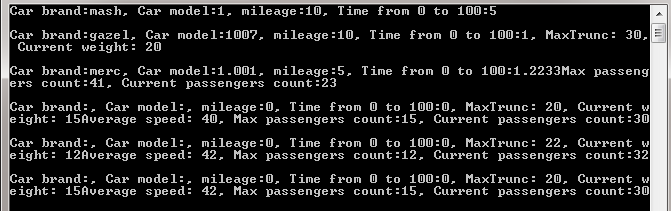
Многократное копирование полей из предков класса.

Схема наследований:



Тесты:





Код:

#include "pch.h"

#include <iostream>

#include <cstdlib>

using namespace std;

template <class T1, class T2>

class AutomobileTransport2 { // базовый класс

public:

void virtual Print();

AutomobileTransport2();

AutomobileTransport2(char\* CarBrand, char\* CarModel, T1 km, T2 time);

AutomobileTransport2(const AutomobileTransport2<T1, T2> &a);

~AutomobileTransport2();

bool Equal(const  AutomobileTransport2<T1, T2>& b);

void Copy(const  AutomobileTransport2<T1, T2>& a);

AutomobileTransport2<T1, T2>Summa(const  AutomobileTransport2<T1, T2>& c);

 AutomobileTransport2<T1, T2>operator =(const  AutomobileTransport2<T1, T2>& a);

AutomobileTransport2<T1, T2>operator +(const  AutomobileTransport2<T1, T2>& c);

bool operator >(const  AutomobileTransport2<T1, T2>& a);

bool operator <(const  AutomobileTransport2<T1, T2>& a);

bool operator ==(const  AutomobileTransport2<T1, T2>& a);

bool operator !=(const  AutomobileTransport2<T1, T2>& a);

bool operator >=(const  AutomobileTransport2<T1, T2>& a);

bool operator <=(const  AutomobileTransport2<T1, T2>& a);

protected: // исправлено: Добавлены протектед поля

char\* carModel;

char\* carBrand;

T1 mileage;

T2 time0To100;

};

template <class T1, class T2> // наследуемый класс

class Truck : public virtual  AutomobileTransport2<T1, T2> { //одиночное наследование

public:

Truck(const Truck<T1, T2> &a); //копирующий конструктор

Truck<T1, T2>operator =(const  Truck<T1, T2>& a); // перегрузка оператора "равно"

Truck() // конструктор по умолчанию

:AutomobileTransport2<T1, T2>::AutomobileTransport2()

{

maxTruncCar = 0;

currentWeight = 0;

};

Truck(char\* CarBrand, char\* CarModel, T1 km, T2 time, int MaxTruncCar, int CurrentWeight) // конструктор с параметрами

:AutomobileTransport2<T1, T2>::AutomobileTransport2(CarBrand, CarModel, km, time)

{

maxTruncCar = MaxTruncCar;

currentWeight = CurrentWeight;

};

void Print() //компонетная функция

{

AutomobileTransport2<T1, T2>::Print();

cout << ", MaxTrunc: " << maxTruncCar << ", Current weight: " << currentWeight;

}

protected: // поля принадлежащие этому классу

int maxTruncCar;

int currentWeight;

};

template<class T1, class T2>

Truck<T1, T2>::Truck(const Truck<T1, T2>& a)

{

Truck<T1, T2>::time0To100 = a.Truck<T1, T2>::time0To100;

Truck<T1, T2>::mileage = a.Truck<T1, T2>::mileage;

Truck<T1, T2>::maxTruncCar = a.Truck<T1, T2>::maxTruncCar;

Truck<T1, T2>::currentWeight = a.Truck<T1, T2>::currentWeight;

Truck<T1, T2>::carBrand = new char[strlen(a.Truck<T1, T2>::carBrand) + 1];

strcpy(Truck<T1, T2>::carBrand, a.Truck<T1, T2>::carBrand);

Truck<T1, T2>::carModel = new char[strlen(a.Truck<T1, T2>::carModel) + 1];

strcpy(Truck<T1, T2>::carModel, a.Truck<T1, T2>::carModel);

}

template<class T1, class T2>

Truck<T1, T2> Truck<T1, T2>::operator =(const Truck<T1, T2> &a)

{

if (this == &a)

return \*this;

Truck<T1, T2>::time0To100 = a.Truck<T1, T2>::time0To100;

Truck<T1, T2>::mileage = a.Truck<T1, T2>::mileage;

Truck<T1, T2>::maxTruncCar = a.Truck<T1, T2>::maxTruncCar;

Truck<T1, T2>::currentWeight = a.Truck<T1, T2>::currentWeight;

Truck<T1, T2>::carBrand = new char[strlen(a.Truck<T1, T2>::carBrand) + 1];

strcpy(Truck<T1, T2>::carBrand, a.Truck<T1, T2>::carBrand);

Truck<T1, T2>::carModel = new char[strlen(a.Truck<T1, T2>::carModel) + 1];

strcpy(Truck<T1, T2>::carModel, a.Truck<T1, T2>::carModel);

return \*this;

}

template <class T1, class T2>

class Bus : public virtual  AutomobileTransport2<T1, T2> { // одиночное наследование

public:

Bus(const Bus<T1, T2> &a)

{

Bus<T1, T2>::time0To100 = a.Bus<T1, T2>::time0To100;

Bus<T1, T2>::mileage = a.Bus<T1, T2>::mileage;

Bus<T1, T2>::maxPassengers = a.Bus<T1, T2>::maxTruncCar;

Bus<T1, T2>::currentPassengers = a.Bus<T1, T2>::currentPassengers;

Bus<T1, T2>::carBrand = new char[strlen(a.Bus<T1, T2>::carBrand) + 1];

strcpy(Bus<T1, T2>::carBrand, a.Bus<T1, T2>::carBrand);

Bus<T1, T2>::carModel = new char[strlen(a.Bus<T1, T2>::carModel) + 1];

strcpy(Bus<T1, T2>::carModel, a.Bus<T1, T2>::carModel);

};//копирующий конструктор

Bus<T1, T2>operator =(const  Bus<T1, T2>& a)

{

if (this == &a)

return \*this;

Bus<T1, T2>::time0To100 = a.Bus<T1, T2>::time0To100;

Bus<T1, T2>::mileage = a.Bus<T1, T2>::mileage;

Bus<T1, T2>::maxPassengers = a.Bus<T1, T2>::maxPassengers;

Bus<T1, T2>::currentPassengers = a.Bus<T1, T2>::currentPassengers;

Bus<T1, T2>::carBrand = new char[strlen(a.Bus<T1, T2>::carBrand) + 1];

strcpy(Bus<T1, T2>::carBrand, a.Bus<T1, T2>::carBrand);

Bus<T1, T2>::carModel = new char[strlen(a.Bus<T1, T2>::carModel) + 1];

strcpy(Bus<T1, T2>::carModel, a.Bus<T1, T2>::carModel);

return \*this;

}

; // перегрузка оператора "равно"

Bus() // конструктор по умолчанию

:AutomobileTransport2<T1, T2>::AutomobileTransport2()

{

maxPassengers = 0;

currentPassengers = 0;

};

Bus(char\* CarBrand, char\* CarModel, T1 km, T2 time, int MaxPassengers, int CurrentPassengers) // конструктор с параметрами

:AutomobileTransport2<T1, T2>::AutomobileTransport2(CarBrand, CarModel, km, time)

{

maxPassengers = MaxPassengers;

currentPassengers = CurrentPassengers;

};

void Print() // компонетная функция

{

AutomobileTransport2<T1, T2>::Print();

cout << "Max passengers count:" << maxPassengers << ", Current passengers count:"

<< currentPassengers;

}

protected: // поля принадлежащие этому классу

int maxPassengers;

int currentPassengers;

};

// Множественное наследование

template <class T1, class T2>

class AutomobileWagon : public Bus<T1, T2>, public Truck<T1, T2> {

public:

AutomobileWagon() :Truck<T1, T2>::Truck(), Bus()  // конструктор по умолчанию

{

averageSpeed = 0;

};

AutomobileWagon(char\* CarBrand, char\* CarModel, T1 km, T2 time, int MaxPassengers, int CurrentPassengers, int MaxTruncCar, int CurrentWeight, int AverageSpeed) // конструктор с параметрами

:Truck<T1, T2>::Truck(CarBrand, CarModel, km, time, MaxTruncCar, CurrentWeight), Bus<T1, T2>::Bus(CarBrand, CarModel, km, time, CurrentPassengers, MaxPassengers)

{

averageSpeed = AverageSpeed;

};

AutomobileWagon(const AutomobileWagon<T1, T2> &a)

{

AutomobileWagon<T1, T2>::time0To100 = a.AutomobileWagon<T1, T2>::time0To100;

AutomobileWagon<T1, T2>::mileage = a.AutomobileWagon<T1, T2>::mileage;

AutomobileWagon<T1, T2>::maxTruncCar = a.AutomobileWagon<T1, T2>::maxTruncCar;

AutomobileWagon<T1, T2>::currentWeight = a.AutomobileWagon<T1, T2>::currentWeight;

AutomobileWagon<T1, T2>::maxPassengers = a.AutomobileWagon<T1, T2>::maxTruncCar;

AutomobileWagon<T1, T2>::currentPassengers = a.AutomobileWagon<T1, T2>::currentPassengers;

AutomobileWagon<T1, T2>::carBrand = new char[strlen(a.AutomobileWagon<T1, T2>::carBrand) + 1];

strcpy(AutomobileWagon<T1, T2>::carBrand, a.AutomobileWagon<T1, T2>::carBrand);

AutomobileWagon<T1, T2>::carModel = new char[strlen(a.AutomobileWagon<T1, T2>::carModel) + 1];

strcpy(AutomobileWagon<T1, T2>::carModel, a.AutomobileWagon<T1, T2>::carModel);

};//копирующий конструктор

AutomobileWagon<T1, T2>operator =(const  AutomobileWagon<T1, T2>& a) // перегруженный оператор"="

{

if (this == &a)

return \*this;

AutomobileWagon<T1, T2>::time0To100 = a.AutomobileWagon<T1, T2>::time0To100;

AutomobileWagon<T1, T2>::mileage = a.AutomobileWagon<T1, T2>::mileage;

AutomobileWagon<T1, T2>::maxPassengers = a.AutomobileWagon<T1, T2>::maxPassengers;

AutomobileWagon<T1, T2>::maxTruncCar = a.AutomobileWagon<T1, T2>::maxTruncCar;

AutomobileWagon<T1, T2>::currentWeight = a.AutomobileWagon<T1, T2>::currentWeight;

AutomobileWagon<T1, T2>::currentPassengers = a.AutomobileWagon<T1, T2>::currentPassengers;

AutomobileWagon<T1, T2>::carBrand = new char[strlen(a.AutomobileWagon<T1, T2>::carBrand) + 1];

strcpy(AutomobileWagon<T1, T2>::carBrand, a.AutomobileWagon<T1, T2>::carBrand);

AutomobileWagon<T1, T2>::carModel = new char[strlen(a.AutomobileWagon<T1, T2>::carModel) + 1];

strcpy(AutomobileWagon<T1, T2>::carModel, a.AutomobileWagon<T1, T2>::carModel);

return \*this;

}

void Print() // компонентная функция

{

Truck<T1, T2>::Print();

cout << "Average speed: " << averageSpeed << ", Max passengers count:" << AutomobileWagon<T1, T2>::maxPassengers << ", Current passengers count:"

<< AutomobileWagon<T1, T2>::currentPassengers;

}

protected:

int averageSpeed;

};

int main()

{

AutomobileTransport2<float, int> mashina((char\*)"mash",(char\*) "1", 10, 5); // объект базового класса

mashina.Print(); // его комп.функция

cout << endl;

Truck<int, char> Gruzovik((char\*)"gazel", (char\*)"1007", 10, '1', 30, 20); // объект класса наследующиегося от базового

Gruzovik.Print(); // его комп.функция

cout << endl;

Bus<char,float> merc((char\*)"merc", (char\*)"1.001", '5', 1.2233, 41, 23); // объект другого класса наследующиегося от базового

merc.Print(); // его комп.функция

cout << endl;

AutomobileWagon<int, int> wagon1((char\*)"brand1",(char\*)"model1",10,20,30,15,20,15,40); // объект класса, который наследуется от двух классов, каждый из которых наследуются от базового.

AutomobileWagon<int, int> wagon2((char\*)"brand2", (char\*)"model2", 12, 22, 32, 12, 22, 12, 42);// объект класса, который наследуется от двух классов, каждый из которых наследуются от базового.

wagon1.Print(); // комп.фукция

cout << endl;

wagon2.Print();

cout << endl;

wagon2 = wagon1;// проверка на работоспособность перегруженного оператора "="

wagon2.Print();

cout << endl;

return 0;

}

template<class T1, class T2>

void AutomobileTransport2<T1, T2>::Print()

{

cout << "Car brand:" << carBrand << ", Car model:" << carModel << ", mileage:" << mileage << ", Time from 0 to 100:" << time0To100;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2>::AutomobileTransport2()

{

carBrand = new char[1];

strcpy(carBrand, "");

carModel = new char[1];

strcpy(carModel, "");

mileage = (T1)0;

time0To100 = (T2)0;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2>::AutomobileTransport2(const AutomobileTransport2<T1, T2>& a)

{

if (&a != this) {

carBrand = new char[strlen(a.carBrand) + 1];

strcpy(carBrand, a.carBrand);

carModel = new char[strlen(a.carModel) + 1];

strcpy(carModel, a.carModel);

mileage = a.mileage;

time0To100 = a.time0To100;

}

}

template<class T1, class T2>

AutomobileTransport2<T1, T2>::~AutomobileTransport2()

{

delete[] carBrand;

delete[] carModel;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::Equal(const AutomobileTransport2<T1, T2>& b)

{

return mileage == b.mileage && time0To100 == b.time0To100;

}

template<class T1, class T2>

void AutomobileTransport2<T1, T2>::Copy(const AutomobileTransport2<T1, T2>& a)

{

carBrand = new char[strlen(a.carBrand) + 1];

strcpy(carBrand, a.carBrand);

carModel = new char[strlen(a.carModel) + 1];

strcpy(carModel, a.carModel);

mileage = a.mileage;

time0To100 = a.time0To100;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2> AutomobileTransport2<T1, T2>::Summa(const AutomobileTransport2<T1, T2>& c)

{

AutomobileTransport2<T1, T2> t;

char\* temp1 = new char[strlen(carBrand) + 1];

char\* temp2 = new char[strlen(carModel) + 1];

strcpy(temp1, carBrand);

strcpy(temp2, carModel);

t.mileage = mileage + c.mileage;

t.time0To100 = time0To100 + c.time0To100;

strcat(temp1, c.carBrand);

strcat(temp2, c.carModel);

t.carBrand = new char[strlen(temp1) + 1];

strcpy(t.carBrand, temp1);

t.carModel = new char[strlen(temp2) + 1];

strcpy(t.carModel, temp2);

return (t);

}

template<class T1, class T2>

AutomobileTransport2<T1, T2> AutomobileTransport2<T1, T2>::operator=(const AutomobileTransport2<T1, T2>& a)

{

if (this == &a)

return \*this;

time0To100 = a.time0To100;

mileage = a.mileage;

carBrand = new char[strlen(a.carBrand) + 1];

strcpy(carBrand, a.carBrand);

carModel = new char[strlen(a.carModel) + 1];

strcpy(carModel, a.carModel);

return \*this;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2> AutomobileTransport2<T1, T2>::operator+(const AutomobileTransport2<T1, T2>& c)

{

AutomobileTransport2<T1, T2> t;

char\* temp1 = new char[strlen(carBrand) + 1];

char\* temp2 = new char[strlen(carModel) + 1];

strcpy(temp1, carBrand);

strcpy(temp2, carModel);

t.mileage = mileage + c.mileage;

t.time0To100 = time0To100 + c.time0To100;

strcat(temp1, c.carBrand);

strcat(temp2, c.carModel);

t.carBrand = new char[strlen(temp1) + 1];

strcpy(t.carBrand, temp1);

t.carModel = new char[strlen(temp2) + 1];

strcpy(t.carModel, temp2);

return (t);

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator>(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 > a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator<(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 < a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator==(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 == a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator!=(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 != a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator>=(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 >= a.mileage + a.time0To100;

}

template<class T1, class T2>

bool AutomobileTransport2<T1, T2>::operator<=(const AutomobileTransport2<T1, T2>& a)

{

return mileage + time0To100 <= a.mileage + a.time0To100;

}

template<class T1, class T2>

AutomobileTransport2<T1, T2>::AutomobileTransport2(char \* CarBrand, char \* CarModel, T1 km, T2 time)

{

carBrand = new char[strlen(CarBrand) + 1];

strcpy(carBrand, CarBrand);

carModel = new char[strlen(CarModel) + 1];

strcpy(carModel, CarModel);

mileage = km;

time0To100 = time;

}