МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Объектно-ориентированное программирование

ЛАБОРАТОРНАЯ РАБОТА № 6

Выполнил:

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

Проверил:

зав. кафедрой Ершов Е.В.

Череповец, 2020 г.

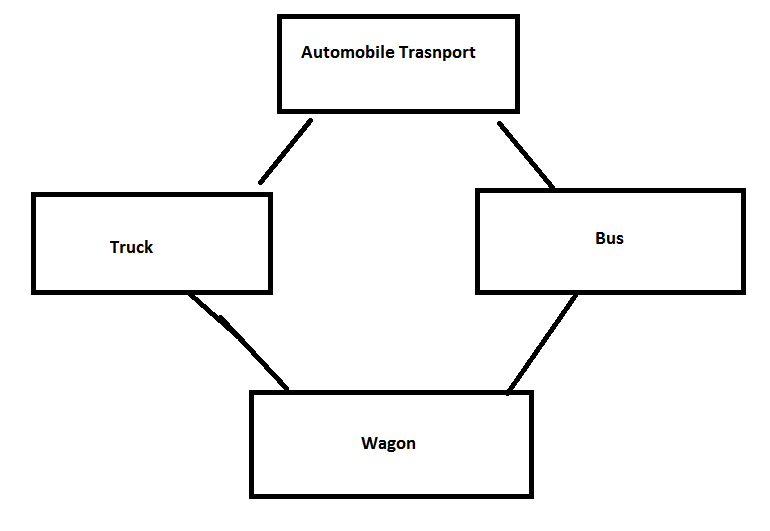
Цель работы: изучить принципы и механизмы множественного наследования, правила доступа к базовым классам; приобрести практические навыки работы с базовыми и производными классами при множественном наследовании.

Задания

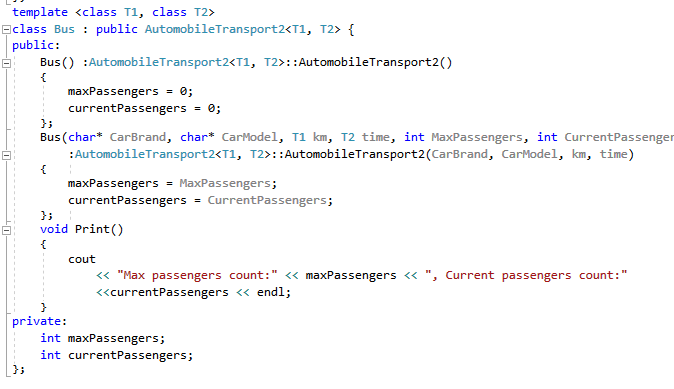
1. Создайте производный класс для АТД, реализованного по заданию лабораторной работы 7, используя одиночное наследование.

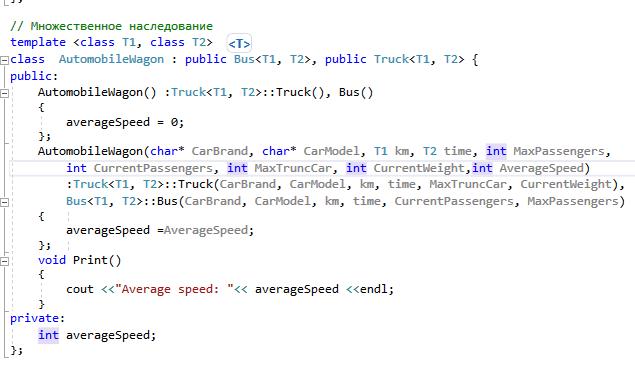
2. Проверьте работоспособность АТД и производного класса на тестовом наборе данных.

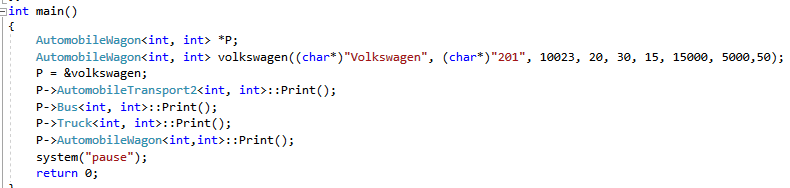
Схема наследований:



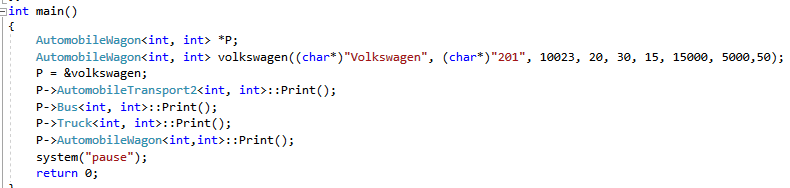
Изменения проведённые из лабораторной работы 8 для получения результата необходимого для 9 лабораторной работы:

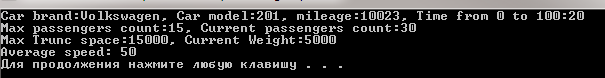






Тесты:





Код:

#include "pch.h"

#include <iostream>

using namespace std;

template <class T1, class T2>

class AutomobileTransport2 {

public:

void 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);

private:

char\* carBrand;

char\* carModel;

T1 mileage;

T2 time0To100;

};

template <class T1, class T2>

class Truck : public AutomobileTransport2<T1, T2> {

public:

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

{

maxTruncCar = 0;

currentWeight = 0;

};

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

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

{

maxTruncCar = MaxTruncCar;

currentWeight = CurrentWeight;

};

void Print()

{

cout << "Max Trunc space:" <<maxTruncCar

<< ", Current Weight:" << currentWeight << endl;

}

private:

int maxTruncCar;

int currentWeight;

};

template <class T1, class T2>

class Bus : public AutomobileTransport2<T1, T2> {

public:

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

{

maxPassengers = 0;

currentPassengers = 0;

};

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

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

{

maxPassengers = MaxPassengers;

currentPassengers = CurrentPassengers;

};

void Print()

{

cout

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

<<currentPassengers << endl;

}

private:

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()

{

averageSpeed = 0;

};

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

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

{

averageSpeed =AverageSpeed;

};

void Print()

{

cout <<"Average speed: "<< averageSpeed <<endl;

}

private:

int averageSpeed;

};

template <class T>

class AutomobileTransport1

{

public:

void Print();

AutomobileTransport1();

AutomobileTransport1(char\* CarBrand, char\* CarModel, T km, float time);

AutomobileTransport1(const AutomobileTransport1<T>&a);

~AutomobileTransport1();

bool Equal(const  AutomobileTransport1<T>& b);

void Copy(const  AutomobileTransport1<T>& a);

AutomobileTransport1<T> Summa(const  AutomobileTransport1<T>& c);

AutomobileTransport1<T> operator =(const  AutomobileTransport1<T>& a);

AutomobileTransport1<T> operator +(const  AutomobileTransport1<T>& a);

bool operator >(const  AutomobileTransport1<T>& a);

bool operator <(const  AutomobileTransport1<T>& a);

bool operator ==(const  AutomobileTransport1<T>& a);

bool operator !=(const  AutomobileTransport1<T>& a);

bool operator >=(const  AutomobileTransport1<T>& a);

bool operator <=(const  AutomobileTransport1<T>& a);

private:

char\* carBrand;

char\* carModel;

T mileage;

float time0To100;

};

int main()

{

AutomobileWagon<int, int> \*P;

AutomobileWagon<int, int> volkswagen((char\*)"Volkswagen", (char\*)"201", 10023, 20, 30, 15, 15000, 5000,50);

P = &volkswagen;

P->AutomobileTransport2<int, int>::Print();

P->Bus<int, int>::Print();

P->Truck<int, int>::Print();

P->AutomobileWagon<int,int>::Print();

system("pause");

return 0;

}

template <class T>

AutomobileTransport1<T>::AutomobileTransport1()

{

carBrand = new char[1];

strcpy(carBrand, "");

carModel = new char[1];

strcpy(carModel, "");

mileage = (T)0;

time0To100 = 0;

}

template <class T>

AutomobileTransport1<T>::AutomobileTransport1(char \* CarBrand, char \* CarModel, T km, float time)

{

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

strcpy(carBrand, CarBrand);

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

strcpy(carModel, CarModel);

mileage = km;

time0To100 = time;

}

template <class T>

AutomobileTransport1<T>::AutomobileTransport1(const AutomobileTransport1<T> & 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 T>

AutomobileTransport1<T>::~AutomobileTransport1()

{

delete[] carBrand;

delete[] carModel;

}

template <class T>

void AutomobileTransport1<T>::Print()

{

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

}

template <class T>

bool AutomobileTransport1<T>::Equal(const AutomobileTransport1<T>& b)

{

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

}

template <class T>

void AutomobileTransport1<T>::Copy(const AutomobileTransport1<T>& 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 T>

AutomobileTransport1<T>  AutomobileTransport1<T>::Summa(const AutomobileTransport1<T>& c)

{

AutomobileTransport1<T> 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 T>

AutomobileTransport1<T> AutomobileTransport1<T>::operator=(const AutomobileTransport1<T> & 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 T>

AutomobileTransport1<T> AutomobileTransport1<T>::operator+(const AutomobileTransport1<T> & c)

{

AutomobileTransport1<T> 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 T>

bool AutomobileTransport1<T>::operator>(const AutomobileTransport1<T> & a)

{

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

}

template <class T>

bool AutomobileTransport1<T>::operator<(const AutomobileTransport1<T> & a)

{

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

}

template <class T>

bool AutomobileTransport1<T>::operator==(const AutomobileTransport1<T> & a)

{

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

}

template <class T>

bool AutomobileTransport1<T>::operator!=(const AutomobileTransport1<T> & a)

{

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

}

template <class T>

bool AutomobileTransport1<T>::operator>=(const AutomobileTransport1<T> & a)

{

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

}

template <class T>

bool AutomobileTransport1<T>::operator<=(const AutomobileTransport1<T> & a)

{

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

}

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 << endl;

}

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;

}