МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Объектно-ориентированное программирование

*Лабораторная работа* *8*

## ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОТНОШЕНИЯ

## ПРОСТОГО НАСЛЕДОВАНИЯ

Выполнил:

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

Проверил:

зав. кафедрой Ершов Е.В.

Череповец, 2020 г.

Цель работы: изучить способы создания производного класса и особенности работы с ним, правила инициализации и доступа к элементам производного класса; приобрести практические навыки наследования.

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 Дайте определение наследования.

Наследование (inheritance) – это механизм получения нового класса на основе существующего класса. Существующий класс может быть дополнен или изменен для создания производного класса.

2 Какие модификаторы прав доступа к производному классу вы знаете и в чем их особенности?

Модификаторы прав доступа к членам класса (public, protected и private) могут применяться в объявлении класса в любом порядке и сколько угодно раз. Открытый член доступен во всей области видимости, где виден класс. Закрытый член доступен другим функциям-членам своего класса. Защищенный член доступен не только другим функциям-членам своего класса, но и функциям-членам класса, унаследованного непосредственно от данного класса.

3 Как выполняется конструктор при наследовании?

При наследовании и инициализации членов класса конструкторы выполняются в следующем порядке:

Базовые классы инициализируются в порядке объявления.

Члены инициализируются в порядке объявления.

Виртуальные базовые классы создаются до того, как создан любой из производных классов, и до того, как созданы невиртуальные базовые классы. Порядок их создания – «из глубины, слева направо». Деструкторы вызываются в обратном выполнению конструкторов порядке.

4 Дайте определение чисто виртуальной функции.

Чисто виртуальная функция – это виртуальная функция-член, тело которой не определено. Она объявляется внутри класса:

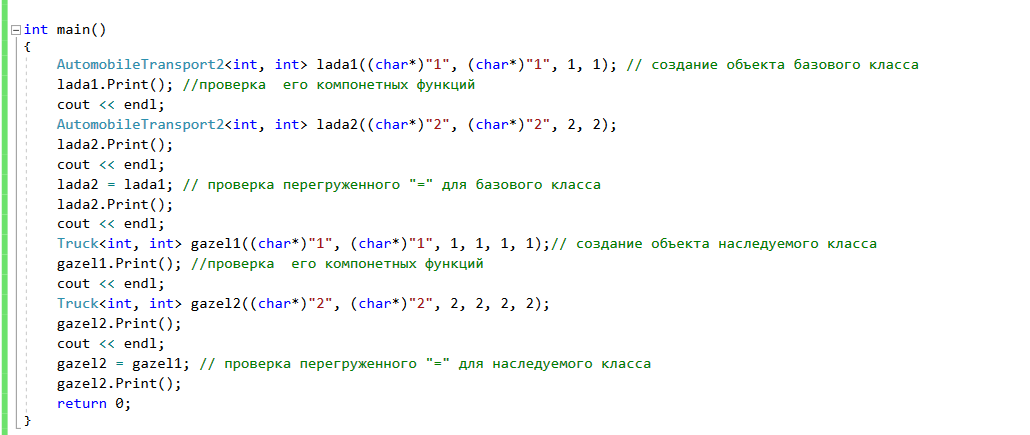
virtual прототип\_функции = 0;

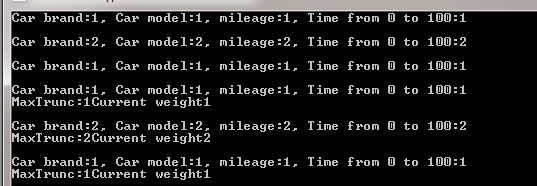
5 Какие операторы используются для разыменования указателя на член класса?

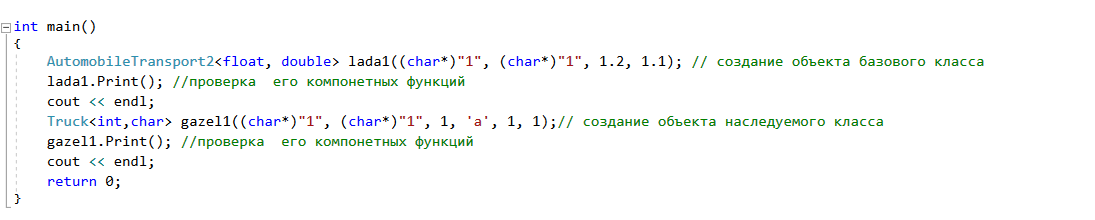
Для разыменования указателя на член класса используются два оператора: .\* и →\*.

Тесты:

1)





2)



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 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 << endl;

}

protected:  // защищённые 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;

}

int main()

{

AutomobileTransport2<int, int> lada1((char\*)"1", (char\*)"1", 1, 1);

 // создание объекта базового класса

lada1.Print(); //проверка  его компонетных функций

cout << endl;

AutomobileTransport2<int, int> lada2((char\*)"2", (char\*)"2", 2, 2);

lada2.Print();

cout << endl;

lada2 = lada1; // проверка перегруженного "=" для базового класса

lada2.Print();

cout << endl;

Truck<int, int> gazel1((char\*)"1", (char\*)"1", 1, 1, 1, 1);// создание объекта наследуемого класса

gazel1.Print(); //проверка  его компонетных функций

cout << endl;

Truck<int, int> gazel2((char\*)"2", (char\*)"2", 2, 2, 2, 2);

gazel2.Print();

cout << endl;

gazel2 = gazel1; // проверка перегруженного "=" для наследуемого класса

gazel2.Print();

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 << 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;

}