**Лабораторная работа 6**

**ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОТНОШЕНИЯ**

**МНОЖЕСТВЕННОГО НАСЛЕДОВАНИЯ**

**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. Используя предыдущую программу, создайте новый производный класс с применением множественного наследования.

2. Проверьте работоспособность АТД и производных классов на тестовом наборе данных.

**Код программы:**

#include <iostream>

#include <stdlib.h>

#include "stdio.h"

#include <iostream>

using namespace std;

// Базовый класс

template <class T1, class T2>

class Agr {

public:

Agr<T1, T2>();

Agr<T1, T2>(const char\* name, T1 n, T2 t);

void input();

void print();

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

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

~Agr<T1, T2>();

Agr <T1, T2>operator+(Agr<T1, T2>& a);

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

protected:

char\* tractor; T1 mass; T2 cost;

};

template <class T1, class T2>

Agr<T1, T2>::Agr()

{

tractor = new char[30];

strcpy(tractor, "Tractor");

mass = 200;

cost = 600;

}

template <class T1, class T2>

Agr<T1, T2>::Agr<T1, T2>(const char\* name, T1 n, T2 t) {

tractor = NULL;

tractor = new char[strlen(name) + 1];

strcpy(tractor, name);

cost = t;

mass = n;

};

template <class T1, class T2>

void Agr<T1, T2>::print() {

cout << endl;

cout << "Трактор: " << tractor << endl;

cout << "Масса: " << mass << " кг" << endl;

cout << "Стоимость: " << cost << " руб" << endl;

};

template <class T1, class T2>

void Agr<T1, T2>::input()

{

cout << "Введите название трактора: ";

cin >> tractor;

cout << "Введите массу трактора (кг): ";

cin >> mass;

cout << "Введите стоимость трактора (руб): ";

cin >> cost;

}

template <class T1, class T2>

Agr<T1, T2>::Agr<T1, T2>(const Agr& a) {

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

strcpy(tractor, a.tractor);

cost = a.cost;

mass = a.mass;

};

template <class T1, class T2>

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

return ((!strcmp(tractor, a.tractor)) &&

cost == a.cost &&

mass == a.mass);

};

template <class T1, class T2>

Agr<T1, T2>::~Agr<T1, T2>() {

delete[]tractor;

};

template <class T1, class T2>

Agr<T1, T2> Agr<T1, T2>::operator+(Agr& a) {

Agr v;

v.tractor = new char[strlen(a.tractor) + strlen(tractor) + 2];

strcpy(v.tractor, tractor);

strcat(v.tractor, ",");

strcat(v.tractor, a.tractor);

v.cost = cost + a.cost;

v.mass = mass + a.mass;

return v;

};

template <class T1, class T2>

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

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

strcpy(tractor, a.tractor);

cost = a.cost;

mass = a.mass;

return(\*this);

};

//Второй базовый класс

template <class T1 , class T2>

class ClassDvig {

public:

ClassDvig<T1, T2>(const char\* dvig);

ClassDvig<T1, T2>();

void inputdvig();

void printdvig();

protected:

char\* dvigP;

};

template <class T1, class T2>

ClassDvig<T1, T2>::ClassDvig<T1, T2>() {

dvigP = new char[30];

strcpy(dvigP, "Колесный");

};

template <class T1, class T2>

ClassDvig<T1, T2>::ClassDvig<T1, T2>(const char\* dvig) {

dvigP = dvig;

};

template <class T1, class T2>

void ClassDvig<T1, T2>::inputdvig()

{

cout << "Введите тип движителя: ";

cin >> dvigP;

}

template <class T1, class T2>

void ClassDvig<T1, T2>::printdvig() {

cout << "Тип движителя: " << dvigP << endl;

};

// Производный класс

template <class T1, class T2>

class Engine : public Agr<T1, T2>, public ClassDvig<T1, T2> {

public:

Engine();

Engine(const char\* name, T1 n, T2 t, const int pow);

~Engine();

void input();

void print();

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

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

protected:

double pow;

};

template <class T1, class T2>

Engine<T1, T2>::Engine() : Agr<T1,T2>(), ClassDvig<T1,T2>()

{

pow = 240;

}

template <class T1, class T2>

Engine<T1, T2>::Engine(const char\* name, T1 n, T2 t, const int pow) : Agr<T1, T2>(name,n,t), ClassDvig<T1, T2>()

{

this->pow = pow;

}

template <class T1, class T2>

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

{

Agr<T1, T2>::~Agr();

}

template <class T1, class T2>

void Engine<T1, T2>::input()

{

Agr<T1, T2>::input();

ClassDvig<T1, T2>::inputdvig();

cout << "Введите мощность двигателя (л.с.): ";

cin >> pow;

}

template <class T1, class T2>

void Engine<T1, T2>::print()

{

Agr<T1, T2>::print();

cout << endl;

ClassDvig<T1, T2>::printdvig();

cout << "Мощность двигателя: " << pow << " л.с." << endl;

}

template <class T1, class T2>

Engine<T1, T2>::Engine<T1, T2>(const Engine& a) {

pow = a.pow;

this->tractor = new char[strlen(a.tractor) + 1];

strcpy(this->tractor, a.tractor);

this->cost = a.cost;

this->mass = a.mass;

};

template <class T1, class T2>

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

pow = a.pow;

this->tractor = new char[strlen(a.tractor) + 1];

strcpy(this->tractor, a.tractor);

this->cost = a.cost;

this->mass = a.mass;

return(\*this);

};

int main()

{

setlocale(LC\_ALL, "Russian");

Agr<int, double> Tractor1;

Engine<int, double> Engine1;

//базовый класс 1

cout << "Введите информацию о тракторе:" << endl;

Tractor1.input();

//базовый класс 2

cout << "Введите информацию о движителе:" << endl;

ClassDvig<int, double> Dvig1;

Dvig1.inputdvig();

cout << endl << "Базовые классы " << endl;

cout << endl << "Базовый класс 1 : " << endl;

Tractor1.print();

cout << endl;

cout << "Базовый класс 2 : " << endl;

Dvig1.printdvig();

cout << endl;

//производный класс с множественным наследованием

cout << endl << "Производный класс " << endl;

cout << "Введите информацию о типе движителя о тракторе и двигателе:" << endl;

Engine1.input();

cout << endl;

cout << "Производный класс: ";

Engine1.print();

cout << endl;

//Без параметров

cout << "Без параметров: ";

Engine<int, double>Engine2;

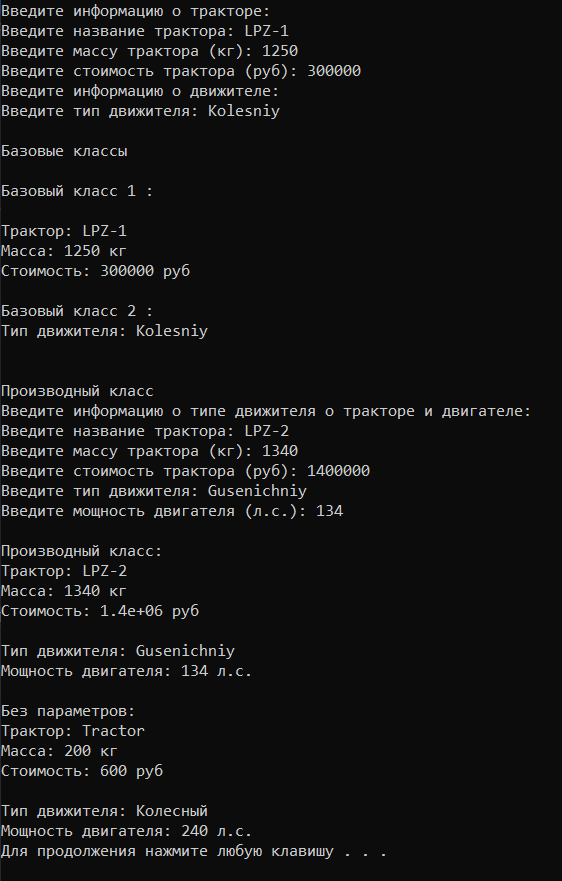
Engine2.print();

system("pause");

return 0;

}

**Результаты работы**

****

**Контрольные** **вопросы**

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

Множественное наследование делает возможным получение производного класса от нескольких базовых классов и позволяет классу перенимать их функциональность.

2.Опишите синтаксис заголовка производного класса при множественном наследовании.

Синтаксис заголовка класса расширяется, чтобы можно было использовать список базовых классов с атрибутами доступа.

class производный: public базовый\_1, private базовый\_2 {…

3.Дайте определение ориентированного ациклического графа.

Это граф, узлы которого являются классами, а ориентированные ребра направлены от производных классов к базовым

4.Где инициируются виртуальные базовые классы?

Виртуальные базовые классы инициализируются перед любыми невиртуальными базовыми классами.

5.Что будет, если из объявлений классов Cow и Buffalo убрать ключевое слово virtual?

Ключевое слово virtual в классе Cow и классе Buffalo предотвращает многократное копирование полей данных weight, price, color из предков класса Beefalo. Так же нельзя будет использовать виртуальные функции.