**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 <string.h>

#include <iostream>

using namespace std;

template <class T1 = int, class T2 = float>

class MemoryDevice {

public:

MemoryDevice(const char\* programa, T1 v, T2 p);

MemoryDevice<T1, T2>();

void print();

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

bool operator==(const MemoryDevice& a);

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

MemoryDevice<T1, T2>& operator++();

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

~MemoryDevice<T1, T2>();

protected:

char\* programm; T1 cost; T2 size;

};

template <class T1, class T2>

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

programm = new char[strlen("EVA 255") + 1];

strcpy(programm, "EVA 255");

cost = 200;

size = 300;

};

template <class T1, class T2>

MemoryDevice<T1, T2>::MemoryDevice<T1, T2>(const char\* programa, T1 v, T2 p) {

programm = new char[strlen(programa) + 1];

strcpy(programm, programa);

cost = v;

size = p;

};

template <class T1, class T2>

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

cout << "Название: " << programm << endl;

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

cout << "Размер: " << size << endl;

};

template <class T1, class T2>

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

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

strcpy(programm, a.programm);

cost = a.cost;

size = a.size;

};

template <class T1, class T2>

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

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

cost == a.cost &&

size == a.size);

}

template <class T1, class T2>

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

delete[] programm;

};

template <class T1, class T2>

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

MemoryDevice <T1, T2> v;

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

strcpy(v.programm, programm);

strcat(v.programm, ",");

strcat(v.programm, a.programm);

v.cost = cost + a.cost;

v.size = size + a.size;

return v;

};

template <class T1, class T2>

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

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

strcpy(programm, a.programm);

cost = a.cost;

size = a.size;

return(\*this);

};

template <class T1, class T2>

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

cost += 1;

size += 1;

return(\*this);

};

//Второй базовый класс

template <class T1 = int, class T2 = float>

class ElectronicMedia {

public:

ElectronicMedia<T1, T2>(int type);

ElectronicMedia<T1, T2>();

void printEM();

protected:

int typeEM;

};

template <class T1, class T2>

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

typeEM = 1;

};

template <class T1, class T2>

ElectronicMedia<T1, T2>::ElectronicMedia<T1, T2>(int type) {

typeEM = type;

};

template <class T1, class T2>

void ElectronicMedia<T1, T2>::printEM() {

if (typeEM == 1)cout << "Класс запоминающего устройства-Оптическое" << endl;

if (typeEM == 2)cout << "Класс запоминающего устройства-Полупроводниковые" << endl;

if (typeEM == 3)cout << "Класс запоминающего устройства-Магнитные" << endl;

};

//производный класс с множественым наследованием

template <class T1 = int, class T2 = float>

class View : public MemoryDevice<T1, T2>, public ElectronicMedia<T1, T2>

{

public:

View<T1, T2>(const char\* programa, T1 v, T2 p, int type, char\* vie);

View<T1, T2>();

void printView();

~View();

protected:

char\* spec;

};

template <class T1, class T2>

View<T1, T2>::View<T1, T2>() : MemoryDevice<T1, T2>(), ElectronicMedia<T1, T2>() {

spec = new char[strlen("EVA 255") + 1];

strcpy(spec, "EVA 255");

};

template <class T1, class T2>

View<T1, T2>::View<T1, T2>(const char\* programa, T1 v, T2 p, int type, char\* vie) : MemoryDevice<T1, T2>(programa, v, p), ElectronicMedia<T1, T2>(type) {

spec = new char[strlen(vie) + 1];

strcpy(spec, vie);

};

template <class T1, class T2>

void View<T1, T2>::printView() {

cout << "Вывод " << endl;

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

ElectronicMedia<T1, T2>::printEM();

cout << "Данное запоминающее устройство принадлежит - " << spec << endl;

};

template <class T1, class T2>

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

delete[] spec;

};

int main()

{

setlocale(LC\_ALL, "Russian");

char\* programa = new char[10];

char\* spec = new char[10];

double cost, size;

int type;

cout << "Введите название запоминающего устройства ";

cin >> programa;

cout << endl << "Введите стоимость запоминающего устройства ";

cin >> cost;

cout << endl << "Введите размер запоминающего устройства ";

cin >> size;

cout << endl << "Введите классификацию запоминающего устройства ";

cin >> type;

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

cout << " 1 " << endl;

MemoryDevice<float, int>ml(programa, cost, size);

ml.print();

cout << " 2 " << endl;

ElectronicMedia<float, int>mg(type);

mg.printEM();

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

cout << endl << "Введите вид запоминающего устройства: " << endl;

cin >> spec;

cout << endl;

View<float, int>b(programa, cost, size, type, spec);

b.print();

b.printEM();

b.printView();

cout << endl << "Параметры отсутствуют" << endl;

View<float, int>c;

c.print();

c.printEM();

c.printView();

return 0;

}

**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.

**Результат работы программы:**

