**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**:

#include "pch.h"

#include <iostream>

#include <clocale>

#include <iostream>

using namespace std;

class Monitors {

private:

bool colored; // Цветные, монохромные.

char \* inform; // Алфавитно-цифровые, графические.

protected:

int Height; // в пикселях

int Width; // в пикселях

int type; /\*

1 - Электроннолучевой монитор;

2 - Жидкокристаллический монитор;

3 - Плазменный монитор;

4 - Проектор;

5 - LED-монитор;

6 - OLED-монитор;

7 - Виртуальный ретинальный монитор;

8 - Лазерный монитор.

\*/

double diagonal; // диагональ

public:

Monitors();

Monitors(int h, int w, int t, bool c, const char\* i, double d);

Monitors(int h, int w);

Monitors(const Monitors &m);

~Monitors();

void input(int h, int w, int t, bool c, const char \* i, double d);

void print();

Monitors copy(const Monitors& m);

Monitors sum(const Monitors& b);

bool comp(const Monitors& b);

Monitors & operator + (Monitors a);

Monitors & operator - (Monitors a);

bool operator > (Monitors a);

bool operator < (Monitors a);

Monitors & operator = (const Monitors& a);

};

Monitors::Monitors() {

Height = 1280;

Width = 720;

type = 8;

colored = false;

const char \* sos = "графический";

inform = new char[strlen(sos) + 1];

strcpy(inform, sos);

diagonal = 16.1;

}

Monitors::Monitors(int h, int w, int t, bool c, const char\* i, double d) {

Height = h;

Width = w;

type = t;

colored = c;

inform = new char[strlen(i) + 1];

strcpy(inform, i);

diagonal = d;

};

Monitors::Monitors(int h, int w) {

Height = h;

Width = w;

type = 5;

colored = false;

const char \* sos = "алфавитно-цифровой";

inform = new char[strlen(sos) + 1];

strcpy(inform, sos);

diagonal = 27.8;

};

Monitors::Monitors(const Monitors &m) {

Height = m.Height;

Width = m.Width;

type = m.type;

colored = m.colored;

inform = new char[strlen(m.inform) + 1];

strcpy(inform, m.inform);

diagonal = m.diagonal;

};

Monitors::~Monitors() {

delete[] inform;

};

void Monitors::input(int h, int w, int t, bool c, const char \* i, double d) {

Height = h;

Width = w;

type = t;

colored = c;

inform = new char[strlen(i) + 1];

strcpy(inform, i);

diagonal = d;

};

void Monitors::print() {

cout << Height << endl;

cout << Width << endl;

cout << type << endl;

colored ? cout << "true" : cout << "false";

cout << endl;

cout << inform << endl;

cout << diagonal << endl;

};

Monitors Monitors::copy(const Monitors& m) {

Monitors h;

if (&m != this) {

Height = m.Height;

Width = m.Width;

type = m.type;

colored = m.colored;

inform = new char[strlen(m.inform) + 1];

strcpy(inform, m.inform);

diagonal = m.diagonal;

}

return (h);

}

Monitors Monitors::sum(const Monitors& b) {

Monitors h;

h.Height = Height + b.Height;

h.Width = Width + b.Width;

h.type = b.type;

h.colored = b.colored;

h.inform = new char[strlen(b.inform) + 1];

strcpy(h.inform, b.inform);

h.diagonal = b.diagonal;

return (h);

}

bool Monitors::comp(const Monitors& b) {

return ((Height \* Width) >= (b.Height \* b.Width));

}

Monitors & Monitors::operator + (Monitors a) {

Height += a.Height;

Width += a.Height;

return(\*this);

}

Monitors & Monitors::operator - (Monitors a) {

Height -= a.Height;

Width -= a.Height;

return(\*this);

}

bool Monitors::operator > (Monitors a) {

return ((Height \* Width) > (a.Height \* a.Width));

}

bool Monitors::operator < (Monitors a) {

return ((Height \* Width) < (a.Height \* a.Width));

}

Monitors & Monitors::operator = (const Monitors& a) {

Height = a.Height;

Width = a.Height;

type = a.type;

colored = a.colored;

delete[] inform;

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

strcpy(inform, a.inform);

diagonal = a.diagonal;

return(\*this);

}

// Class PC

class PC {

private:

char\* color;

int Length; // mm

protected:

int HDD; // Gb

int RAM; // Gb

char\* CPU;

char\* Card;

public:

PC();

PC(const char\* c, int l, int H, int R, const char\* C, const char\* card);

PC(const PC &m);

~PC();

void input(char\* c, int l, int H, int R, const char\* C, const char\* card);

void print();

PC& operator = (const PC& a);

bool bolshe(const PC& b);

};

PC::PC() {

color = new char[strlen("black") + 1];

strcpy(color, "black");

Length = 380;

HDD = 1000;

RAM = 8;

CPU = new char[strlen("Core i5 9400F") + 1];

strcpy(CPU, "Core i5 9400F");

Card = new char[strlen("GeForce GTX 1660") + 1];

strcpy(Card, "GeForce GTX 1660");

}

PC::PC(const char\* c, int l, int H, int R, const char\* C, const char\* card) {

color = new char[strlen(c) + 1];

strcpy(color, c);

Length = l;

HDD = H;

RAM = R;

CPU = new char[strlen(C) + 1];

strcpy(CPU, C);

Card = new char[strlen(card) + 1];

strcpy(Card, card);

}

PC::PC(const PC &m) {

color = new char[strlen(m.color) + 1];

strcpy(color, m.color);

Length = m.Length;

HDD = m.HDD;

RAM = m.RAM;

CPU = new char[strlen(m.CPU) + 1];

strcpy(CPU, m.CPU);

Card = new char[strlen(m.Card) + 1];

strcpy(Card, m.Card);

}

PC::~PC() {

delete[] color;

delete[] CPU;

delete[] Card;

}

void PC::input(char\* c, int l, int H, int R, const char\* C, const char\* card) {

color = new char[strlen(c) + 1];

strcpy(color, c);

Length = l;

HDD = H;

RAM = R;

CPU = new char[strlen(C) + 1];

strcpy(CPU, C);

Card = new char[strlen(card) + 1];

strcpy(Card, card);

}

void PC::print() {

cout << color << endl;

cout << Length << endl;

cout << HDD << endl;

cout << RAM << endl;

cout << CPU << endl;

cout << Card << endl;

}

PC& PC::operator = (const PC& a){

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

strcpy(color, a.color);

Length = a.Length;

HDD = a.HDD;

RAM = a.RAM;

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

strcpy(Card, a.Card);

return(\*this);

}

bool PC::bolshe(const PC& b){

return HDD >= b.HDD;

}

// class Monoblocks

class Monoblocks : public Monitors, public PC {

private:

double Weight; // kg

public:

Monoblocks();

Monoblocks(double weight, int h, int w, double d, int Hd, int r, const char \* s, const char\* card);

Monoblocks(const Monoblocks &m);

~Monoblocks();

void input(double weight, int h, int w, double d, int Hd, int r, const char \* s, const char\* card);

void print();

Monoblocks& operator = (const Monoblocks& a);

};

Monoblocks::Monoblocks() :Monitors(), PC() {

Weight = 6.17;

HDD = 500;

RAM = 4;

CPU = new char[strlen("Celeron N3350") + 1];

strcpy(CPU, "Celeron N3350");

Height = 1080;

Width = 1920;

diagonal = 21.5;

Card = new char[strlen("GeForce GTX 980") + 1];

strcpy(Card, "GeForce GTX 980");

}

Monoblocks::Monoblocks(double weight, int h, int w, double d, int Hd, int r, const char \* s, const char \* card):Monitors(h,w,1,true,"алфавитный",d),PC("red",300,Hd,r,s,card) {

Weight = weight;

HDD = Hd;

RAM = r;

CPU = new char[strlen(s) + 1];

strcpy(CPU, s);

Height = h;

Width = w;

diagonal = d;

Card = new char[strlen(card) + 1];

strcpy(Card, card);

}

Monoblocks::Monoblocks(const Monoblocks &m) :Monitors(m), PC(m) {

Weight = m.Weight;

Height = m.Height;

Width = m.Width;

HDD = m.HDD;

RAM = m.RAM;

CPU = new char[strlen(m.CPU) + 1];

strcpy(CPU, m.CPU);

diagonal = m.diagonal;

Card = new char[strlen(m.Card) + 1];

strcpy(Card, m.Card);

}

Monoblocks::~Monoblocks() {

}

void Monoblocks::input(double weight, int h, int w, double d, int Hd, int r, const char \* s, const char\* card) {

Weight = weight;

HDD = Hd;

RAM = r;

CPU = new char[strlen(s) + 1];

strcpy(CPU, s);

Height = h;

Width = w;

diagonal = d;

Card = new char[strlen(card) + 1];

strcpy(Card, card);

}

void Monoblocks::print() {

cout << Weight << endl;

cout << HDD << endl;

cout << RAM << endl;

cout << CPU << endl;

cout << Height << endl;

cout << Width << endl;

cout << diagonal << endl;

cout << Card << endl;

}

Monoblocks& Monoblocks::operator = (const Monoblocks& a) {

Monitors::operator=(a);

PC::operator=(a);

Weight = a.Weight;

HDD = a.HDD;

RAM = a.RAM;

Height = a.Height;

Width = a.Width;

delete[] CPU;

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

strcpy(CPU, a.CPU);

diagonal = a.diagonal;

delete[] Card;

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

strcpy(Card, a.Card);

return(\*this);

}

int main() {

setlocale(LC\_ALL, "Russian");

Monitors monitor1;

PC pc1, pc2("black", 450, 444, 16, "Intel Core i9", "GeForce GT 960");

Monoblocks MB1, MB2(3.70,1920,860,25.15,800,16,"Celeron N230","GeForce GTX 1660"), MB3(MB1);

cout << "Monitors:\n";

monitor1.print();

cout << endl << "PC:\n";

pc1.print();

cout << endl << "Monoblocks:\n";;

MB1.print();

cout << endl;

MB2.print();

cout << endl;

MB3.print();

cout << endl << "copy and summa\n";

MB1.copy(MB2.sum(MB2));

MB3.print();

cout << endl;

MB1.print();

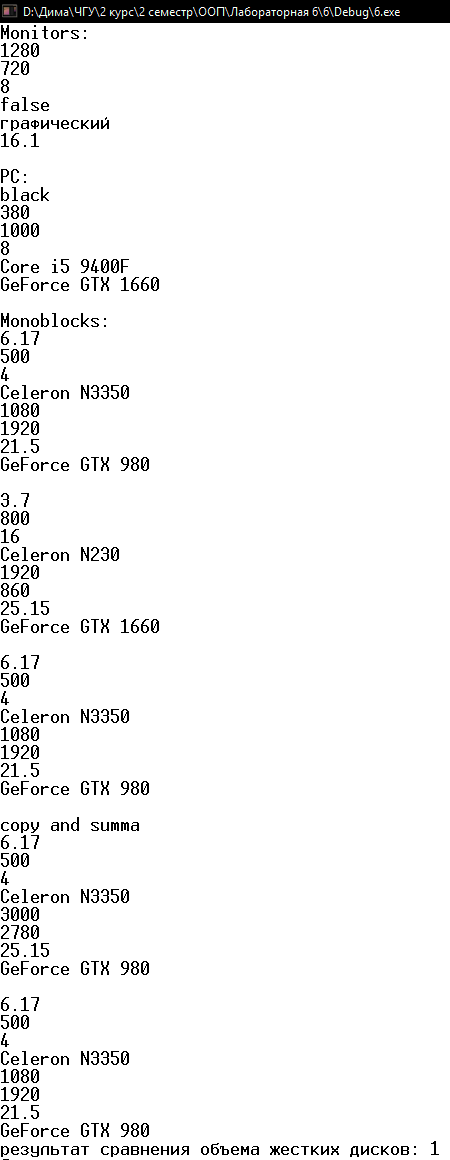
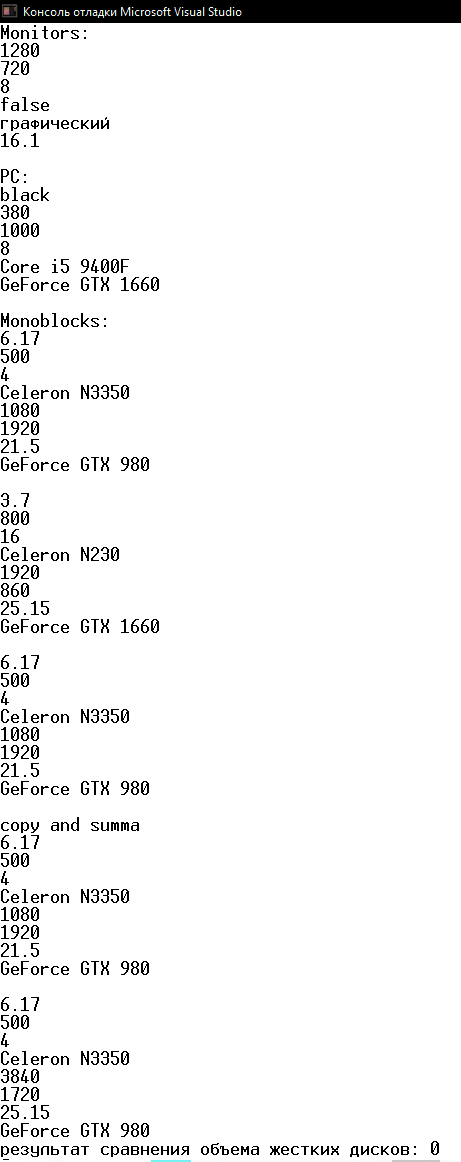
cout << "результат сравнения объема жестких дисков: " << MB1.bolshe(MB2) << endl;

system("pause");

return 0;

}

**Тестирование**:

**Контрольные вопросы:**

1. С какой целью и в каких случаях используется множественное наследование?
2. Опишите синтаксис заголовка производного класса при множественном наследовании.
3. Дайте определение ориентированного ациклического графа.
4. Где инициируются виртуальные базовые классы?
5. Что будет, если из объявлений классов Cow и Buffalo убрать ключевое слово virtual (см. последний пример в теоретических положениях)?

**Ответы:**

1. Множественное наследование делает возможным получение производного класса от нескольких базовых классов.
2. class third: public first, public second {}
3. Ациклический ориентированный граф (DAG) – это граф, узлы которого являются классами, а ориентированные ребра направлены от производных классов к базовым.
4. Виртуальные базовые классы инициализируются перед любыми невиртуальными базовыми классами и в том порядке, в котором они появляются в DAG наследования при просмотре его снизу-вверх и слева направо.
5. Ключевое слово virtual в классе Cow и классе Buffalo предотвращает многократное копирование полей данных weight, price, color из предков класса Beefalo. Следовательно, его отсутствие приведет к многократному копированию.