**Лабораторная работа 9**

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 "stdafx.h"

#include <clocale>

#include <time.h>

#include <iostream>

using namespace std;

template <class T, class M>

class Monitors {

private:

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.

\*/

bool colored; // Цветные, монохромные.

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

protected:

T Height; // в пикселях

T Width; // в пикселях

M diagonal; // в дюймах

public:

Monitors() {

Height = 1280;

Width = 720;

type = 8;

colored = false;

char \* sos = "графический";

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

strcpy(inform, sos);

diagonal = 22;

};

Monitors(T h, T w, int t, bool c, char\* i, M d) {

Height = h;

Width = w;

type = t;

colored = c;

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

strcpy(inform, i);

diagonal = d;

};

Monitors(T h, T w) {

Height = h;

Width = w;

type = 5;

colored = false;

char \* sos = "алфавитно-цифровой";

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

strcpy(inform, sos);

diagonal = 22;

};

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() {

delete[] inform;

};

void input(T h, T w, int t, bool c, char \* i, M d) {

Height = h;

Width = w;

type = t;

colored = c;

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

strcpy(inform, i);

diagonal = d;

};

void print() {

cout << Height << endl;

cout << Width << endl;

cout << type << endl;

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

cout << endl;

cout << inform << endl;

cout << diagonal << endl;

};

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 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 comp(const Monitors& b) {

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

}

Monitors & operator + (Monitors a) {

Height += a.Height;

Width += a.Height;

return(\*this);

}

Monitors & operator - (Monitors a) {

Height -= a.Height;

Width -= a.Height;

return(\*this);

}

bool operator == (Monitors a) {

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

}

bool operator > (Monitors a) {

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

}

bool operator < (Monitors a) {

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

}

Monitors & operator = (Monitors a) {

Height = a.Height;

Width = a.Width;

type = a.type;

colored = a.colored;

delete[] inform;

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

strcpy(inform, a.inform);

diagonal = a.diagonal;

return(\*this);

}

};

template <class O>

class PC {

private:

char\* color;

int Length; // mm

protected:

O HDD; // Gb

O RAM; // Gb

char\* CPU;

char\* Card;

public:

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("Core i5 9400F") + 1];

strcpy(Card, "GeForce GTX 1660");

}

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

}

void input(char\* c, int l, O H, O R, char C, 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 print() {

cout << color << endl;

cout << Length << endl;

cout << HDD << endl;

cout << RAM << endl;

cout << CPU << endl;

cout << Card << endl;

}

};

template <class T, class M, class O>

class Monoblocks: public Monitors< T, M>, public PC <O> {

private:

double Weight; // kg

public:

Monoblocks() {

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;

}

Monoblocks(double W, O h, O r, char\* c, T H, T w, M d) {

Weight = W;

HDD = h;

RAM = r;

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

strcpy(CPU, c);

Height = H;

Width = w;

diagonal = d;

}

Monoblocks(const Monoblocks &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;

}

void input(double W, O h, O r, char\* c, T H, T w, M d) {

Weight = W;

HDD = h;

RAM = r;

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

strcpy(CPU, c);

Height = H;

Width = w;

diagonal = d;

};

void print() {

cout << Weight << endl;

cout << HDD << endl;

cout << RAM << endl;

cout << CPU << endl;

cout << Height << endl;

cout << Width << endl;

cout << diagonal << endl;

};

};

template <class N>

void copy(N a[], N b[], int n) {

for (int i = 0; i < n; i++) {

a[i] = b[i];

}

}

int main() {

srand(time(0));

setlocale(LC\_ALL, "Russian");

Monitors <double, char> monitor3(1920.58, 1080.25, 3, true, "графический", 134);

PC <int> PC1, PC2("gold", 250, 800, 16, "Ryzen 5 2500X", "Radeon RX 5500XT");

Monoblocks <int, int, float> MB1, MB2(5.5, 1000, 8, "A4 9125", 1600, 1200, 27.3), MB3(MB2);

PC1.print();

cout << endl;

PC2.print();

cout << endl;

cout << endl;

MB1.print();

cout << endl;

MB2.print();

cout << endl;

MB3.print();

cout << endl;

MB1.input(5, 128, 2,"Pentium J5005", 1280, 720, 22.5);

MB1.print();

system("pause");

return 0;

}

**Тестирование**:

 