***Лабораторная работа***

## ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОТНОШЕНИЯ

**ПРОСТОГО НАСЛЕДОВАНИЯ**

ЦЕЛЬ РАБОТЫ:

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

**Задания:**

1. Создайте производный класс для АТД, реализованного по заданию лабораторной работы 4, используя одиночное наследование.

2. Проверьте работоспособность АТД и производного класса на тестовом наборе данных.

Код:

#include "stdafx.h"

#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, char\* i, double d);

Monitors(int h, int w) ;

Monitors(const Monitors &m) ;

~Monitors() ;

void input(int h, int w, int t, bool c, 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 = (Monitors a);

};

Monitors::Monitors() {

Height = 1280;

Width = 720;

type = 8;

colored = false;

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, 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;

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, 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 = (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 Monoblocks

class Monoblocks: public Monitors {

private:

int HDD; // Gb

int RAM; // Gb

char\* CPU;

public:

Monoblocks();

Monoblocks(int h, int w, int t, bool c, char\* i, double d, int Hd, int r, char\* s);

Monoblocks(const Monoblocks &m);

~Monoblocks();

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

void print();

Monoblocks& operator = (Monoblocks a);

};

Monoblocks::Monoblocks():Monitors(){

HDD = 500;

RAM = 8;

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

strcpy(CPU, "Celeron N3350");

Height = 1900;

Width = 980;

diagonal = 17.18;

}

Monoblocks::Monoblocks(int h, int w, int t, bool c, char\* i, double d, int Hd, int r, char\* s):Monitors(h, w, t, c, i, d) {

HDD = Hd;

RAM = r;

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

strcpy(CPU, s);

Height = h;

Width = w;

diagonal = d;

}

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

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;

}

Monoblocks::~Monoblocks(){

delete[] CPU;

//Monitors::~Monitors(); // пришел к выводу, что при завершении работы деструктора производного класса вызывается деструктор базового класса. А при его явном вызове деструктор повторится дважды и произойдет ошибка.

}

void Monoblocks::input(int h, int w, int t, bool c, char\* i, double d, int Hd, int r, char\* s){

HDD = Hd;

RAM = r;

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

strcpy(CPU, s);

Height = h;

Width = w;

diagonal = d;

}

void Monoblocks::print(){

cout << HDD << endl;

cout << RAM << endl;

cout << CPU << endl;

cout << Height << endl;

cout << Width << endl;

cout << diagonal << endl;

}

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

Monitors::operator=(a);

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;

return(\*this);

}

int main() {

setlocale(LC\_ALL, "Russian");

Monitors monitor1, monitor2(1600, 1200, 3, true, "графический",22.13);

Monoblocks MB1, MB2(1920, 780, 8, true, "алфавитный", 33.8, 400, 8, "Celeron N230"), MB3(MB1);

cout << "Monitors:\n";

monitor1.print();

cout << endl;

monitor2.print();

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

MB1.print();

cout << endl;

MB2.print();

cout << endl;

MB3.print();

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

MB3.copy(MB1.sum(MB2));

MB3.print();

cout << endl << "operator =\n";

MB2.input(300, 200, 1, false, "графический", 11.5, 256, 2, "Intel duo");

MB1 = MB2;

cout << endl;

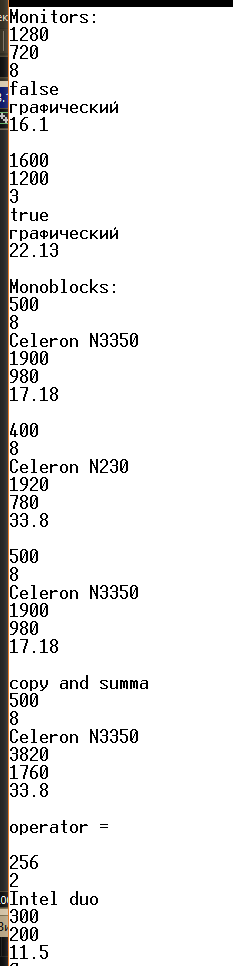
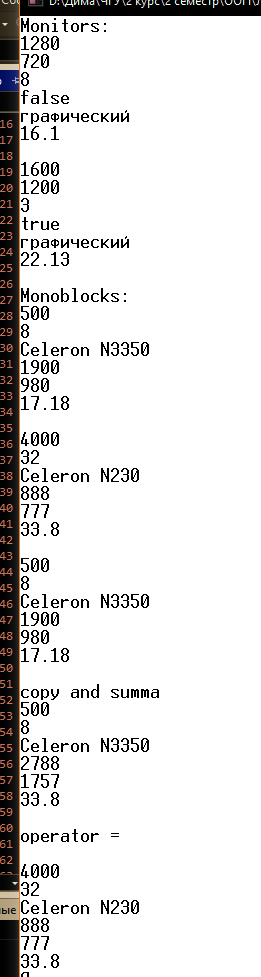
MB1.print();

system("pause");

return 0;

}

**Тестирование:**

**Контрольные вопросы:**

1. Дайте определение наследования.
2. Какие модификаторы прав доступа к производному классу вы знаете и в чем их особенности?
3. Как выполняется конструктор при наследовании?
4. Дайте определение чисто виртуальной функции.
5. Какие операторы используются для разыменования указателя на член класса?

**Ответы:**

1. Наследование – это механизм получения нового класса на основе существующего класса.
2. Если наследование открытое – «public», то открытые элементы базового класса наследуются, закрытые не наследуются, а защищенные наследуются как закрытые. Если наследование закрытое – «private», то никакие элементы базового класса не доступны объектам производного класса. Если наследование защищенное – «protected», то разрешен доступ методов производного класса к общим элементам базового класса, но запрещен доступ для всех других методов.
3. При наследовании и инициализации членов класса конструкторы выполняются в следующем порядке:
4. Базовые классы инициализируются в порядке объявления;
5. Члены инициализируются в порядке объявления.
6. Чисто виртуальная функция – это виртуальная функция-член, тело которой не определено. Она объявляется внутри класса:

*virtual прототип\_функции = 0;*

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