***Попов.Е.С***

***Лабораторная работа* *8***

## 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 7, используя одиночное наследование.

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

Код:

#include "pch.h"

#include <iostream>

#include <string>

using namespace std;

void PrintLine() {

cout << "-================================-\n";

}

//class Sensorniy : public Myphone<T>

//{

//public:

//

//private:

//

//};

template <class T>

class Myphone

{

private:

char\* mark;

T releaseY;

T price;

char\* pr;

public:

Myphone();

Myphone(char\* a, T b, T c, char\* d);

Myphone(const Myphone &a);

~Myphone();

Myphone Sum(const Myphone &a);

/\*void copy(const Myphone &a);\*/

bool Compair(Myphone &a);

void destroy();

void SetMark(char\* a);

void GetMark();

void SetReleaseY(T a);

int GetReleaseY();

void SetPrice(T a);

void GetPrice();

void Setpr(char\* a);

void Getpr();

Myphone& operator+(const Myphone &a);

bool operator>(const Myphone &r);

Myphone& operator= (const Myphone &r);

bool operator==(const Myphone &a);

};

template <class T> Myphone<T>::Myphone()

{

mark = new char[11];

pr = new char[11];

strcpy(mark, "none");

strcpy(pr, "none");

price = 0;

releaseY = 0;

}

template <class T> Myphone<T>::Myphone(char \* a, T b, T c, char \* d)

{

mark = a;

releaseY = b;

price = c;

pr = d;

}

template <class T> Myphone<T>::Myphone(const Myphone<T> & a)

{// констр копирования

// тело конструктора

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

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

strcpy(mark, a.mark);

strcpy(pr, a.pr);

price = a.price;

releaseY = a.releaseY;

}

template <class T>Myphone<T>::~Myphone()

{

//delete[] mark;

//delete[] pr;

cout << "Memory has been succesfully cleaned" << endl;

};

template <class T>void Myphone<T>::GetMark() {

cout.width(5);

cout << "mark: " << mark << "\n";

}

template <class T>void Myphone<T>::SetReleaseY(T a) {

releaseY = a;

}

template <class T>int Myphone<T>::GetReleaseY() {

cout.width(5);

cout << "released: " << releaseY << "\n";

return releaseY;

}

template <class T>void Myphone<T>::SetMark(char\* a) {

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

strcpy(mark, a);

}

template <class T>void Myphone<T>::SetPrice(T a) {

price = a;

}

template <class T>void Myphone<T>::GetPrice() {

cout.width(5);

cout << "price: " << price << "\n";

}

template <class T>void Myphone<T>::Setpr(char\* a) {

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

strcpy(pr, a);

}

template <class T>void Myphone<T>::Getpr() {

cout.width(5);

cout << "processor: " << pr << "\n";

}

template <class T>Myphone<T> & Myphone<T>::operator+(const Myphone<T> & a)

{

Myphone v;

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

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

strcpy(v.mark, mark);

strcat(v.mark, ",");

strcat(v.mark, a.mark);

strcpy(v.pr, pr);

strcat(v.pr, ",");

strcat(v.pr, a.pr);

v.price = price + a.price;

v.releaseY = releaseY + a.releaseY;

v.GetMark();

v.Getpr();

v.GetPrice();

v.GetReleaseY();

return v;

}

template <class T>bool Myphone<T>::operator>(const Myphone<T> & a)

{

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

price > a.price &&

releaseY > a.releaseY);

}

template <class T>Myphone<T> & Myphone<T>::operator=(const Myphone<T> & r)

{

price = r.price;

releaseY = r.releaseY;

pr = r.pr;

mark = r.mark;

return \*this;

}

template <class T>bool Myphone<T>::operator==(const Myphone<T> & a)

{

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

price == a.price &&

releaseY == a.releaseY);

}

template <class T>void Myphone<T>::destroy() {

delete[] mark;

delete[] pr;

cout << "Memory has been succesfully cleaned" << endl;

};

template <class T>Myphone<T> Myphone<T>::Sum(const Myphone<T> &a)

{

Myphone v;

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

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

strcpy(v.mark, mark);

strcat(v.mark, ",");

strcat(v.mark, a.mark);

strcpy(v.pr, pr);

strcat(v.pr, ",");

strcat(v.pr, a.pr);

v.price = price + a.price;

v.releaseY = releaseY + a.releaseY;

v.GetMark();

v.Getpr();

v.GetPrice();

v.GetReleaseY();

return v;

}

template <class T>bool Myphone<T>::Compair(Myphone<T> &a)

{

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

price == a.price &&

releaseY == a.releaseY);

}

template <class T, class T2>

class Myphone1

{

private:

T2\* mark;

T releaseY;

T price;

T2\* pr;

public:

Myphone1();

Myphone1(T2\* a, T b, T c, T2\* d);

Myphone1(const Myphone1 &a);

~Myphone1();

Myphone1 Sum(const Myphone1 &a);

/\*void copy(const Myphone &a);\*/

bool Compair(Myphone1 &a);

void destroy();

void SetMark(T2\* a);

void GetMark();

void SetReleaseY(T a);

int GetReleaseY();

void SetPrice(T a);

void GetPrice();

void Setpr(T2\* a);

void Getpr();

Myphone1& operator+(const Myphone1 &a);

bool operator>(const Myphone1 &r);

Myphone1& operator= (const Myphone1 &r);

bool operator==(const Myphone1 &a);

};

template <class T, class T2> Myphone1<T, T2>::Myphone1()

{

mark = new char[11];

pr = new char[11];

strcpy(mark, "none");

strcpy(pr, "none");

price = 0;

releaseY = 0;

}

template <class T, class T2> Myphone1<T, T2>::Myphone1<T, T2>(T2 \* a, T b, T c, T2 \* d)

{

mark = a;

releaseY = b;

price = c;

pr = d;

}

template <class T, class T2> Myphone1<T, T2>::Myphone1<T, T2>(const Myphone1<T, T2> & a)

{// констр копирования

// тело конструктора

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

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

strcpy(mark, a.mark);

strcpy(pr, a.pr);

price = a.price;

releaseY = a.releaseY;

}

template <class T, class T2> Myphone1<T, T2>::~Myphone1()

{

//delete[] mark;

//delete[] pr;

cout << "Memory has been succesfully cleaned" << endl;

};

template <class T, class T2 >void Myphone1<T, T2>::GetMark() {

cout.width(5);

cout << "mark: " << mark << "\n";

}

template <class T, class T2 >void Myphone1<T, T2>::SetReleaseY(T a) {

releaseY = a;

}

template <class T, class T2 >int Myphone1<T, T2>::GetReleaseY() {

cout.width(5);

cout << "released: " << releaseY << "\n";

return releaseY;

}

template <class T, class T2 >void Myphone1<T, T2>::SetMark(T2\* a) {

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

strcpy(mark, a);

}

template <class T, class T2 >void Myphone1<T, T2>::SetPrice(T a) {

price = a;

}

template <class T, class T2 >void Myphone1<T, T2>::GetPrice() {

cout.width(5);

cout << "price: " << price << "\n";

}

template <class T, class T2 >void Myphone1<T, T2>::Setpr(T2\* a) {

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

strcpy(pr, a);

}

template <class T, class T2 >void Myphone1<T, T2>::Getpr() {

cout.width(5);

cout << "processor: " << pr << "\n";

}

template <class T, class T2 >Myphone1<T, T2> & Myphone1<T, T2>::operator+(const Myphone1<T, T2> & a)

{

Myphone1 v;

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

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

strcpy(v.mark, mark);

strcat(v.mark, ",");

strcat(v.mark, a.mark);

strcpy(v.pr, pr);

strcat(v.pr, ",");

strcat(v.pr, a.pr);

v.price = price + a.price;

v.releaseY = releaseY + a.releaseY;

v.GetMark();

v.Getpr();

v.GetPrice();

v.GetReleaseY();

return v;

}

template <class T, class T2 >bool Myphone1<T, T2>::operator>(const Myphone1<T, T2> & a)

{

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

price > a.price &&

releaseY > a.releaseY);

}

template <class T, class T2 >Myphone1<T, T2> & Myphone1<T, T2>::operator=(const Myphone1<T, T2> & r)

{

price = r.price;

releaseY = r.releaseY;

pr = r.pr;

mark = r.mark;

return \*this;

}

template <class T, class T2 >bool Myphone1<T, T2>::operator==(const Myphone1<T, T2> & a)

{

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

price == a.price &&

releaseY == a.releaseY);

}

template <class T, class T2 >void Myphone1<T, T2>::destroy() {

delete[] mark;

delete[] pr;

cout << "Memory has been succesfully cleaned" << endl;

};

template <class T, class T2 >Myphone1<T, T2> Myphone1<T, T2>::Sum(const Myphone1<T, T2> &a)

{

Myphone v;

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

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

strcpy(v.mark, mark);

strcat(v.mark, ",");

strcat(v.mark, a.mark);

strcpy(v.pr, pr);

strcat(v.pr, ",");

strcat(v.pr, a.pr);

v.price = price + a.price;

v.releaseY = releaseY + a.releaseY;

v.GetMark();

v.Getpr();

v.GetPrice();

v.GetReleaseY();

return v;

}

template <class T, class T2 >bool Myphone1<T, T2>::Compair(Myphone1<T, T2> &a)

{

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

price == a.price &&

releaseY == a.releaseY);

}

class Sensorniy : public Myphone<char>

{

public:

void SetOS(char\* a);

void GetOS();

void SetNfc(bool i);

void GetNfc();

private:

char\* OS;

bool NFC;

};

void Sensorniy::SetOS(char\* a) {

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

strcpy(OS, a);

}

void Sensorniy::GetOS()

{

cout << "OS: " << OS << "\n";

}

void Sensorniy::GetNfc()

{

if (NFC) {

cout << "include NFC\n";

}

else {

cout << "don't have NFC\n";

}

}

void Sensorniy::SetNfc(bool i)

{

if (i)

{

NFC = 1;

}

else

{

NFC = 0;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

PrintLine();

Sensorniy i6;

cout << "проверка работы методов унаследованных от базового класса\n";

i6.SetMark((char\*)"huawei");

i6.GetMark();

PrintLine();

Sensorniy i7;

i7 = i6;

i7.GetMark();

PrintLine();

cout << "проверка работы методов производного класса\n";

i6.SetOS((char\*)"windows");

i6.GetOS();

i6.SetNfc(1);

i6.GetNfc();

i7.SetNfc(0);

i7.GetNfc();

PrintLine();

Myphone<int> G5;

G5.SetMark((char\*)"SAMSUNG");

G5.GetMark();

G5.SetReleaseY(2018);

G5.GetReleaseY();

G5.SetPrice(15000);

G5.GetPrice();

G5.Setpr((char\*)"AMD g4560");

G5.Getpr();

PrintLine();

cout << "объект созданный с помощью конструктора по умолчанию\n";

Myphone<int> Pixel;

Pixel.GetMark();

Pixel.GetReleaseY();

Pixel.GetPrice();

Pixel.Getpr();

PrintLine();

Myphone<int> M5s\_lite((char\*)"Xiaomi", 2013, 16000, (char\*)"privet");

Myphone<int> M5s = Myphone<int>(M5s\_lite);

cout << "объект созданный с помощью констр копирования" << endl;

M5s.GetMark();

M5s.GetReleaseY();

M5s.GetPrice();

M5s.Getpr();

PrintLine();

M5s\_lite.GetMark();

M5s\_lite.GetReleaseY();

M5s\_lite.GetPrice();

M5s\_lite.Getpr();

PrintLine();

if (Pixel.Compair(M5s)) {

cout << "equal" << endl;

}

else {

cout << "not equal" << endl;

}

PrintLine();

//Pixel.Sum(G5);

//PrintLine();

Myphone<int> a;

a = M5s;

cout << "объект созданный из пергруженного оператора присваивания" << endl;

a.GetMark();

a.GetReleaseY();

a.GetPrice();

a.Getpr();

PrintLine();

M5s.GetMark();

M5s.GetReleaseY();

M5s.GetPrice();

M5s.Getpr();

PrintLine();

if (a == M5s) {

cout << "1й объект равен второму" << endl;

}

else {

cout << "1й объект не равен второму" << endl;

}

PrintLine();

Myphone<int> b;

cout << "объект созданный из пергруженного оператора сложения" << endl;

b = a + G5;

PrintLine();

if (Pixel > M5s) {

cout << "1й объект больше второго" << endl;

}

else {

cout << "1й объект не больше второго" << endl;

}

PrintLine();

cout << "объект типа float\n";

Myphone<float> c;

c.GetMark();

c.SetPrice(12.68956);

c.GetPrice();

c.Getpr();

PrintLine();

Myphone<float> c1;

c1.GetMark();

c1.SetPrice(80.654);

c1.GetPrice();

c1.Getpr();

PrintLine();

Myphone<float> c2;

c2.GetMark();

c2.SetPrice(1200.654);

c2.GetPrice();

c2.Getpr();

PrintLine();

cout << "объект типа double\n";

Myphone<double> y;

y.GetMark();

y.SetPrice(1200.665);

y.GetPrice();

y.Getpr();

PrintLine();

Myphone<double> y1;

y.GetMark();

y.SetPrice(55.661115);

y.GetPrice();

y.Getpr();

PrintLine();

Myphone<double> y2;

y.GetMark();

y.SetPrice(8.6465665);

y.GetPrice();

y.Getpr();

PrintLine();

cout << "------------------------------------Тест шаблонного класса с 2 параметрами------------------------------------\n";

PrintLine();

cout << " float и char\n";

PrintLine();

Myphone1<float, char> f;

f.GetMark();

f.Getpr();

f.GetPrice();

f.GetReleaseY();

PrintLine();

Myphone1<float, char> v;

v.SetMark((char\*)"slor");

v.Setpr((char\*)"gagaga");

v.SetPrice(120.85);

v.GetMark();

v.Getpr();

v.GetPrice();

v.GetReleaseY();

PrintLine();

Myphone1<float, char> m;

m.SetMark((char\*)"slor");

m.Setpr((char\*)"gagaga");

m.SetPrice(13.8005);

m.GetMark();

m.Getpr();

m.GetPrice();

m.GetReleaseY();

PrintLine();

cout << "присваивание\n";

PrintLine();

m = v;

m.GetMark();

m.Getpr();

m.GetPrice();

m.GetReleaseY();

PrintLine();

cout << "сравнение\n";

PrintLine();

if (f > m) {

cout << "1й объект больше второго" << endl;

}

else {

cout << "1й объект не больше второго" << endl;

}

PrintLine();

PrintLine();

cout << " double и char\n";

PrintLine();

Myphone1<double, char> z;

z.SetPrice(1400);

z.GetMark();

z.Getpr();

z.GetPrice();

z.GetReleaseY();

PrintLine();

Myphone1<double, char> q;

q.SetPrice(1.6565656);

q.GetMark();

q.Getpr();

q.GetPrice();

q.GetReleaseY();

PrintLine();

Myphone1<double, char> u;

u.SetPrice(56.99);

u.GetMark();

u.Getpr();

u.GetPrice();

u.GetReleaseY();

PrintLine();

cout << "сравнение2\n";

PrintLine();

if (q == u) {

cout << "1й объект равен второму" << endl;

}

else {

cout << "1й объект не равен второму" << endl;

}

PrintLine();

cout << "сложение\n";

PrintLine();

u = q + z;

PrintLine();

/\*Pixel.destroy();\*/

system("pause");

return 0;

}

**Тестирование:**

