***Попов.Е.С***

***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 произ-водный класс с применением множественного наследования.

2. Проверьте работоспособность АТД и производных классов на тестовом наборе данных.

**Код программы:**

//#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);

}

//ОДИНОЧНОЕ НАСЛЕДОВАНИЕ

class Sensorniy : public Myphone<char>

{

public:

void SetOS(char\* a);

void GetOS();

void SetNfc(bool i);

void GetNfc();

Sensorniy() {

OS = new char[11];

strcpy(OS, "none");

}

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;

}

}

//МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ

class raskladnoi : public Myphone<char>

{

public:

raskladnoi() {

kolvo\_zgib = 2516000;

}

void GetZgib() {

cout <<"Гарантированное кол-во изгибов="<< kolvo\_zgib<<"раз\n";

}

void SetZgib(int n) {

kolvo\_zgib = n;

}

private:

int kolvo\_zgib;

};

class sgibayushiysa : public raskladnoi, public Sensorniy

{

public:

void GetDback() {

cout << "диагональ заднего экрана=" << Dback << "дюйма \n";

}

void SetDback(float n) {

Dback = n;

}

private:

float Dback;

};

//sgibayushiysa::sgibayushiysa()

//{

//

//}

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;

cout << "проверка работы множественного наследования\n";

sgibayushiysa f3;

f3.GetZgib();

f3.GetNfc();

f3.GetOS();

f3.SetZgib(12000);

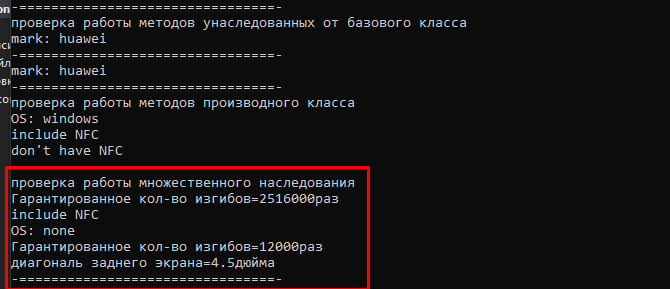
f3.GetZgib();

f3.SetDback(4.5);

f3.GetDback();

PrintLine();

Пример работы программы. Красным цветом выделена проверка работы программы с множественным наследованием.



Контрольные вопросы

1. С какой целью и в каких случаях используется множественное наследо-вание?

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)?

1. Множественное наследование применяется для получения производного класса от нескольких базовых классов. Это необходимо для того, чтобы не повторять один и тот же код несколько раз.
2. При множественном наследовании синтаксис заголовка следующий: имя класса, спецификатор доступа, имя класса от которого происходит наследование и через запятую перечисляются другие классы и их спецификаторы доступа
3. орграф, в котором отсутствуют направленные циклы, но могут быть «параллельные» пути, выходящие из одного узла и разными путями приходящие в конечный узел
4. Виртуальные базовые классы инициализируются (вызывается void-конструктор) перед любыми не виртуальными базовыми классами и в том порядке, в котором они появляются в ПАГе наследования при просмотре его снизу-вверх и слева направо.
5. Произойдет многократное копирование полей данных weight, price, color из предков класса Beefalo