*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 программы**

#include "pch.h"

#include <iostream>

#include <string.h>

using namespace std;

template <class T = int, class F = float> //шаблонный БК

class sendM

{

public:

sendM<T, F>(const char\* c, T b, F d, const char\* y);

sendM<T, F>(const sendM<T, F> &a);

sendM<T, F>();

~sendM<T, F>();

void print();

bool operator==(sendM<T, F> a);

sendM operator+(sendM<T, F> &a);

sendM &operator=(const sendM<T, F> &a);

protected:

char\* type;

T weight;

F price;

char\* marka;

};

class sendIN { // 2 БК

public:

sendIN(bool u);

sendIN();

sendIN(const sendIN &a);

~sendIN();

void print();

protected:

bool USB;

};

template <class T, class F> // реализация шаблонного БК (1 БК)

sendM<T, F>::sendM() {

type = new char[10];

strcpy(type, "-");

weight = 0;

price = 0;

marka = new char[10];

strcpy(marka, "-");

}

template <class T, class F>

sendM<T, F>::sendM(const sendM<T, F> &a) {

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

strcpy(type, a.type);

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

strcpy(marka, a.marka);

weight = a.weight;

price = a.price;

}

template <class T, class F>

sendM<T, F>::sendM(const char\* c, T b, F d, const char\* y)

{

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

strcpy(type, c);

weight = b;

price = d;

marka = new char[strlen(y) + 1];

strcpy(marka, y);

}

template <class T, class F>

sendM<T, F>::~sendM() {

type = NULL; marka = NULL;

}

template <class T, class F>

void sendM<T, F>::print()

{

cout << "Тип:" << type << endl;

cout << "Вес:" << weight << endl;

cout << "Стоимость:" << price << endl;

cout << "Марка:" << marka << endl;

}

template <class T, class F>

bool sendM<T, F>::operator==(sendM<T, F> a) {

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

weight == a.weight && price == a.price &&

(!strcmp(type, a.type)));

}

template <class T, class F>

sendM<T, F> sendM<T, F>::operator+(sendM<T, F> &a) {

sendM s;

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

strcpy(s.type, type);

strcat(s.type, a.type);

s.weight = weight + a.weight;

s.price = price + a.price;

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

strcpy(s.marka, marka);

strcat(s.marka, a.marka);

return s;

}

template <class T, class F>

sendM<T, F> &sendM<T, F>::operator=(const sendM<T, F> &a)

{

type = NULL;

type = a.type;

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

strcpy(type, a.type);

weight = a.weight;

price = a.price;

marka = NULL;

marka = a.marka;

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

strcpy(marka, a.marka);

return (\*this);

}

// реализация БК без шаблона (2 БК)

sendIN::sendIN() {

USB = 0;

}

sendIN::sendIN(bool u)

{

USB = u;

}

sendIN::~sendIN() {

}

void sendIN::print()

{

cout << "Вход USB(:\nДа=1\nНет=0\nВпишите значение=" << USB << endl;

}

// ПК без множественного наследования

template <class T, class F>

class sendrate : public sendM<double, int > {

public:

sendrate<T, F>(const char\* c, T b, F d, const char\* y, int r);

sendrate<T, F>(const sendrate<T, F> &a);

sendrate<T, F>();

~sendrate<T, F>();

void print();

sendrate& operator = (const sendrate<T, F> &a);

protected:

int R;

};

template <class T, class F>

sendrate<T, F>::sendrate(const char\* c, T b, F d, const char\* y, int r) :sendM<T, F>(c, b, d, y)

{

R = r;

}

template <class T, class F>

sendrate<T, F>::sendrate(const sendrate<T, F> &a) {

R = a.R;

this->type = new char[strlen(a.type) + 1];

strcpy(this->type, a.type);

this->marka = new char[strlen(a.marka) + 1];

strcpy(this->marka, a.marka);

this->weight = a.weight;

this->price = a.price;

}

template <class T, class F>

sendrate<T, F>& sendrate<T, F>::operator=(const sendrate &a) {

R = a.R;

this->price = a.price;

this->weight = a.weight;

this->type = new char[strlen(a.type) + 1];

strcpy(this->type, a.type);

this->marka = new char[strlen(a.marka) + 1];

strcpy(this->marka, a.marka);

return(\*this);

};

template <class T, class F>

sendrate<T, F>::sendrate() : sendM<T, F>() {

R = 0;

}

template <class T, class F>

void sendrate<T, F>::print()

{

sendM::print();

cout << "Рейтинг:" << R << endl;

}

template <class T, class F>

sendrate<T, F>::~sendrate() {

sendM<T, F>::~sendM();

}

template <class T, class F> // ПК с множ. наследованием

class sendPC :public sendIN, public sendM<T, F> {

public:

sendPC<T, F>(const char\* c, T b, F d, const char\* y, int r, bool u);

sendPC<T, F>();

sendPC<T, F>(const sendPC<T, F> &a);

void print();

~sendPC();

protected:

int R;

};

template <class T, class F>

sendPC<T, F>::sendPC() : sendIN(), sendM<T, F>()

{

R = 0;

USB = 0;

}

template <class T, class F>

sendPC<T, F>::sendPC(const char\* c, T b, F d, const char\* y, int r, bool u) :sendIN(u), sendM<T, F>(c, b, d, y) {

R = r;

USB = u;

}

template <class T, class F>

sendPC<T, F>::sendPC(const sendPC<T, F> &a) {

R = a.R;

this->type = new char[strlen(a.type) + 1];

strcpy(this->type, a.type);

this->marka = new char[strlen(a.marka) + 1];

strcpy(this->marka, a.marka);

this->weight = a.weight;

this->price = a.price;

this->USB = a.USB;

}

template <class T, class F>

void sendPC<T, F>::print()

{

sendM<T, F>::print();

sendIN::print();

cout << "Рейтинг:" << R << endl;

}

template <class T, class F>

sendPC<T, F>::~sendPC() {

sendM<T, F>::~sendM();

sendIN::~sendIN();

}

int main()

{

setlocale(LC\_ALL, "Russian");

char\* ty = new char[10];

char\* tyMN = new char[10];

char\* tyBC = new char[10];

int w, wMN, r, rMN, wBC;

double f, fMN, fBC;

bool vhod;

char\* mark = new char[10];

char\* markMN = new char[10];

char\* markBC = new char[10];

cout << "Базовый шаблонный класс:\n";

cout << "Введите тип модема: ";

cin >> tyBC;

cout << endl << "Введите вес: ";

cin >> wBC;

cout << endl << "Введите стоимость(руб.): ";

cin >> fBC;

cout << endl << "Введите марку модема: ";

cin >> markBC;

cout << endl;

sendM<double, int> e(tyBC, wBC, fBC, markBC);

e.print();

cout << '\t' << "Базовый шаблонный класс через производный(2БК)" << endl;

cout << "Введите тип модема: ";

cin >> ty;

cout << endl << "Введите вес: ";

cin >> w;

cout << endl << "Введите стоимость(руб.): ";

cin >> f;

cout << endl << "Введите марку модема: ";

cin >> mark;

cout << endl << "Введите рейтинг модема: ";

cin >> r;

cout << endl;

sendrate<double, int> a(ty, w, f, mark, r); //создание объекта a производного класса sendrate

a.print(); // вывод a

sendrate<double, int> b(a); // копирование объекта b производного класса sendrate

b.print(); // вывод b

if (a == b)

cout << "Устройства одинаковые\n";

else cout << "Устройства разные\n";

cout << "Операция =\n"; // проверка работы функции БК приравнивания

e = b;

e.print();

cout << endl;

cout << "Сложение\n"; // проверка работы функции БК сложения

e = a + b;

e.print();

cout << endl;

cout << '\t' << "Вывод производного класса с множественным наследованием" << endl;

cout << "Введите тип: ";

cin >> tyMN;

cout << endl << "Введите вес: ";

cin >> wMN;

cout << endl << "Введите стоимость(руб.): ";

cin >> fMN;

cout << endl << "Введите марку : ";

cin >> markMN;

cout << endl << "Введите рейтинг : ";

cin >> rMN;

cout << endl << "Ввод типа USB?:\nДа=1, Нет=0\n ";

cin >> vhod;

sendPC<double, int> c(tyMN, wMN, fMN, markMN, rMN, vhod); // создание объекта c производного класса с множественным наследованием sendPC

c.print();

cout << endl;

sendPC<double, int> d(c); // копирование объекта c производного класса с множественным наследованием sendPC

d.print();

cout << endl;

return 0;

}

# 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 и позволяет классу использовать их функции.

2. Синтаксис заголовка класса расширяется, чтобы можно было использовать список базовых классов с атрибутами доступа.

class производный: public БК\_1, protected БК\_2 { …}

3. Ориентированный ациклический граф – граф, узлы которого являются классами, а ориентированные ребра направлены от производных классов к базовым.

4. Виртуальные базовые классы инициируются перед любыми невиртуальными классами.

5. Ключевое слово virtual в классах Cow и Buffalo предотвращает многократное копирование полей данных weight, price, color из предков класса Beefalo. Так же виртуальные классы нельзя будет использовать.

# Тестовые данные

  
