Лабораторная работа 6

**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.

Задание:

Используя предыдущую программу, создайте новый производный класс с применением множественного наследования. Проверьте работоспособность АТД и производных классов на тестовом наборе данных.

**Ход работы:**

В качестве дополнительного класса для множественного наследования было решено создать класс-интерфейс Setinterface, который содержит функции для работы с объектами подкласса Setofitems и наследуется им в дополнение к классу Offurniture.

**Код программы:**

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

class Setinterface {

public:

virtual void additems(int s)=0;

virtual void definesepsale(bool k) = 0;

};

template <class T>

class Offurniture {

private:

T ID;

protected:

char \*name;

char \*country;

T cost;

public:

Offurniture();

Offurniture(char \*n);

Offurniture(char \*n, T a, char \*co, T c);

Offurniture(char \*n, char \*co, T c);

void print()const;

Offurniture(const Offurniture &m);

void input(char \*n, T a, char \*co, T c);

Offurniture &copy(const Offurniture &m);

Offurniture sum(Offurniture b);

bool comp(Offurniture b);

~Offurniture();

Offurniture operator +(const Offurniture m);

Offurniture operator +(T m);

Offurniture operator -(const Offurniture m);

Offurniture operator -(T m);

Offurniture operator =(const Offurniture m);

Offurniture operator =(T m);

bool operator <(Offurniture m);

bool operator <(T m);

bool operator >(Offurniture m);

bool operator >(T m);

bool operator ==(Offurniture m);

bool operator ==(T m);

};

class Setofitems : public Offurniture<double>, virtual public Setinterface {

private:

Offurniture::name;

Offurniture::country;

Offurniture::cost;

int amountofitems = 0;

bool partsale = 0;

public:

void print();

Setofitems(char\* n, char\* co, double c, int aoi, bool ps);

Setofitems();

Setofitems(const Setofitems& m);

Setofitems &copy(Setofitems &m);

void additems(int s) {

amountofitems += s;

};

void definesepsale(bool k) {

partsale = k;

};

Setofitems summary(Setofitems b);

void input(char\* n, char\* co, double c, int aoi, bool ps);

};

Setofitems Setofitems::summary(const Setofitems b) {

Setofitems h;

h.cost = cost + b.cost;

strcpy(h.name, "-");

strcpy(h.country, "-");

h.amountofitems = amountofitems + b.amountofitems;

h.partsale = partsale + b.partsale;

return h;

}

template <class T>

bool Offurniture<T>::comp(Offurniture<T> b) {

return (cost > b.cost);

}

template <class T>

Offurniture<T> &Offurniture<T>::copy(const Offurniture<T> &m) {

if (&m != this) {

delete[]name;

delete[]country;

name = new char[strlen(m.name) + 1];

strcpy(name, m.name);

ID = m.ID;

country = new char[strlen(m.country) + 1];

strcpy(country, m.country);

cost = m.cost;

}

return(\*this);

}

template <class T>

void Offurniture<T>::input(char \*n, T a, char \*co, T c) {

strcpy(name, n);

ID = a;

strcpy(country, co);

cost = c;

}

template <class T>

Offurniture<T>::Offurniture<T>() {

name = new char[3];

strcpy(name, "-");

ID = 0;

country = new char[3];

strcpy(country, "-");

cost = 0;

}

template <class T>

Offurniture<T>::Offurniture<T>(char\* n) {

name = new char[strlen(n) + 1];

strcpy(name, n);

ID = 0;

country = new char[2];

strcpy(country, "-");

cost = 0;

}

template <class T>

Offurniture<T>::~Offurniture<T>() {

delete[]name;

delete[]country;

}

template <class T>

Offurniture<T> Offurniture<T>::sum(const Offurniture<T> b) {

Offurniture h;

h.cost = cost + b.cost;

strcpy(h.name, "-");

strcpy(h.country, "-");

return h;

}

// Перегрузка операторов

template <class T>

Offurniture<T> Offurniture<T>::operator +(const Offurniture<T> m) {

Offurniture h;

h.cost = cost + m.cost;

strcpy(h.name, "-");

strcpy(h.country, "-");

return h;

}

template <class T>

Offurniture<T> Offurniture<T>::operator +(T m) {

cost = cost + m;

return(\*this);

}

template <class T>

Offurniture<T> Offurniture<T>::operator -(const Offurniture<T> m) {

Offurniture h;

h.cost = cost - m.cost;

strcpy(h.name, "-");

strcpy(h.country, "-");

return h;

}

template <class T>

Offurniture<T> Offurniture<T>::operator -(T m) {

cost = cost - m;

return(\*this);

}

template <class T>

Offurniture<T> Offurniture<T>::operator =(const Offurniture<T> m) {

strcpy(name, m.name);

ID = m.ID;

strcpy(country, m.country);

cost = m.cost;

return(\*this);

}

template <class T>

Offurniture<T> Offurniture<T>::operator =(T m) {

cost = m;

return(\*this);

}

template <class T>

bool Offurniture<T>::operator <(const Offurniture<T> m) {

return(cost<m.cost);

}

template <class T>

bool Offurniture<T>::operator <(T m) {

return(cost<m);

}

template <class T>

bool Offurniture<T>::operator >(const Offurniture<T> m) {

return(cost>m.cost);

}

template <class T>

bool Offurniture<T>::operator >(T m) {

return(cost>m);

}

template <class T>

bool Offurniture<T>::operator ==(const Offurniture<T> m) {

return(cost == m.cost);

}

template <class T>

bool Offurniture<T>::operator ==(T m) {

return(cost == m);

}

template <class T>

Offurniture<T>::Offurniture<T>(const Offurniture<T> &m) {

name = new char[strlen(m.name) + 1];

strcpy(name, m.name);

ID = m.ID;

country = new char[strlen(m.country) + 1];

strcpy(country, m.country);

cost = m.cost;

}

template <class T>

Offurniture<T>::Offurniture<T>(char \*n, T a, char \*co, T c) {

name = new char[strlen(n) + 1];

strcpy(name, n);

ID = a;

country = new char[strlen(co) + 1];

strcpy(country, co);

cost = c;

}

template <class T>

Offurniture<T>::Offurniture<T>(char \*n, char \*co, T c) {

name = new char[strlen(n) + 1];

strcpy(name, n);

country = new char[strlen(co) + 1];

strcpy(country, co);

cost = c;

}

template <class T>

void Offurniture<T>::print()const {

if (name!="") cout << "Название предмета мебели: " << name << endl;

if (ID >= 0) cout << "ID предмета мебели:" << ID << endl;

if (country != "") cout << "Страна-производитель: " << country << endl;

if (cost >= 0) cout << "Стоимость предмета мебели(в рублях) :" << cost << endl;

}

void Setofitems::print() {

Offurniture<double>::print();

cout << "Число предметов мебели в наборе: " << amountofitems << endl;

if (partsale) cout << "Покупка отдельных элементов набора возможна" << endl;

else cout << "Покупка отдельных элементов набора невозможна" << endl;

}

Setofitems::Setofitems(char \*n, char \*co, double c, int aoi, bool ps):Offurniture<double>(n, co, c){

amountofitems = aoi;

partsale = ps;

}

Setofitems::Setofitems() : Offurniture<double>(){

amountofitems = 0;

partsale = 0;

}

Setofitems::Setofitems(const Setofitems& m) {

name = new char[strlen(m.name) + 1];

strcpy(name, m.name);

amountofitems = m.amountofitems;

partsale = m.partsale;

country = new char[strlen(m.country) + 1];

strcpy(country, m.country);

cost = m.cost;

}

Setofitems& Setofitems::copy(Setofitems& m) {

if (&m != this) {

delete[]name;

delete[]country;

name = new char[strlen(m.name) + 1];

strcpy(name, m.name);

country = new char[strlen(m.country) + 1];

strcpy(country, m.country);

cost = m.cost;

amountofitems = m.amountofitems;

partsale = m.partsale;

}

return(\*this);

}

void Setofitems::input(char\* n, char\* co, double c, int aoi, bool ps) {

name = new char[strlen(n) + 1];

strcpy(name, n);

amountofitems = aoi;

partsale = ps;

country = new char[strlen(co) + 1];

strcpy(country, co);

cost = c;

}

int main()

{

setlocale(LC\_ALL, "russian");

Setofitems set1("Комплект мебели NOVA S", "Россия", 100000, 7, 1);

cout << "Объект set1 Производного класса Setofitems" << endl;

set1.print();

cout << endl;

Setofitems set2("Комплект мебели Суперэконом", "Россия", 100, 150, 0);

cout << "Объект set2 Производного класса Setofitems" << endl;

set2.print();

cout << endl;

cout << "Использование функции определения возможности раздельной продажи предметов definesepsale на объекте set1 (меняем значение на 0):" << endl;

set1.definesepsale(0);

set1.print();

cout << endl;

cout << "Объект set5 с параметрами по умолчанию, на котором была использована функция добавления предметов additems (добавлено 15 предметов):" << endl;

Setofitems set5;

set5.additems(15);

set5.print();

cout << endl;

if (set1 > 100) cout << "set1 больше 100 по стоимости согласно оператору сравнения базового класса" << endl;

else cout << "set1 меньше 100 по стоимости согласно оператору сравнения базового класса" << endl;

cout << endl;

cout << "Объект set4, являющийся суммой объектов set2 и set5, но с тремя дополнительными предметами и опцией покупки отдельных предметов:" << endl;

Setofitems set4;

set4.copy(set2.summary(set5));

set4.additems(3);

set4.definesepsale(1);

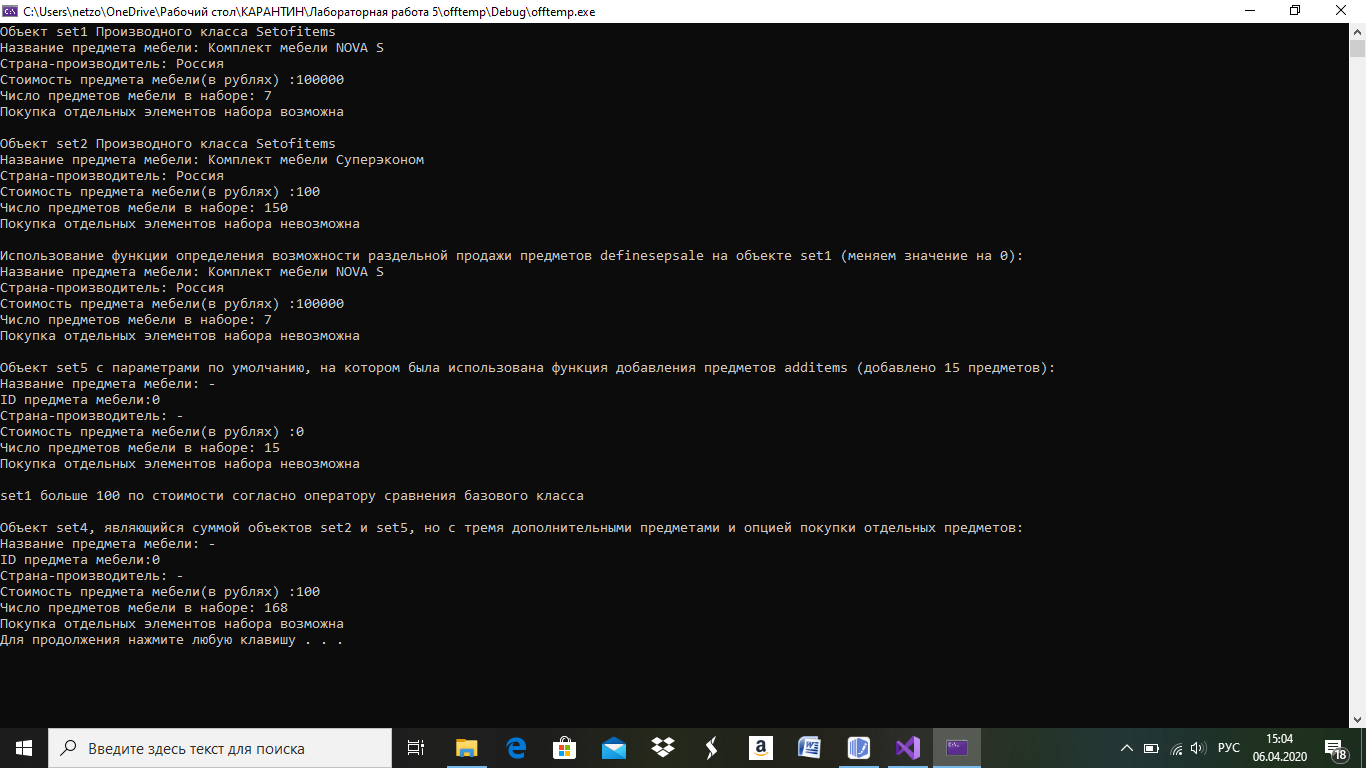
set4.print();

system("pause");

return 0;

}

**Тестирование:**



**Контрольные вопросы**

1. Множественное наследование используется для получения производного класса от нескольких базовых классов, что позволяет производному классу наследовать свойства и методы всех этих классов, и зачастую используется для объединения интерфейса с реализацией.
2. При множественном наследовании после имени класса ставится двоеточие, после которого через запятую перечисляются наследуемые подклассом классы, перед каждым из которых ставится модификатор доступа, определяющий тип наследования класса.
3. Ориентированный ациклический граф – это ориентированный граф, в котором нет направленных циклов, но могут существовать «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. Что будет, если из объявлений классов Cow и Buffalo убрать ключевое слово virtual (см. последний пример в теоретических положениях)?

Удаление ключевого слова virtual в классе Cow и классе Buffalo приведёт к многократному копированию полей данных weight, price, color из предков класса Beefalo, что остановит остановит программу, а компилятор выдаст ошибку неоднозначности.