Лабораторная работа 5

## 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.

Задание:

Создайте производный класс для АТД, реализованного по заданию лабораторной работы 4, используя одиночное наследование. Проверьте работоспособность АТД на тестовом наборе данных.

Код программы:

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

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);/\* возвращаемый тип&\*/

Offurniture& operator =(T m); /\* возвращаемый тип&\*/

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> {

private:

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();

Setofitems& copy(Setofitems& m);

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

};

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): Offurniture<double>(m) {

amountofitems = m.amountofitems;

partsale = m.partsale;

}

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);

}

Setofitems::~Setofitems() {

Offurniture<double>::~Offurniture<double>();

}

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");

Offurniture<int> a("Стол прямолинейный Эконом", 5413, "Россия", 1470);

cout << "Объект а базового класса c параметрами типа int" << endl;

a.print();

cout << endl;

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 << "Тестирование функции копирования производного класса (копия объекта set2):" << endl;

Setofitems set3;

set3.copy(set2);

set3.print();

cout << endl;

cout << "Тестирование функции копирования базового класса (копия объекта set1):" << endl;

Setofitems set4;

set4.Offurniture<double>::copy(set1);

set4.print();

cout << endl;

cout << "Использование оператора input БК и перегруженного оператора сложения БК на объекте set1:" << endl;

set1.input("Изменённое имя", "изменённая страна", 110, 40, 1);

set1 + 12345;

set1.print();

cout << endl;

cout << "Объект set5 с параметрами по умолчанию, на котором была использована функция input базового класса:" << endl;

Setofitems set5;

set5.Offurniture<double>::input("Комплект мебели", 0, "Китай", 680);

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;

system("pause");

return 0;

}

**Ответы на контрольные вопросы:**

1. Наследование – это механизм получения нового класса на основе существующего класса.
2. Существуют следующие модификаторы прав доступа к членам класса: public, pro-tected и private. Данные модификаторы могут применяться в объявлении класса в любом порядке и сколько угодно раз. Открытый член доступен во всей области видимости, где виден класс. Закрытый член доступен другим функциям-членам своего класса. Защищенный член доступен не только другим функциям-членам своего класса, но и функциям-членам класса, унаследованного непосредственно от данного класса.
3. При наследовании и инициализации членов класса конструкторы выполняются в следующем порядке:

Базовые классы инициализируются в порядке объявления.

Члены инициализируются в порядке объявления.

1. Чисто виртуальная функция – это виртуальная функция-член, тело которой не определено. Она объявляется внутри класса следующим образом: virtual прототип\_функции = 0; Класс с хотя бы одной виртуальной функцией является абстрактным и не может иметь объектов.
2. Для разыменования указателя на член класса используются два оператора: .\* и →\*.

**Тестирование:**

