МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Череповецкий государственный университет»

**Лабораторная работа № 2**

**«ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОТНОШЕНИЯ**

**МНОЖЕСТВЕННОГО НАСЛЕДОВАНИЯ»**

**Выполнил:**

студент гр. 1ИВТпб-01-21оп

Климов А.Г.  
**Проверил:** преподаватель

Ганичева О.Г.  
Отметка о зачете:

Череповец

2017 год

**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.

**Вариант**

A 16. Осветительные приборы.

**Задания**

1. Используя предыдущую программу, создайте новый производный класс с применением множественного наследования.

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

**Ход работы**

**Текст программы:**

**ClassLight.h**

// ClassLamps.h

// Блок защиты от повторного включения:

#ifndef LAMPS\_H\_INCLUDED

#define LAMPS\_H\_INCLUDED

#include "ClassLight.h"

#include <iostream>

#include <assert.h>

#include <signal.h>

class lamps :public light<int> {

public:

lamps() {

name3 = new char[strlen("Lamp name") + 1];

strcpy(name3, "Lamp name");

j = 0;

cost\_lamp = 18;

};

lamps(char \*name1, char name2, int k, float d, char \*nm3, float j) :light(name1, name2, k, d) {

name3 = new char[strlen(nm3) + 1];

strcpy(name3, nm3);

this->j = j;

}

void print\_m();

~lamps() { delete[]name3; } //деструктор

private:

char\* name3;

float j;

int \*p;

int size;

protected:

float cost\_lamp=0; // цена лампы прибора

float cost\_lamp\_2=20;

};

#endif //MY\_SYMBOL\_H

**ClassLight.cpp**

#include "ClassLight.h"

#include <iostream>

#include <string>

template<typename T> void light<T>::input(char \*nm1, char nm2, int i, float j)

{

name1 = new char[strlen(nm1) + 1];

strcpy(name1, nm1);

name2 = nm2;

k = i;

d = j;

}

template<typename T> void light<T>::print()

{

std::cout << "Название: " << name1 << std::endl;

std::cout << "Цоколь: " << name2 << k << std::endl;

std::cout << "Яркость света: " << d << std::endl;

}

template<typename T> void light<T>::prisv(light b)

{

delete[]name1;

name1 = new char[strlen(b.name1) + 1];

strcpy(name1, b.name1);

name2 = b.name2;

k = b.k;

d = b.d;

}

template<typename T> bool light<T>::sravn(light b)

{

return ((strcmp(name1, b.name1) == 0) && (name2 == b.name2) && (k == b.k) && (d == b.d)) ? true : false;

}

//конструкторы

template<typename T> light<T>::light() {

name1 = new char[strlen("Unknown") + 1];

strcpy(name1, "Unknown");

name2 = 'E';

k = 0;

d = 0;

cost\_light = 555;

}

template<typename T> light<T>::light(char \*nm1, char nm2, T i, float j) {

name1 = new char[strlen(nm1) + 1];

strcpy(name1, nm1);

name2 = nm2;

k = i;

d = j;

}

template<typename T> light<T>::light(const light &t) {

name1 = new char[strlen(t.name1) + 1];

strcpy(name1, t.name1);

name2 = t.name2;

k = t.k;

d = t.d;

}

//перегрузка операторов

template<typename T> light<T> light<T>::operator++() {

k++;

d++;

return (\*this);

}

template<typename T> light<T> operator-(light<T> a, light<T> b) {

light<T> h;

h.k = a.k - b.k;

h.d = a.d - b.d;

return(h);

}

template<typename T> light<T>& light<T>::operator=(const light &m) {

k = m.k;

d = m.d;

delete[]name1;

name1 = new char[strlen(m.name1) + 1];

strcpy(name1, m.name1);

return(\*this);

}

template class light<int>;

template class light<float>;

template class light<char>;

**ClassLamps.h**

// ClassLamps.h

// Блок защиты от повторного включения:

#ifndef LAMPS\_H\_INCLUDED

#define LAMPS\_H\_INCLUDED

#include "ClassLight.h"

#include <iostream>

#include <assert.h>

#include <signal.h>

class lamps :public light<int> {

public:

lamps() {

name3 = new char[strlen("Lamp name") + 1];

strcpy(name3, "Lamp name");

j = 0;

cost\_lamp = 18;

};

lamps(char \*name1, char name2, int k, float d, char \*nm3, float j) :light(name1, name2, k, d) {

name3 = new char[strlen(nm3) + 1];

strcpy(name3, nm3);

this->j = j;

}

void print\_m();

~lamps() { delete[]name3; } //деструктор

private:

char\* name3;

float j;

int \*p;

int size;

protected:

float cost\_lamp=0; // цена лампы прибора

float cost\_lamp\_2=20;

};

#endif //MY\_SYMBOL\_H

**ClassLamps.cpp**

#include "ClassLamps.h"

#include <iostream>

void lamps::print\_m() {

light::print();

std::cout << name3<< " " << j;

std::cout << std::endl;

}

**ClassTest.h**

#include "ClassLight.h"

#include "ClassLamps.h"

#include <iostream>

class light\_lamps :public light<int>, public lamps {

public:

light\_lamps() {

std::cout << "Инициализация light\_lamps\n"; }

void light\_lamps\_addname(char\* nm4)

{

name4 = new char[strlen(nm4) + 1];

strcpy(name4, nm4);

}

void print\_ss() {

std::cout << "Название light\_lamps: " << name4 << std::endl;

};

int cost\_light\_and\_lamp() { return light<int>::cost\_light+lamps::cost\_lamp; }

private:

char \*name4;

};

**Main.cpp**

#include <iostream>

#include "ClassLight.h"

#include "ClassLamps.h"

#include "ClassTest.h"

using namespace std;

void main()

{

setlocale(LC\_ALL, "Russian");

lamps l1("Светильник1", 'E', 40, 10.8, "Лампа", 22);

lamps l2("Светильник2", 'E', 40, 10.8, "Лампа", 22);

l1.print\_m();

cout << endl;

l2.print\_m();

cout << endl;

light\_lamps z1;

cout <<"Вывод цены осветительного прибора и лампы: "<< z1.cost\_light\_and\_lamp()<<endl;

cout << endl;

z1.light\_lamps\_addname("Название345435");

z1.print\_ss();

cout << endl;

//l1.print();

system("pause");

}

**Результаты тестирования:**

*Входные данные 1 (шаблон класса light для k):*

lamps l1("Светильник1", 'E', 40, 10.8, "Лампа", 22);

lamps l2("Светильник2", 'E', 40, 10.8, "Лампа", 22);

cout <<"Вывод цены осветительного прибора и лампы: "<< z1.cost\_light\_and\_lamp()<<endl;

z1.light\_lamps\_addname("Название345435");

*Результат 1 Рис.1:*

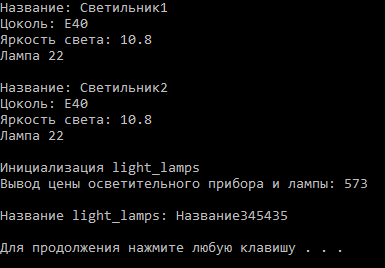


Рис. 1. Результат 1

**Ответы на контрольные вопросы:**

1. С какой целью и в каких случаях используется множественное наследование?

Множественное наследование (multiple inheritance) делает возможным получение производного класса от нескольких базовых классов.

2. Опишите синтаксис заголовка производного класса при множественном наследовании.

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

Например:

class shape {

// класс для интерфейса фигуры

};

class tview {

// класс, реализующий просмотр текста

};

class tshape : public shape, private tview {

// адаптер текстового просмотра,

// позволяющий просматривать фигуры

};

В этом примере производный класс tshape открыто наследует базовый класс shape и закрыто наследует класс tview. Подобная схема создания классов называется адаптерной схемой. Она использует множественное наследование для объединения интерфейса с реализацией. Поэтому такой прием создания классов еще называют классом-примесью.

3. Дайте определение ориентированного ациклического графа.

Родительские отношения между классами описываются ориентированным ациклическим графом (DAG) наследования. DAG – это граф, узлы которого являются классами, а ориентированные ребра направлены от производных классов к базовым.

Если члены различных классов имеют идентичные имена, то при создании от этих классов производного класса могут возникнуть неопределенности. Подобное наследование допускается при условии, что пользователь не делает двусмысленных ссылок на такой член.

4. Где инициируются виртуальные базовые классы?

При множественном наследовании два базовых класса могут быть получены от общего предка. Если оба базовых класса используются обычным образом своим производным классом, то такой класс будет иметь два подобъекта общего предка. Подобного дублирования можно избежать с помощью виртуального наследования. Например:

class DomesticAnimal {

protected:

int weight; float price; char color[20];

public:

DomesticAnimal(void) {weight=0; price=0.; strcpy(color,'none'); }

DomesticAnimal(int aweight, float aprice,char \*acolor) {

weight=aweight; price=aprice; strcpy(color,acolor);

}

virtual void print(void) { cout << weight << price << color; }

};

class Cow: public virtual DomesticAnimal {

public:

Cow(void) { }

Cow(int aweight,float aprice,char \*acolor) {

weight=aweight; price=aprice; strcpy(color,acolor); }

void print(void) { cout << ' Cow has propeties'; DomesticAnimal::print(); }

};

class Buffalo: public virtual DomesticAnimal {

public:

Buffalo(void) { }

Buffalo(int aweight,float price,char \*acolor) {

weight=aweight; price=aprice; strcpy(color,acolor); }

void print(void) { cout<< ' Buffalo has propeties';

DomesticAnimal::print(); }

};

class Beefalo: public Cow,public Buffalo {

public:

Beefalo(int aweight,float aprice,char \*acolor) {

weight=aweight; price=aprice; strcpy(color,acolor); }

void print(void) { cout << ' beefalo has propeties';

DomesticAnimal::print(); }

};

main() {

Cow aCow(1400,375.0,'black and white'); // void const Domestic

// const par Cow

Beefalo aBeefalo(1700,525.0,'Brown and black'); // void cons Domestic

// void const Cow

// void const Buffalo

// const par Beefalo

DomesticAnimal& myCow=aCow;

DomesticAnimal& myBeefalo=aBeefalo;

my.Cow.print();

myBeefalo.print();

}

Код приведенной выше программы решает проблему класса Beefalo, который наследует поля данных weight, price, color. Объекты класса Beefalo имеют по одному полю данных для веса, цены и цвета.

Виртуальные базовые классы инициализируются (вызывается void-конструктор) перед любыми невиртуальными 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)?

Если виртуальный базовый класс имеет хотя бы один конструктор, то он должен иметь void-конструктор.

Ключевое слово virtual в классе Cow и классе Buffalo предотвращает многократное копирование полей данных weight, price, color из предков класса Beefalo.

**Вывод:** изучил принципы и механизмы множественного наследования, правила доступа к базовым классам; приобрёл практические навыки работы с базовыми и производными классами при множественном наследовании.