МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Череповецкий государственный университет»

**Лабораторная работа № 1**

**«ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОТНОШЕНИЯ ПРОСТОГО НАСЛЕДОВАНИЯ»**

**Выполнил:**

студент гр. 1ИВТпб-01-21оп

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

Ганичева О.Г.  
Отметка о зачете:

Череповец

2017 год

**Цель работы:** изучить способы создания производного класса и особенности работы с ним, правила инициализации и доступа к элементам производного класса; приобрести практические навыки наследования.

**Вариант**

A 16. Осветительные приборы.

**Задания**

1. Создайте производный класс для АТД, реализованного по заданию лабораторной работы 7, используя одиночное наследование.

2. Проверьте работоспособность АТД и производного класса на тестовом наборе данных.

**Ход работы**

**Текст программы:**

**ClassLight.h**

// ClassLight.h

// Блок защиты от повторного включения:

#ifndef LIGHT\_H\_INCLUDED

#define LIGHT\_H\_INCLUDED

#include <assert.h>

template <class T> class light

{

public:// набор функций

light& operator=(const light &m);

friend light operator-(light a, light b);

light operator++();

//

light(); //конструктор по умолчанию

light(char \*nm1, char nm2, T i, float j); //конструктор с параметрами

light(const light &t); //конструктор с const, меняющий содержимое, к которому получает доступ (копирующий конструктор)

~light() { delete[]name1; } //деструктор

//

void input(char \*nm1, char nm2, int i, float j);

void print();

void prisv(light b);

bool sravn(light b);

void destr() { delete[]name1; } // уничтожение объекто

private:

char \*name1; // название осветительного прибора

char name2; // тип лампочки

T k; // размер цоколя

float d; // яркость света

};

#endif //LIGHT\_H\_INCLUDED

**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;

}

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;

};

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;

};

#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;

}

**Main.cpp**

#include <iostream>

#include <iostream>

#include "ClassLight.h"

#include "ClassLamps.h"

using namespace std;

void main()

{

setlocale(LC\_ALL, "Russian");

light <int> t1("Светильник1", 'E', 77, 10.8);

t1.print();

cout << endl;

light <float> t2("Светильник2", 'E', 77, 10.8);

t2.print();

cout << endl;

light <char> t3("Светильник3", 'E', 77, 10.8);

t3.print();

cout << endl;

lamps l1("Светильник1", 'F', 42, 11.4, "Лампа", 55);

l1.print\_m();

cout << endl;

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

l2.print\_m();

cout << endl;

system("pause");

}

**Результаты тестирования:**

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

light <int> t1("Светильник1", 'E', 77, 10.8);

t1.print();

cout << endl;

light <float> t2("Светильник2", 'E', 77, 10.8);

t2.print();

cout << endl;

light <char> t3("Светильник3", 'E', 77, 10.8);

t3.print();

cout << endl;

lamps l1("Светильник1", 'F', 42, 11.4, "Лампа", 55);

l1.print\_m();

cout << endl;

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

l2.print\_m();

cout << endl;

*Результат 1 Рис.1:*

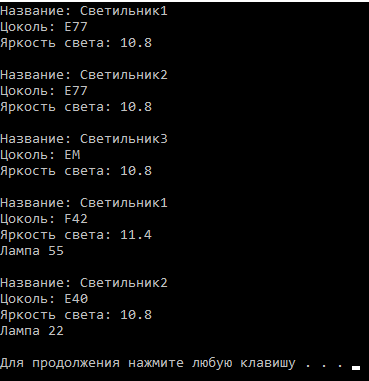


Рис. 1. Результат 1

**Ответы на контрольные вопросы:**

1. Дайте определение наследования.

Наследование (inheritance) – это механизм получения нового класса на основе существующего класса. Существующий класс может быть дополнен или изменен для создания производного класса. Наследование осуществляется с помощью конструкции:

class имя\_класса:

public|protected|private)opt имя\_базового\_класса

{

объявления членов

};

2. Какие модификаторы прав доступа к производному классу вы знаете и в чем их особенности?

Модификаторы прав доступа к членам класса (public, protected и private) 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. Если все члены открыты, то ключевое слово class можно заменить ключевым словом struct.

Список\_арг\_базового\_класса используется при вызове соответствующего конструктора базового класса и реализуется перед тем как будет выполняться тело конструктора производного класса.

Открыто наследуемый класс является подтипом своего базового класса. Во многих случаях переменная производного класса может рассматриваться как переменная типа базового класса. Указатель, тип которого «указатель на базовый класс», может указывать на объекты открыто наследуемого класса. Ссылка на производный класс неявно может быть преобразована в ссылку на открытый базовый класс. Можно объявить ссылку на базовый класс и инициализировать ее ссылкой на объект открыто наследуемого класса.

3. Как выполняется конструктор при наследовании?

При наследовании и инициализации членов класса конструкторы выполняются в следующем порядке:

1. Базовые классы инициализируются в порядке объявления.

2. Члены инициализируются в порядке объявления.

Виртуальные базовые классы создаются до того, как создан любой из производных классов, и до того, как созданы невиртуальные базовые классы. Порядок их создания – «из глубины, слева направо». Деструкторы вызываются в обратном выполнению конструкторов порядке.

4. Дайте определение чисто виртуальной функции.

Чисто виртуальная функция – это виртуальная функция-член, тело которой не определено. Она объявляется внутри класса:

virtual прототип\_функции = 0;

Класс, который имеет хотя бы одну виртуальную функцию, называется абстрактным базовым классом. Нельзя объявлять переменные абстрактного базового класса, но можно объявлять и полиморфно использовать указатели на такой базовый класс. Чисто виртуальный деструктор должен иметь определение.

5. Какие операторы используются для разыменования указателя на член класса?

Указатель на член класса имеет тип T::\*, где T – имя класса. Для разыменования указателя на член класса используются два оператора: .\* и →\*.

Рассмотрим, например, выражение z.\*fet. Здесь сначала разыменовывается указатель для получения переменной-члена, а затем происходит доступ к члену объекта z.

Виртуальные функции:

Ключевое словоvirtual служит спецификатором функции, который обеспечивает механизм для динамического выбора на этапе выполнения подходящей функции-члена среди функций базового и производного классов. Оно может применяться для изменения объявлений только функций-членов. Виртуальная функция должна иметь исполняемое тело и имеет семантику вызова. Виртуальная функция может замещаться в производном классе. Выбор того, какое определение функции вызвать для виртуальной функции, происходит динамически на этапе выполнения. Самый распространенный случай – это когда базовый класс содержит виртуальную функцию, а производные классы имеют свои версии этой функции. Указатель базового класса может указывать либо на объект базового класса, либо на объект производного класса. Выбор вызываемой функции-члена будет произведен на этапе выполнения и будет зависеть от типа объекта, а не от типа указателя. При отсутствии члена производного типа по умолчанию используется виртуальная функция базового класса.

**Вывод:** изучил способы создания производного класса и особенности работы с ним, правила инициализации и доступа к элементам производного класса; приобрёл практические навыки наследования.