Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Уфимский государственный авиационный технический университет»

Факультет информатики и робототехники

Кафедра вычислительной математики и кибернетики

Отчет по лабораторной работе №5

по дисциплине «Объектно-ориентированное программирование»

на тему: «Жизненный цикл объектов С++ и Виртуальность»

Выполнил:

Студент группы ПРО-228Б И.И.Гареев

Проверил: В.А. Котельников

Уфа – 2022

Содержание

[1 Цель работы 3](#_Toc93780556)

[2 Программная часть 4](#_Toc93780557)

[2.1 Класс Animal 4](#_Toc93780558)

[2.2 Класс Cat 4](#_Toc93780559)

[2.3 Класс Base и Desc 4](#_Toc93780560)

[2.4 Функции func1, func2, func3 4](#_Toc93780561)

[2.5 Функции func\_1, func\_2, func\_3 4](#_Toc93780562)

[2.6 Вызов перекрываемого, виртуального и наследуемого методов 5](#_Toc93780563)

[2.7 Проверка на принадлежность некоторому классу и приведение типов 5](#_Toc93780564)

[2.8 Создание класса Base и его потомка Desc 6](#_Toc93780565)

[2.9 Умные указатели 6](#_Toc93780566)

[3 Заключение 7](#_Toc93780567)

[4 Приложение А 8](#_Toc93780568)

# Цель работы

Определение и реализация тестовых классов, и написание программы, иллюстрирующей их использование.

Определения

* перекрываемых методов
* виртуальных методов, в т.ч. деструкторов

Реализация

* проверки на принадлежность некоторому классу
* безопасного приведения типов (dynamic\_cast)
* безопасного приведения типов (вручную)

Передача объектов как параметров в функции и возвращения объектов как результата из функции, контроль их жизненного цикла.

Умные указатели unique\_ptr и shared\_ptr.

# Программная часть

## Класс Animal

Создаю класс Animal, в котором есть виртуальный метод sound, метод foo1, который будет переопределяться, метод foo2, который будет наследоваться, виртуальный метод foo3, виртуальный метод string classname, виртуальный метод bool isA и деструктор ~Animal.

## Класс Dog

Создаю класс Dog, который наследуется от Animal, в котором есть конструктор без параметров, перекрытый метод sound, переопределенный метод foo1, перекрытый метод foo3, isA, string classname, метод run и деструктор ~Dog.

## Класс Base и Desc

Создаю простой класс Base и его наследник Desc, в которых есть конструкторы без параметров, с параметрами и копирования.

## Функции func1, func2, func3

Создаю три функции func1, func2, func3 для проверки механизма передачи объектов как параметров.

## Функции func\_1, func\_2, func\_3

Создаю три функции func\_1, func\_2, func\_3 для проверки механизма возврата объектов из функции.

## Вызов перекрываемого, виртуального и наследуемого методов

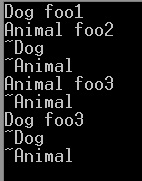


Рисунок .

На Рисунок 2.6 создаю объект класса наследника, вызываю у него переопределяемый метод foo1 и наследуемый метод foo2, потом удаляю. Далее создаю объект базового класса и вызываю у него виртуальный метод, потом удаляю. Создаю объект класса наследника, помещенного в переменную базового класса и вызываю у него наследуемый виртуальный метод, потом удаляю.

## Проверка на принадлежность некоторому классу и приведение типов

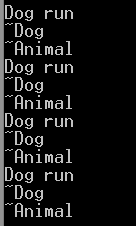


Рисунок .

На Рисунок 2.2 создаю объект класса наследника, помещенного в переменную базового класса и с помощью метода classname проверяю принадлежность к классу, если принадлежит, то реализовываю приведение типов, потом удаляю. Создаю объект класса наследника, помещенного в переменную базового класса и с помощью метода isA проверяю принадлежность к классу, если принадлежит, то реализовываю безопасное приведение типов, потом удаляю. Создаю объект класса наследника, помещенного в переменную базового класса и реализовываю опасное приведение типов, потом удаляю. Создаю объект класса наследника, помещенного в переменную базового класса и с помощью dynamic\_cast реализовываю безопасное приведение типов, потом удаляю.

## Создание класса Base и его потомка Desc

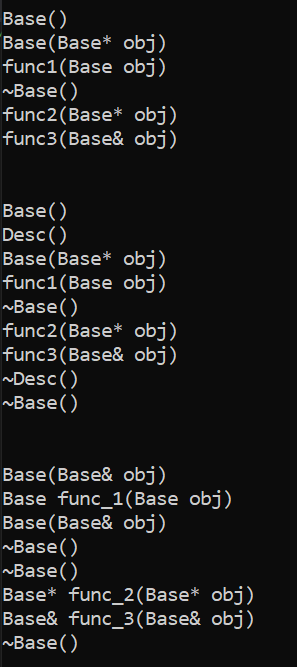


Рисунок .

На Рисунок 2.3 показан результат работы программы. В данной программе реализовано: передача объектов как параметров функции, проверка механизма передачи объектов как параметров в функции, проверка механизмов возврата объектов из функции.

## Умные указатели

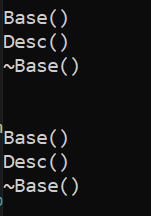


Рисунок .

На Рисунок 2.4 показан результат работы программы. В данной программе реализовано: использование умных указателей unique\_ptr и shared\_ptr, создание объектов, управляемых с помощью умных указателей.

# Заключение

В ходе данной лабораторной работы я научился перекрывать и переопределять методы, проверять на принадлежность к классу, реализовывать безопасное приведение типов и работать с умными указателями.

https://github.co

# Приложение А

#include <iostream>

#include<conio.h>

class Point {

protected:

int x, y;

public:

Point() { //конструктор

printf("Point()\n"); //отладочный вывод

x = 0;

y = 0;

}

Point(int x, int y) { //конструктор с параметрами

printf("Point(int x, int y)\n");

this->x = x;

this->y = y;

}

Point(const Point& p) { //конструктор копирования

printf("Point(const Point &p)\n");

x = p.x;

y = p.y;

}

virtual ~Point() { //деструктор

printf("%d, %d\n", x, y); //вывод текущего состояния объекта

printf("~Point()\n");

}

void move(int dx, int dy) {

x = x + dx;

y = y + dy;

}

void reset();

};

void Point::reset() {

x = 0;

y = 0;

}

class ColoredPoint : public Point { //класс потомок

protected:

int color;

public:

ColoredPoint() : Point() {

printf("ColoredPoint()\n");

color = 0;

}

ColoredPoint(int x, int y, int color) : Point(x, y) {

printf("ColoredPoint(int x, int y)\n");

this->color = color;

}

ColoredPoint(const ColoredPoint& p) {

printf("ColoredPoint(const Point &p)\n");

x = p.x;

y = p.y;

color = p.color;

}

~ColoredPoint() {

printf("%d, %d color=%d\n", x, y, color); //вывод текущего состояния объекта

printf("~ColoredPoint()\n");

}

void ChangeColor(int NewColor) {

color = NewColor;

}

};

class Section {

protected:

Point\* p1;

Point\* p2;

public:

Section() { //конструктор

printf("Section()\n"); //отладочный вывод

p1 = new Point;

p2 = new Point;

}

Section(int x1, int y1, int x2, int y2) { //конструктор с параметрами

printf("Section(int x1, int y1, int x2, int y2)\n");

p1 = new Point(x1, y1);

p2 = new Point(x2, y2);

}

Section(const Section& s) { //конструктор копирования

printf("Section(const Section &s)\n");

p1 = new Point(\*(s.p1));

p2 = new Point(\*(s.p2));

}

~Section() { //деструктор

delete p1;

delete p2;

printf("~Section()\n");

}

};

int main()

{

setlocale(LC\_ALL, "rus");

{

printf("Статическое создание объектов и их удаление\n");

Point p;

Point p2(1, 5);

Point p3(p2);

}

printf("--------------------------------------------------------------\n");

{

printf("Динамическое создание объектов и их удаление\n");

Point\* p = new Point;

Point\* p2 = new Point(7, 7);

Point\* p3 = new Point(\*p2);

delete p;

delete p2;

delete p3;

}

printf("--------------------------------------------------------------\n");

{

printf("Изменение координат объектов с заданными координатами\n");

Point\* p = new Point(7, 17);

Point\* p2 = new Point(4, 4);

Point\* p3 = new Point(\*p2);

p->move(3, 3);

p2->move(1, 1);

p3->reset();

p3->move(2, 2);

delete p;

delete p2;

delete p3;

}

printf("--------------------------------------------------------------\n");

{

printf("Объекты класса наследника\n");

ColoredPoint\* p = new ColoredPoint(3, 4, 87);

ColoredPoint\* p2 = new ColoredPoint(1, 2, 3);

ColoredPoint\* p3 = new ColoredPoint(\*p2);

delete p;

delete p2;

delete p3;

}

printf("--------------------------------------------------------------\n");

{

printf("Помещение объектов в переменные различных типов\n");

Point\* p1 = new ColoredPoint(3, 4, 87);

ColoredPoint\* p2 = new ColoredPoint(1, 2, 3);

p1->reset();

p2->ChangeColor(4);

delete p1;

delete p2;

}

printf("--------------------------------------------------------------\n");

{

printf("Композиция\n");

Section\* s1 = new Section(1, 2, 3, 4);

Section\* s2 = new Section(\*s1);

delete s1;

delete s2;

}

\_getch();

}