ФГБОУ ВО

Уфимский университет науки и технологий

Кафедра ВМиК

Отчет

по лабораторной работе №2

по теме “Объекты и классы”

по дисциплине

**«Объектно-ориентированное программирование»**

Выполнил: ст.гр. ПРО-231Б

Семенов Г.Д.

Принял:

Котельников В.А.

Уфа 2023 г.

**Задание:**

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

**Ход работы**

Создание и удаление статически созданных объектов:

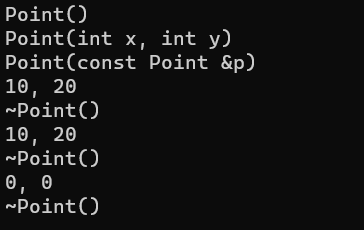


Рисунок 1. Создание объектов производится с помощью конструктора без параметров, конструктора с параметрами и копирующего конструктора.

Использование методов класса:

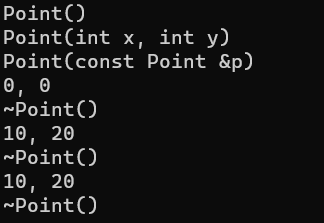


Рисунок 2. На объекте P2 применяется сначала применяется метод reset(), сбрасывающий параметры объекта до (0,0), позже применяется метод move(10, 20), меняющий параметры объекта на 10 и 20 соответственно.

Создание и удаление объектов классов-наследников:

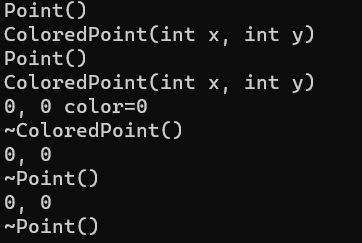


Рисунок 3. Создание и удаление объектов класса-наследника ColoredPoint.

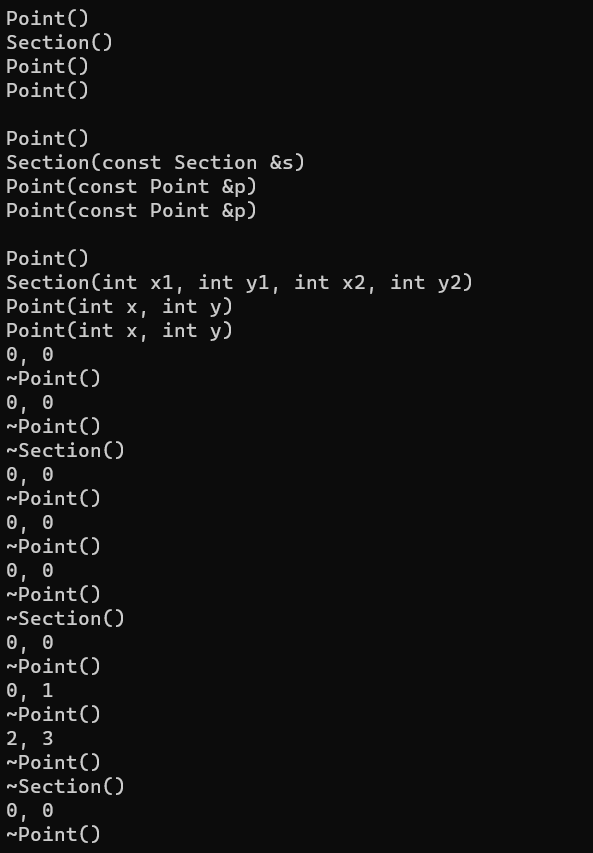
****

Рисунок 4. Создание и удаление класса section, основанного на композиции объектов класса Point.

**Заключение**

Таким образом, мы научились определению и реализации тестовых классов, и написали программу, иллюстрирующую их использование. (код, делающий нечто, и при этом обязательный отладочный вывод в консоль, для контроля того, какие методы вызываются – особенно важен отладочный вывод во всех конструкторах и деструкторах).

**Приложение 1.**

**Код**

#include <iostream>

using namespace std;

class Point {

protected:

int x, y;

public:

Point() { //Конструктор без параметров

printf("Point()\n");

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) { //Копирующий конструктор

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

x = p.x;

y = p.y;

}

~Point() { //Деструктор

printf("%d, %d\n", x, y);

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

}

void move(int dx, int dy) { //Метод в определении

x = x + dx;

y = y + dy;

}

void reset();

};

void Point::reset() { //Метод после определения

x = 0;

y = 0;

}

class ColoredPoint : public Point { //Класс-наследник

protected:

int color;

public:

ColoredPoint() : Point() {

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

color = p.color;

x = p.x;

y = p.y;

}

~ColoredPoint() {

printf("%d, %d color=%d\n", x, y, color);

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

}

void change\_color(int new\_color) {

color = new\_color;

}

};

class Section {

protected:

Point \*p1;

Point \*p2;

Point p3;

public:

Section() {

printf("Section()\n");

p1 = new Point;

p2 = new Point;

}

Section(int x1, int y1, int x2, int y2) {

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) { //Композиция

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

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

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

p3 = s.p3;

}

~Section() {

delete p1;

delete p2;

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

}

};

int main()

{

//Создание и удаление статических объектов

{

Point p;

Point p2(10, 20);

Point p3(p2);

}

printf("\n");

//Создание и удаление динамических объектов

Point\* p4 = new Point();

Point\* p5 = new Point(10, 20);

Point\* p6 = new Point(\*p5);

//Использование методов класса

p5->reset();

p5->move(10, 20);

delete p4;

delete p5;

delete p6;

printf("\n");

//Создание и удаление классов-наследников

ColoredPoint\* cp1 = new ColoredPoint();

Point\* cp2 = new ColoredPoint();

delete cp1;

delete cp2;

printf("\n");

//Создание и удаление композиций

Section\* s1 = new Section;

printf("\n");

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

printf("\n");

Section\* s3 = new Section(0, 1, 2, 3);

delete s1;

delete s2;

delete s3;

printf("\n");

system("Pause");

}