Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Факультет информатики и робототехники

Кафедра вычислительной математики и кибернетики

Лабораторная работа №2

“Объекты и Классы”

По дисциплине “Объектно-ориентированное программирование”

Выполнил:

Студент группы ПРО-231Б

Гулин Г.В.

Проверил:

доцент кафедры ВМиК

Котельников В.А.

Уфа-2023

**Цель работы**

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

**Задание**

* Определения
  + классов
  + атрибутов, методов
  + классов-наследников
  + конструкторов (обязательно несколько: без параметров, с параметрами, с параметром-объектом того же класса), деструкторов
  + доступности атрибутов и методов (показать, как влияют модификаторы private, protected, public)
* Реализация
  + методов объектов
  + реализация методов сразу в определении или после определения
  + конструкторов, деструкторов
* Создание и использование (обращение к атрибутам, вызов методов)
  + статически создаваемых объектов («MyClass obj;»)
  + динамически создаваемых объектов («MyClass \*obj = new MyClass();»)
  + объектов с помощью различных конструкторов (**у каждого** создаваемого объекта должны быть: конструктор без параметров, с параметрами, с параметром-объектом того же класса – конструктор копирования)
  + инициализация полей класса в списке инициализации конструктора
  + помещение объектов в переменные различных типов (объяснять, чем отличается MyBase \* obj = new MyBase() от MyBase \* obj = new MyDeriv())
  + объектов классов-наследников (проверить и продемонстрировать, какие конструкторы классов при этом вызываются)
  + композиция объектов: атрибутом одного объекта класса A является другой объект класса B (не указатель!), инициализируемый в списке инициализации конструктора класса A; показать, когда этот объект класса B удаляется
  + композиция объектов: атрибутом одного объекта класса A является указатель на другой объект класса B, создаваемый в конструкторе класса A и уничтожаемый в деструкторе класса A; композируемые классы должны определяться отдельно (не один в другом); при композиции показать, в чем разница, если объект класса А хранит прямо объект класса В или указатель на объект класса В.
* Уничтожение
  + статически созданных объектов
  + динамически созданных объектов
  + объектов классов-наследников (проверить и продемонстрировать, какие деструкторы классов при этом вызываются)

**Ход работы**

1. Создание объектов на стеке

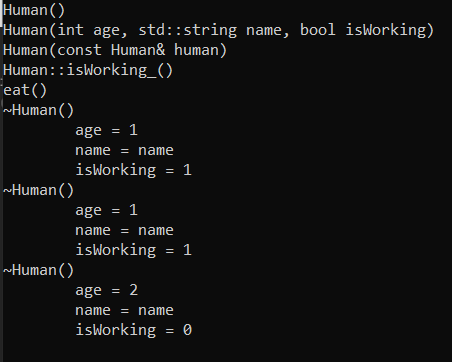


Рисунок 1. Отладочный вывод создания объектов на стеке

1. Создание объектов на куче

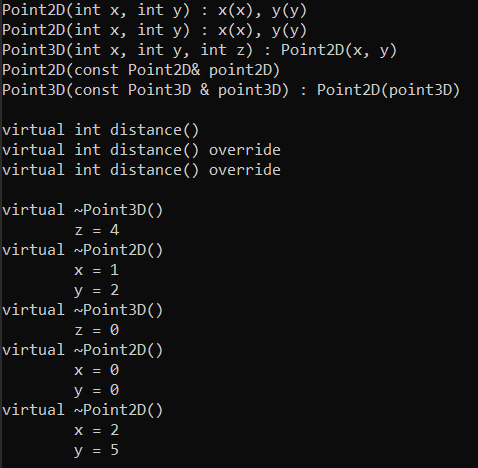


Рисунок 2. Отладочный вывод создания объектов на куче

1. Создание дочернего класса и его объектов, помещение объектов в переменные их родительских классов.

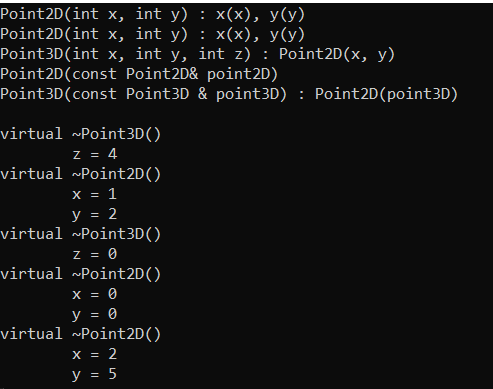


Рисунок 3. Отладочный вывод создания объектов родительского и дочернего классов, помещение в переменную класса его потомков

1. Композиция

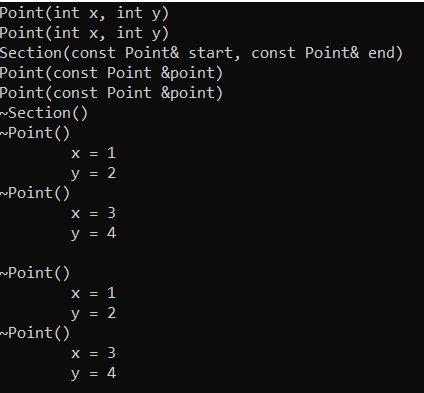


Рисунок 4. Отладочный вывод композиции

**Вывод**

В результате выполнения работы было освоено создание классов, создание дочерних классов, композиция, реализация методов внутри и вне описания класса, изменение атрибутов доступа через поля класса, а также удаление объектов класса.

Ссылка на репозиторий: https://github.com/repseD/Gulin\_Labs\_OOP

**Приложение 1**

**Main.cpp**

#include <iostream>

class Human {

public:

int age;

void eat() {

printf("eat()\n");

}

Human() {

printf("Human()\n");

age = 0;

name = "name";

isWorking = false;

}

Human(int age, std::string name, bool isWorking) {

printf("Human(int age, std::string name, bool isWorking)\n");

this->age = age;

this->name = name;

this->isWorking = isWorking;

}

Human(const Human& human) {

printf("Human(const Human& human)\n");

this->age = human.age;

this->isWorking = human.isWorking;

this->name = human.name;

}

~Human() {

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

printf("\tage = %d\n", age);

printf("\tname = %s\n", name.c\_str());

printf("\tisWorking = %d\n", isWorking);

}

bool isWorking\_();

private:

std::string name;

protected:

bool isWorking;

};

bool Human::isWorking\_() {

printf("Human::isWorking\_()\n");

return isWorking;

}

class Bob : public Human {

public:

Bob() {

isWorking = false; // имею доступ к этому полю

//name = "sda" // нет доступа, из-за private

}

};

int main() {

{

Human human;

Human human1(1, "name", true);

Human human2(human1);

human.age = 2;

//human.name

//human.isWorking не могу из-за модификатора доступа private и protected

human.isWorking\_();

human1.eat();

}

}

**Main2.cpp**

#include "iostream"

#include "math.h"

class Point2D {

protected:

int x;

int y;

public:

Point2D() {

printf("Point2D()\n");

x = 0;

y = 0;

}

Point2D(int x, int y) : x(x), y(y) {

printf("Point2D(int x, int y) : x(x), y(y)\n");

}

Point2D(const Point2D &point2D) {

printf("Point2D(const Point2D& point2D)\n");

x = point2D.x;

y = point2D.y;

}

virtual ~Point2D() {

printf("virtual ~Point2D()\n");

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

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

}

virtual int distance() {

printf("virtual int distance()\n");

return sqrt(x \* x + y \* y);

}

};

class Point3D : public Point2D {

protected:

int z;

public:

Point3D() : Point2D() {

printf("Point3D() : Point2D()\n");

z = 0;

}

Point3D(int x, int y, int z) : Point2D(x, y) {

printf("Point3D(int x, int y, int z) : Point2D(x, y)\n");

this->z = z;

}

Point3D(const Point3D &point3D) : Point2D(point3D) {

printf("Point3D(const Point3D & point3D) : Point2D(point3D)\n");

z = point3D.z;

}

virtual ~Point3D() {

printf("virtual ~Point3D()\n");

printf("\tz = %d\n", z);

}

virtual int distance() override {

printf("virtual int distance() override\n");

return sqrt(x \* x + y \* y + z \* z);

}

void moveToZero(){

x = 0;

y = 0;

z = 0;

}

};

int main() {

Point2D \*point2D = new Point2D(2,5);

Point3D \*point3D = new Point3D(1, 2, 4);

Point2D \*point2D1 = new Point3D(\*point3D);

printf("\n");

point2D->distance();

point3D->distance();

point3D->moveToZero();

point2D1->distance();

printf("\n");

delete point2D1;

delete point3D;

delete point2D;

}

**Main3.cpp**

#include "iostream"

class Point {

private:

int x;

int y;

public:

Point() {

printf("Point()\n");

x = 0;

y = 0;

}

Point(int x, int y) {

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

this->x = x;

this->y = y;

}

Point(const Point &point) {

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

x = point.x;

y = point.y;

}

~Point(){

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

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

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

}

};

class Section{

private:

Point start;

Point end;

public:

Section() : start(), end(){

printf("Section()\n");

}

Section(int x1, int y1, int x2, int y2) : start(x1, y1), end(x2, y2){

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

}

Section(const Point& start, const Point& end): start(start), end(end){

printf("Section(const Point& start, const Point& end)\n");

}

Section(const Section& section): start(section.start), end(section.end){

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

}

~Section(){

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

}

};

int main() {

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

Section\* section1 = new Section(\*section);

delete section1;

delete section;

}

**Main4.cpp**

#include "iostream"

class Point {

private:

int x;

int y;

public:

Point() {

printf("Point()\n");

x = 0;

y = 0;

}

Point(int x, int y) {

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

this->x = x;

this->y = y;

}

Point(const Point &point) {

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

x = point.x;

y = point.y;

}

~Point() {

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

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

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

}

};

class Section {

private:

Point \*start;

Point \*end;

public:

Section() {

printf("Section()\n");

start = new Point();

end = new Point();

}

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

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

start = new Point(x1, y1);

end = new Point(x2, y2);

}

Section(const Point &start, const Point &end) {

printf("Section(const Point& start, const Point& end)\n");

this->start = new Point(start);

this->end = new Point(end);

}

Section(const Section &section) {

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

start = new Point(\*section.start);

end = new Point(\*section.end);

}

~Section() {

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

delete start;

delete end;

}

};

int main() {

Point \*start = new Point(1, 2);

Point \*end = new Point(3, 4);

Section \*section = new Section(\*start, \*end);

delete section;

printf("\n");

delete start;

delete end;

}

**Main5.cpp**

#include "iostream"

class Point {

private:

int x;

int y;

public:

Point() {

printf("Point()\n");

x = 0;

y = 0;

}

Point(int x, int y) {

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

this->x = x;

this->y = y;

}

Point(const Point &point) {

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

x = point.x;

y = point.y;

}

~Point() {

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

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

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

}

};

class Section {

private:

Point start;

Point \*end;

public:

Section() {

printf("Section()\n");

end = new Point();

}

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

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

start = Point(x1, y1);

end = new Point(x2, y2);

}

Section(const Point &start, const Point &end) {

printf("Section(const Point& start, const Point& end)\n");

this->start = Point(start);

this->end = new Point(end);

}

Section(const Section &section) {

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

start = Point(section.start);

end = new Point(\*section.end);

}

~Section() {

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

delete end;

}

};

int main() {

Point start = Point(1, 2);

Point \*end = new Point(3, 4);

Section \*section = new Section(start, \*end);

delete section;

printf("\n");

delete end; }