Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра ІУС

Дисципліна: «Технології об’єктно орієнтованого програмування»

**ПРАКТИЧНА РОБОТА № 2**

**«ОРГАНІЗАЦІЯ ВЗАЄМОДІЇ МІЖ КЛАСАМИ»**

|  |  |
| --- | --- |
|  | Прийняла:  Білова Т. Г.  з оцінкою «\_\_»  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2020р. |

|  |  |
| --- | --- |
|  | Виконав:  Ст. гр. ІТУ-19-2  Куренков Б.М. |

**МЕТА РОБОТИ**

Засвоєння поняття статичних та константних членів та методів класу, розглянути особливості реалізації дружніх функцій та класів, набуття навичок організації та обробки масивів об’єктів, створення та використання покажчиків на об’єкт, передачі та повернення об’єктів з функцій.

**ПОСТАНОВКА ЗАДАЧІ**

**Завдання з максимальною оцінкою 10 балів**

Варіант 9

1. Створити клас Point (точка на площині).

Закриті поля класу: координати x, y

Методи класу:

– конструктор за замовченням;

– конструктор з параметрами (реалізується за допомогою списку

ініціалізації);

– модифікатор, що запитує координати точки з екрану;

– селектори, що повертають значення закритих полів;

– визначення відстані від точки до начала координат;

– визначення відстані між двома точками.

2. Створити клас Quadrangle (чотирикутник)

Закриті поля класу:

– масив покажчиків (чотири елемента) на об’єкти класу Point - вершини

чотирикутника;

– статична змінна, яка підраховує кількість об’єктів класу.

Методи класу:

– конструктор за замовченням (динамічно виділяє пам&#39;ять на чотири

елементи класу Point, використовуються конструктори за замовчуванням

класу Point);

– конструктор з параметрами (приймає чотири об’єкти класу Point);

– конструктор копіювання та деструктор;

– перевантажений оператор присвоювання;

– конструктор переміщення;

– селектори, що повертають довжину сторін чотирикутника;

– метод виведення інформації про чотирикутник;

– метод, що визначає тип чотирикутника – прямокутник, квадрат, ромб,

паралелограм;

– методи обчислення периметру чотирикутника.

3. Створити дружню функцію, яка приймає в якості параметрів об’єкти

класів Point та Quadrangle, та виводить інформацію чи належить точка

чотирикутнику (логічна).

4. В головній програмі:

– продемонструвати роботу всіх методів класу;

– створити масив об’єктів класу Quadrangle та відсортувати його за

зростанням периметру чотирикутників.

**ХІД РОБОТИ**

#pragma once

#include <iostream>

#include <cmath>

#include <iomanip>

#include <utility>

enum {rectangle, square, parallelogram, rhombus};

class Quadrangle;

class Point {

float x, y;

public:

Point();

Point(float, float);

void setCord(float, float);

float\* getCord() const;

float distance\_to\_origin() const;

float distance\_btwn\_points(Point&) const;

friend bool isPointInQuadrangle(const Point& point, const Quadrangle& quad);

friend class Quadrangle;

};

class Quadrangle {

static int count\_of\_Quadrangle;

int q\_num;

Point\*\* points = new Point\*[4];

public:

Quadrangle();

Quadrangle(Point&, Point&, Point&, Point&);

Quadrangle(const Quadrangle&);

Quadrangle(Quadrangle&&);

Quadrangle& operator=(const Quadrangle&);

Quadrangle& operator=(Quadrangle&&);

float\* getSides();

void show();

float getPerimeter();

int getType();

int getQ\_id();

friend bool isPointInQuadrangle(const Point& point, const Quadrangle& quad);

~Quadrangle();

};

int Quadrangle::count\_of\_Quadrangle = 0;

//------------------------------------------------------------------------------------

Point::Point() : x(0), y(0) {

;

}

Point::Point(float x, float y) {

this->x = x; this->y = y;

}

void Point::setCord(float x, float y) {

this->x = x; this->y = y;

}

float\* Point::getCord() const {

float\* tmp = new float[2];

tmp[0] = x; tmp[1] = y;

return tmp;

}

float Point::distance\_to\_origin() const {

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

}

float Point::distance\_btwn\_points(Point& point) const {

if (&point == NULL) return NULL;

return abs(sqrt(pow((x - point.x), 2) + pow((y - point.y), 2)));

}

//------------------------------------------------------------------------------------

Quadrangle::Quadrangle(){

q\_num = ++count\_of\_Quadrangle;

points = NULL;

}

Quadrangle::Quadrangle(Point& p1, Point& p2, Point& p3, Point& p4) {

q\_num = ++count\_of\_Quadrangle;

points[0] = &p1, points[1] = &p2, points[2] = &p3, points[3] = &p4;

}

Quadrangle::Quadrangle(const Quadrangle& quad) { // конструктор копирования

q\_num = ++count\_of\_Quadrangle;

points[0] = quad.points[0], points[1] = quad.points[1], points[2] = quad.points[2], points[3] = quad.points[3];

}

Quadrangle::Quadrangle(Quadrangle&& quad) { // конструктор перемещения

q\_num = ++count\_of\_Quadrangle;

points = quad.points;

quad.points = NULL;

}

Quadrangle& Quadrangle::operator=(const Quadrangle& quad) { // оператор копирования

if (&quad == this)

return \*this;

if (quad.points != NULL)

points[0] = quad.points[0], points[1] = quad.points[1], points[2] = quad.points[2], points[3] = quad.points[3];

else

points = NULL;

return \*this;

}

Quadrangle& Quadrangle::operator=(Quadrangle&& quad) { // оператор перемещения

if (&quad == this)

return \*this;

points = quad.points;

quad.points = NULL;

return \*this;

}

float\* Quadrangle::getSides() {

float\* tmp = new float[4];

if (points == NULL) { tmp[0] = tmp[1] = tmp[2] = tmp[3] = 0; return tmp; }

tmp[0] = points[0]->distance\_btwn\_points(\*points[1]);

tmp[1] = points[1]->distance\_btwn\_points(\*points[2]);

tmp[2] = points[2]->distance\_btwn\_points(\*points[3]);

tmp[3] = points[3]->distance\_btwn\_points(\*points[0]);

return tmp;

}

void Quadrangle::show() {

std::cout << "Quadrangle num is: " << q\_num << "\n";

std::cout << "Static value is: " << count\_of\_Quadrangle << "\n";

if (points == NULL)

{ std::cout << "Quadrangle is empty.\n\n"; }

else {

std::cout << \*(points[0]->getCord()) << " " << \*(points[0]->getCord() + 1) << std::endl;

std::cout << \*(points[1]->getCord()) << " " << \*(points[1]->getCord() + 1) << std::endl;

std::cout << \*(points[2]->getCord()) << " " << \*(points[2]->getCord() + 1) << std::endl;

std::cout << \*(points[3]->getCord()) << " " << \*(points[3]->getCord() + 1) << std::endl;

std::cout << "Distanse between points:";

float\* tmp = new float[4];

tmp = this->getSides();

std::cout << std::setw(9) << tmp[0] << std::setw(9) << tmp[1] << std::setw(9) << tmp[2] << std::setw(9) << tmp[3] <<"\n";

delete[] tmp;

std::cout << "Perimeter is: " << this->getPerimeter() << "\n\n";

}

}

float Quadrangle::getPerimeter() {

float\* tmp = new float[4];

tmp = this->getSides();

float perimeter = tmp[0] + tmp[1] + tmp[2] + tmp[3];

delete[] tmp;

return perimeter;

}

int Quadrangle::getType() {

if (points == NULL) { std::cout << "Quadrangle is empty!\n\n"; return -1; }

this->show();

/\*на паралельность\*/

if (abs((points[1]->x - points[0]->x) / (points[3]->x - points[2]->x)) == abs((points[1]->y - points[0]->y) / (points[3]->y - points[2]->y))

|| abs((points[1]->x - points[0]->x) / (points[3]->x - points[2]->x)) == abs((points[2]->y - points[1]->y) / (points[0]->y - points[3]->y))

|| abs((points[2]->x - points[1]->x) / (points[0]->x - points[3]->x)) == abs((points[2]->y - points[1]->y) / (points[0]->y - points[3]->y))

|| abs((points[1]->x - points[0]->x) / (points[3]->x - points[2]->x)) == abs((points[2]->y - points[1]->y) / (points[0]->y - points[3]->y))) {

float\* sides = this->getSides();

/\*прямой угол\*/

if (0 == (((points[1]->x - points[0]->x) \* (points[2]->x - points[1]->x) + (points[1]->y - points[0]->y) \* (points[2]->y - points[1]->y)))) {

std::cout << "rectangle or square\n";

/\*a=b=c=d\*/

if (sides[0] == sides[1] && sides[0] == sides[2] && sides[0] == sides[3] && sides[1] == sides[2] && sides[1] == sides[3] && sides[2] == sides[3]) {

std::cout << "square\n\n\n"; return square;

}

/\*!(a=b=c=d)\*/

else {

std::cout << "rectangle\n\n\n"; return rectangle;

}

}

/\*непрямой угол\*/

else {

std::cout << "parallelogram or rhombus\n";

/\*a=b=c=d\*/

if (sides[0] == sides[1] && sides[0] == sides[2] && sides[0] == sides[3] && sides[1] == sides[2] && sides[1] == sides[3] && sides[2] == sides[3]) {

std::cout << "rhobmus\n\n\n"; return rhombus;

}

/\*!(a=b=c=d)\*/

else {

std::cout << "parallelogram\n\n\n"; return parallelogram;

}

}

}

/\*непаралелен\*/

else { std::cout << "simple quadrangle\n\n\n"; return 5; }

return 0;

}

int Quadrangle::getQ\_id() {

return q\_num;

}

Quadrangle::~Quadrangle() {

delete[] points;

--count\_of\_Quadrangle;

}

bool isPointInQuadrangle(const Point& point, const Quadrangle& quad) {

if (point.x == quad.points[0]->x && point.y == quad.points[0]->y || point.x == quad.points[1]->x && point.y == quad.points[1]->y

|| point.x == quad.points[2]->x && point.y == quad.points[2]->y || point.x == quad.points[3]->x && point.y == quad.points[3]->y) return 1;

else return 0;

}

----------------------------------------------------------------------------------------------

#include "Quadrangle\_and\_Point.h"

int main() {

setlocale(LC\_ALL, "Russian");

/\*произвольный четырехугольник\*/Point p4[4] = { {10, -15}, {31, 37}, {52, 27}, {35, -26} };

/\*прямоугольник\*/Point p0[4] = { {1, -1}, {3, 3}, {5, 2}, {3, -2} };

/\*квадрат\*/Point p1[4] = { {-4, 4}, {4, 4}, {4, -4}, {-4, -4} };

/\*паралелограм\*/Point p2[4] = { {0, 0}, {2, 3}, {2, 0}, {0, -3} };

/\*ромб\*/Point p3[4] = { {-2, 0}, {0, 3}, {2, 0}, {0, -3} };

Quadrangle\* quad4 = new Quadrangle(p4[0], p4[1], p4[2], p4[3]),

\* quad0 = new Quadrangle(p0[0], p0[1], p0[2], p0[3]),

\* quad1 = new Quadrangle(p1[0], p1[1], p1[2], p1[3]),

\* quad2 = new Quadrangle(p2[0], p2[1], p2[2], p2[3]),

\* quad3 = new Quadrangle(p3[0], p3[1], p3[2], p3[3]),

\* quad\_empty = new Quadrangle;

std::cout << "Было создано 5 четырехугольников, выведем информацию о первом из них.\n\n";

quad4->show();

std::cout << "Как видим, на самом деле их 6. Все из-за созданного, но не упомянутого программистом ещё одного, пустого четырехугольника.\n\n";

quad\_empty->show();

std::cout << "Пытаясь вывести информацию о нём, поулчаем вот это. Действительно, он ведь пуст.\n\n";

std::cout << "Переместим же содержимое первого четырехугольника в пустой шестой с помощью конструктора перемещения.\n";

\*quad\_empty = std::move(\*quad4);

std::cout << "Перемещение выполнено. Отныне шестой владеет данными первого, а первый - пуст.\n\n";

quad4->show();

quad\_empty->show();

std::cout << "Вернув все на свои места, продолжим работу.\n\n";

\*quad4 = std::move(\*quad\_empty);

float a, b;

std::cout << "Проверим пренадлежит ли точка {a; b} шестому четырехугольнику. Введите а и b\n a = ";

std::cin >> a;

std::cout << " b = ";

std::cin >> b;

std::cout << "\n";

true == isPointInQuadrangle({ a, b }, \*quad4) ? std::cout << "Да, пренадлежит.\n\n" : std::cout << "Нет, не пренадлежит.\n\n";

float c, d;

std::cout << "Создадим точку {c; d} и узнаем расстояния:\n1) от неё до начала координат.\n2) от неё до точки {a; b}\n\nВведите c и d\n c = ";

std::cin >> c;

std::cout << " d = ";

std::cin >> d;

std::cout << "\n";

Point ab{ a,b }, cd{ c, d };

std::cout << "1) " << cd.distance\_to\_origin();

std::cout << "\n2) " << cd.distance\_btwn\_points(ab);

std::cout << "\n\n";

std::cout << "Создадим массив со всеми четырехугольниками, затем отсортируем их по возрастанию периметра.\nВыводя на экран результат, будем удалять четырехугольники\n";

Quadrangle\* quads[6] = { quad4, quad0, quad1, quad2, quad3, quad\_empty };

std::cout << "\n\nДо сортировки:\n\n";

for (int i = 0; i < 6; ++i)

quads[i]->show();

Quadrangle\* tmp;

for (int i = 0; i < 6; ++i) {

for (int j = i + 1; j < 6; ++j)

if (quads[i]->getPerimeter() < quads[j]->getPerimeter())

{

tmp = quads[j];

quads[j] = quads[i];

quads[i] = tmp;

}

}

std::cout << "\n\nПосле сортировки:\n\n";

for (int i = 0; i < 6; ++i) {

quads[i]->show();

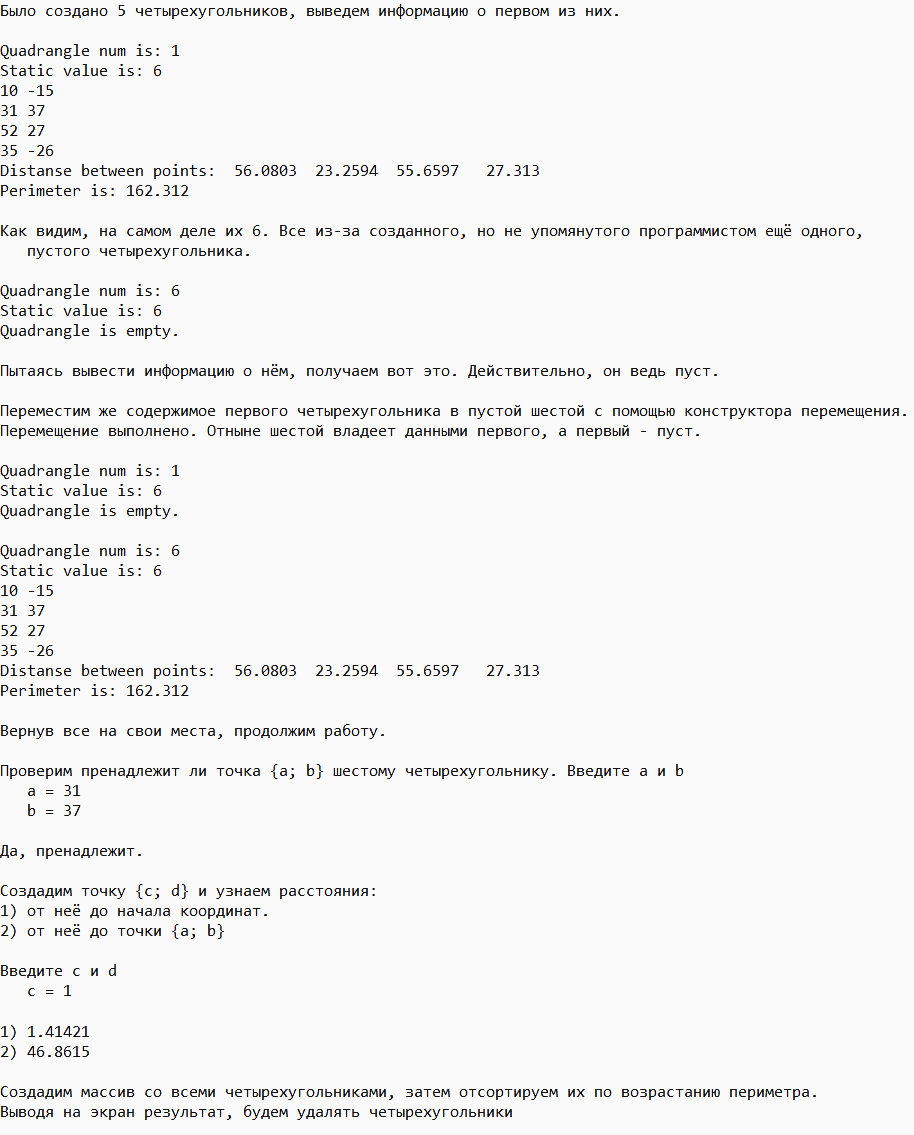
delete quads[i];

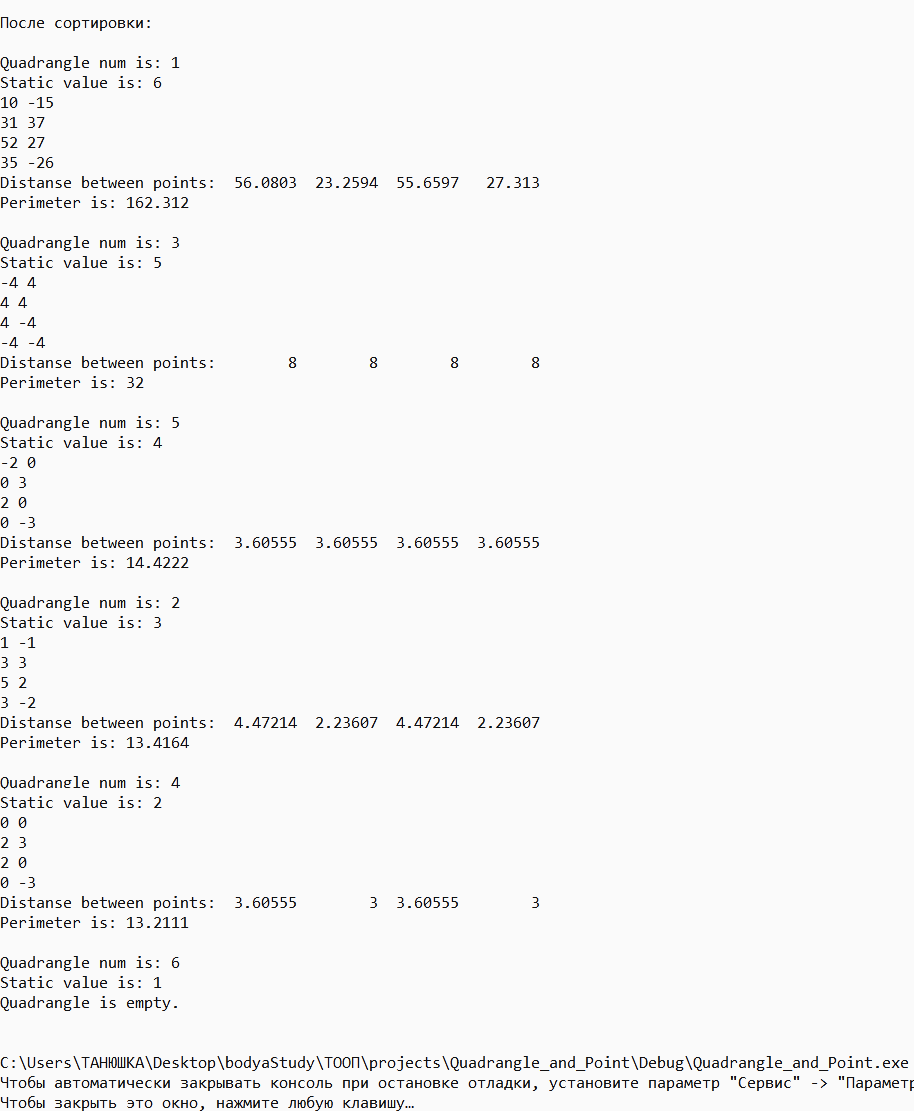
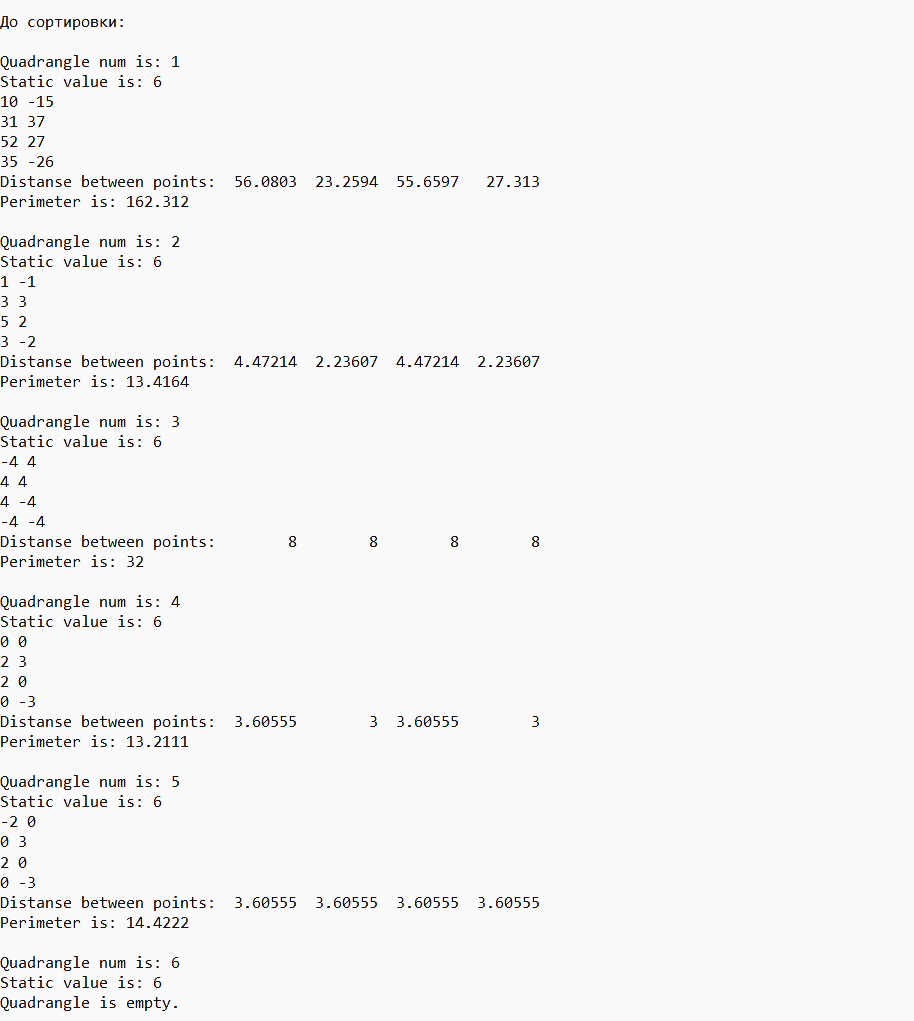
}

return 0;

}

**ВИКОНАННЯ ЕКСПЕРИМЕНТІВ ПО НАЛАГОДЖЕННЮ ПРОГРАМИ**

****

****

Висновки.

Протягом лабораторної роботи мною було засвоєно поняття статичних та константних членів та методів класу, були розглянуті особливості реалізації дружніх функцій та класів, були набуті навички організації та обробки масивів об’єктів, створення та використання покажчиків на об’єкт, передачі та повернення об’єктів з функцій.