Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

III семестр

Задание 3: «Наследование, полиморфизм»

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №9 |
| Студент: | Игитова Александра Андреевна |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 07.12.2019 |

Москва, 2019

1. **Задание**

Написать программу с базовым классом Figure и производными классами квадрата, прямоугольника с треугольником, которые наследуются от класса Figure. Должна быть возможность вычисления центра фигуры и ее площади при помощи переопределения виртуальных функций родительского класса.

1. **Адрес репозитория на GitHub**

<https://github.com/SandraIgitova/oop_exercise_04>

1. **Код программы на С++**

main4.cpp

#include <cstdio>

#include <stdlib.h>

#include <cmath>

#include <vector>

#include <iostream>

#include <cassert>

#include <tuple>

#include "rectangle.h"

#include "square.h"

#include "triangle.h"

#include "templates.h"

void FigureMenuOutput(uint32\_t& x) {

std::cout << "\n\

1. Ввести треугольник\n\

2. Ввести прямоугольник\n\

3. Ввести квадрат\n\n" << std::endl;

std::cin >> x;

};

void ArgTypeOutput(uint32\_t& x) {

std::cout << "\

1. Структура\n\

2. Tuple\n\n" << std::endl;

std::cin >> x;

};

template <typename T>

using vertex\_t = std::pair<T, T>;

void TriangleInput(vertex\_t<TYPE> &A, vertex\_t<TYPE> &B, vertex\_t<TYPE> &C) {

std::cout << "Введите 6 чисел координат сторон треугольника, чередуя X и Y: \n";

std::cin >> A.first >> A.second >> B.first >> B.second >> C.first >> C.second;

};

void RectangleInput(vertex\_t<TYPE> &A, vertex\_t<TYPE> &B, vertex\_t<TYPE> &C, vertex\_t<TYPE> &D) {

std::cout << "Введите 8 чисел координат сторон четырехугольника, чередуя X и Y: \n";

std::cin >> A.first >> A.second >> B.first >> B.second >> C.first >> C.second >> D.first >> D.second;

};

void SquareInput(vertex\_t<TYPE> &A, vertex\_t<TYPE> &B, vertex\_t<TYPE> &C, vertex\_t<TYPE> &D) {

std::cout << "Введите 8 чисел координат сторон четырехугольника, чередуя X и Y: \n";

std::cin >> A.first >> A.second >> B.first >> B.second >> C.first >> C.second >> D.first >> D.second;

};

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

int main()

{

using vertex\_t = std::pair<TYPE, TYPE>;

vertex\_t a, b, c, d;

uint32\_t x = 0;

while (std::cin)

{

FigureMenuOutput(x);

if (x == 1) {

TriangleInput(a, b, c);

ArgTypeOutput(x);

if (x == 1) {

Triangle<TYPE> Triangle(a, b, c);

// передаем в аргументе объект треугольник

Print(Triangle);

std::cout << "Площадь фигуры: \n";

std::cout << "\t" << Area(Triangle) << std::endl;

std::cout << "Центр фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << Center(Triangle).first << "\t" << Center(Triangle).second << std::endl;

}

if (x == 2) {

// передаем в аргументе tuple из трех pair

std::tuple<vertex\_t, vertex\_t, vertex\_t> argsT{ a,b,c };

Print(argsT);

std::cout << "Площадь фигуры: \n";

std::cout << "\t" << Area(argsT) << std::endl;

std::cout << "Центр фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << Center(argsT).first << "\t" << Center(argsT).second << std::endl;

}

continue;

}

if (x == 2) {

RectangleInput(a, b, c, d);

if (!IsRectangle(a, b, c, d)) {

std::cout << "Данная фигура не является прямоугольником." << std::endl;

continue;

}

std::cout << "Данная фигура является прямоугольником." << std::endl;

ArgTypeOutput(x);

if (x == 1) {

Rectangle<TYPE> Rectangle(a, b, c, d);

// передаем в аргументе объект четырехугольник

Print(Rectangle);

std::cout << "Площадь фигуры: \n";

std::cout << "\t" << Area(Rectangle) << std::endl;

std::cout << "Центр фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << Center(Rectangle).first << "\t" << Center(Rectangle).second << std::endl << std::endl;

}

if (x == 2) {

// передаем в аргументе tuple из четырех pair

std::tuple<vertex\_t, vertex\_t, vertex\_t, vertex\_t> argsT{ a,b,c,d };

Print(argsT);

std::cout << "Площадь фигуры: \n";

std::cout << "\t" << Area(argsT) << std::endl;

std::cout << "Центр фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << Center(argsT).first << "\t" << Center(argsT).second << std::endl;

}

}

if (x == 3) {

SquareInput(a, b, c, d);

if (!IsSquare(a, b, c, d)) {

std::cout << "Данная фигура не является квадратом." << std::endl;

continue;

}

std::cout << "Данная фигура является квадратом." << std::endl;

ArgTypeOutput(x);

if (x == 1) {

Rectangle<TYPE> Rectangle(a, b, c, d);

// передаем в аргументе объект четырехугольник

Print(Rectangle);

std::cout << "Площадь фигуры: \n";

std::cout << "\t" << Area(Rectangle) << std::endl;

std::cout << "Центр фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << Center(Rectangle).first << "\t" << Center(Rectangle).second << std::endl << std::endl;

}

if (x == 2) {

// передаем в аргументе tuple из четырех pair

std::tuple<vertex\_t, vertex\_t, vertex\_t, vertex\_t> argsT{ a,b,c,d };

Print(argsT);

std::cout << "Площадь фигуры: \n";

std::cout << "\t" << Area(argsT) << std::endl;

std::cout << "Центр фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << Center(argsT).first << "\t" << Center(argsT).second << std::endl;

}

}

}

return 0;

}

figure\_calculations.h

#include <stdlib.h>

#include <cmath>

#include <vector>

#include <iostream>

#include <cassert>

#include <tuple>

#define TYPE double // тип координат

template <typename T>

using vertex\_t = std::pair<T, T>; // вершина std:pair<TYPE,TYPE>

// сортировка вершин

template <typename T>

void Sort(int n, vertex\_t<T> \*arr) {

for (int i = 1; i < n; i++) {

for (int j = i; j > 0 && arr[j - 1].first > arr[j].first; j--) {

vertex\_t<T> tmp = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = tmp;

}

}

}

//------------------- математика------------------------------

template <typename T>

bool IsRectangle(vertex\_t<T> A, vertex\_t<T> B, vertex\_t<T> C, vertex\_t<T> D) {

vertex\_t<T> mas[4] = { A, B, C, D };

Sort(4, mas); // для введения точек в произвольном порядке

if (mas[0].first == mas[1].first) {

if (mas[0].second > mas[1].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[0]; mas[0] = hlp; }

if (mas[2].second < mas[3].second) { vertex\_t<T> hlp = mas[2]; mas[2] = mas[3]; mas[3] = hlp; }

}

else if (mas[1].second < mas[3].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[3]; mas[3] = hlp; }

vertex\_t<T> vector1, vector2, vector3, vector4;

vector1.first = mas[1].first - mas[0].first; vector1.second = mas[1].second - mas[0].second;

vector2.first = mas[2].first - mas[1].first; vector2.second = mas[2].second - mas[1].second;

vector3.first = mas[3].first - mas[2].first; vector3.second = mas[3].second - mas[2].second;

vector4.first = mas[0].first - mas[3].first; vector4.second = mas[0].second - mas[3].second;

// проверяем два угла скалярными произведениями

if (((vector1.first \* vector2.first + vector1.second \* vector2.second) == 0) && ((vector3.first \* vector2.first + vector3.second \* vector2.second) == 0) && ((vector4.first \* vector3.first + vector4.second \* vector3.second) == 0)) {

return true;

}

else { return false; }

}

template <typename T>

bool IsSquare(vertex\_t<T> A, vertex\_t<T> B, vertex\_t<T> C, vertex\_t<T> D) {

vertex\_t<T> mas[4] = { A, B, C, D };

Sort(4, mas); // для введения точек в произвольном порядке

if (mas[0].first == mas[1].first) {

if (mas[0].second > mas[1].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[0]; mas[0] = hlp; }

if (mas[2].second < mas[3].second) { vertex\_t<T> hlp = mas[2]; mas[2] = mas[3]; mas[3] = hlp; }

}

else if (mas[1].second < mas[3].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[3]; mas[3] = hlp; }

double d1 = sqrt(pow(mas[1].first - mas[0].first, 2) + pow(mas[1].second - mas[0].second, 2));

double d2 = sqrt(pow(mas[2].first - mas[1].first, 2) + pow(mas[1].second - mas[2].second, 2));

vertex\_t<T> vector1, vector2, vector3, vector4;

vector1.first = mas[1].first - mas[0].first; vector1.second = mas[1].second - mas[0].second;

vector2.first = mas[2].first - mas[1].first; vector2.second = mas[2].second - mas[1].second;

vector3.first = mas[3].first - mas[2].first; vector3.second = mas[3].second - mas[2].second;

vector4.first = mas[0].first - mas[3].first; vector4.second = mas[0].second - mas[3].second;

// проверяем два угла скалярными произведениями и равенство сторон

if (((vector1.first \* vector2.first + vector1.second \* vector2.second) == 0) && ((vector3.first \* vector2.first + vector3.second \* vector2.second) == 0) && ((vector4.first \* vector3.first + vector4.second \* vector3.second) == 0) && (d1 == d2)) {

return true;

}

else { return false; }

}

// сортировка вершин

/\*template <typename T>

void Sort(int n, vertex\_t<T> \*arr) {

for (int i = 1; i < n; i++) {

for (int j = i; j > 0 && arr[j - 1].first > arr[j].first; j--) {

vertex\_t<T> tmp = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = tmp;

}

}

}\*/

// площадь прямоугольника

template <typename T>

double RectangleArea(vertex\_t<T> a, vertex\_t<T> b, vertex\_t<T> c, vertex\_t<T> d)

{

vertex\_t<T> mas[4] = { a, b, c, d };

Sort(4, mas);

// x- first, y - second

if (mas[0].first == mas[1].first) {

if (mas[0].second > mas[1].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[0]; mas[0] = hlp; }

if (mas[2].second < mas[3].second) { vertex\_t<T> hlp = mas[2]; mas[2] = mas[3]; mas[3] = hlp; }

}

else if (mas[1].second < mas[3].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[3]; mas[3] = hlp; }

double d1 = sqrt(pow(mas[1].first - mas[0].first, 2) + pow(mas[1].second - mas[0].second, 2));

double d2 = sqrt(pow(mas[2].first - mas[1].first, 2) + pow(mas[1].second - mas[2].second, 2));

return d1 \* d2;

}

// площадь треугольника по формуле Герона

template <typename T>

double TriangleArea(vertex\_t<T> a, vertex\_t<T> b, vertex\_t<T> c)

{

vertex\_t<T> mas[3] = { a, b, c };

Sort(3, mas);

// x- first, y - second

if (mas[0].first == mas[1].first)

if (mas[0].second > mas[1].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[0]; mas[0] = hlp; }

if (mas[1].first == mas[2].first)

if (mas[2].second > mas[1].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[2]; mas[2] = hlp; }

else if (mas[1].second < mas[2].second) { vertex\_t<T> hlp = mas[1]; mas[1] = mas[2]; mas[2] = hlp; }

double a1 = sqrt(pow(mas[1].first - mas[0].first, 2) + pow(mas[1].second - mas[0].second, 2));

double b1 = sqrt(pow(mas[1].first - mas[2].first, 2) + pow(mas[2].second - mas[1].second, 2));

double c1 = sqrt(pow(mas[2].first - mas[0].first, 2) + pow(mas[0].second - mas[2].second, 2));

double p = (a1 + b1 + c1) / 2;

return sqrt(p\*(p - a1)\*(p - b1)\*(p - c1));

}

// центр прямоугольника

template <typename T>

vertex\_t<T> RectangleCenter(vertex\_t<T> a, vertex\_t<T> b, vertex\_t<T> c, vertex\_t<T> d) {

vertex\_t<T> center;

// x- first, y - second

center.first = (a.first + b.first + c.first + d.first) / 4;

center.second = (a.second + b.second + c.second + d.second) / 4;

return center;

}

// центр треугольника

template <typename T>

vertex\_t<T> TriangleCenter(vertex\_t<T> a, vertex\_t<T> b, vertex\_t<T> c) {

vertex\_t<T> center;

// x- first, y - second

center.first = (a.first + b.first + c.first) / 3;

center.second = (a.second + b.second + c.second) / 3;

return center;

}

rectangle.h

#pragma once

#include <iostream>

#include <tuple>

#include "figure\_calculations.h"

template <class T>

class Rectangle{// Прямоугольник

using vertex\_t = std::pair<T, T>;

public:

vertex\_t a, b, c, d;

Rectangle(vertex\_t A, vertex\_t B, vertex\_t C, vertex\_t D) {

a.first = A.first; a.second = A.second;

b.first = B.first; b.second = B.second;

c.first = C.first; c.second = C.second;

d.first = D.first; d.second = D.second;

};

double Area() {

return RectangleArea(a, b, c, d);

};

vertex\_t Center() {

return RectangleCenter(a, b, c, d);

};

void Print() {

std::cout << "Координаты вершин фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << a.first << "\t" << a.second << std::endl;

std::cout << "\t" << b.first << "\t" << b.second << std::endl;

std::cout << "\t" << c.first << "\t" << c.second << std::endl;

std::cout << "\t" << d.first << "\t" << d.second << std::endl;

};

};

square.h

#pragma once

#include <iostream>

#include <tuple>

#include "figure\_calculations.h"

template <class T>

class Square{// Квадрат

using vertex\_t = std::pair<T, T>;

public:

vertex\_t a, b, c, d;

Square(vertex\_t A, vertex\_t B, vertex\_t C, vertex\_t D) {

a.first = A.first; a.second = A.second;

b.first = B.first; b.second = B.second;

c.first = C.first; c.second = C.second;

d.first = D.first; d.second = D.second;

};

double Area() {

return RectangleArea(a, b, c, d);

};

vertex\_t Center() {

return RectangleCenter(a, b, c, d);

};

void Print() {

std::cout << "Координаты вершин фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << a.first << "\t" << a.second << std::endl;

std::cout << "\t" << b.first << "\t" << b.second << std::endl;

std::cout << "\t" << c.first << "\t" << c.second << std::endl;

std::cout << "\t" << d.first << "\t" << d.second << std::endl;

};

};

triangle.h

#pragma once

#include <iostream>

#include <tuple>

#include "figure\_calculations.h"

template <class T>

class Triangle{// Треугольник

using vertex\_t = std::pair<T, T>;

public:

vertex\_t a, b, c;

Triangle(vertex\_t A, vertex\_t B, vertex\_t C) {

a.first = A.first; a.second = A.second;

b.first = B.first; b.second = B.second;

c.first = C.first; c.second = C.second;

};

double Area() {

return TriangleArea(a, b, c);

};

vertex\_t Center() {

return TriangleCenter(a, b, c);

};

void Print() {

std::cout << "Координаты вершин фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << a.first << "\t" << a.second << std::endl;

std::cout << "\t" << b.first << "\t" << b.second << std::endl;

std::cout << "\t" << c.first << "\t" << c.second << std::endl;

};

};

templates.h

#pragma once

#include "figure\_calculations.h"

template <class T>

double Area(std::tuple<vertex\_t<T>, vertex\_t<T>, vertex\_t<T>> argsT) {

return TriangleArea(std::get<0>(argsT), std::get<1>(argsT), std::get<2>(argsT));

};

template <class T>

double Area(std::tuple<vertex\_t<T>, vertex\_t<T>, vertex\_t<T>, vertex\_t<T>> argsT) {

return RectangleArea(std::get<0>(argsT), std::get<1>(argsT), std::get<2>(argsT), std::get<3>(argsT));

};

// центр

template <class T>

vertex\_t<T> Center(std::tuple<vertex\_t<T>, vertex\_t<T>, vertex\_t<T>> argsT) {

return TriangleCenter(std::get<0>(argsT), std::get<1>(argsT), std::get<2>(argsT));

};

template <class T>

vertex\_t<T> Center(std::tuple<vertex\_t<T>, vertex\_t<T>, vertex\_t<T>, vertex\_t<T>> argsT) {

return RectangleCenter(std::get<0>(argsT), std::get<1>(argsT), std::get<2>(argsT), std::get<3>(argsT));

};

// перегрузка

template <class T>

double Area(Rectangle<T> Rectangle) {

return Rectangle.Area();

};

template <class T>

double Area(Triangle<T> Triangle) {

return Triangle.Area();

};

template <class T>

vertex\_t<T> Center(Rectangle<T> Rectangle) {

return Rectangle.Center();

};

template <class T>

vertex\_t<T> Center(Triangle<T> Triangle) {

return Triangle.Center();

};

// вывод

template <typename T>

void Print(Rectangle<T> Rectangle) {

Rectangle.Print();

};

template <typename T>

void Print(Square<T> Square) {

Square.Print();

};

template <typename T>

void Print(std::tuple<vertex\_t<T>, vertex\_t<T>, vertex\_t<T>, vertex\_t<T>> argsT) {

std::cout << "Координаты вершин фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << std::get<0>(argsT).first << "\t" << std::get<0>(argsT).second << std::endl;

std::cout << "\t" << std::get<1>(argsT).first << "\t" << std::get<1>(argsT).second << std::endl;

std::cout << "\t" << std::get<2>(argsT).first << "\t" << std::get<2>(argsT).second << std::endl;

std::cout << "\t" << std::get<3>(argsT).first << "\t" << std::get<3>(argsT).second << std::endl;

std::cout << "Площадь фигуры: \n";

std::cout << "\t" << Area(argsT) << std::endl;

std::cout << "Центр фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << Center(argsT).first << "\t" << Center(argsT).second << std::endl << std::endl;

};

template <typename T>

void Print(Triangle<T> Triangle) {

Triangle.Print();

};

template <typename T>

void Print(std::tuple<vertex\_t<T>, vertex\_t<T>, vertex\_t<T>> argsT) {

std::cout << "Координаты вершин фигуры: \n";

std::cout << "\t" << "X" << "\t" << "Y" << std::endl;

std::cout << "\t" << std::get<0>(argsT).first << "\t" << std::get<0>(argsT).second << std::endl;

std::cout << "\t" << std::get<1>(argsT).first << "\t" << std::get<1>(argsT).second << std::endl;

std::cout << "\t" << std::get<2>(argsT).first << "\t" << std::get<2>(argsT).second << std::endl;

};

1. **Вывод**

Программа производит проверки для корректности ввода координат фигур, но не умеет генерировать другие точки по нескольким заданным. Вычисляет площади и центры, делает проверки ввода координат и команд.