Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа**

**по курсу «ООП»**

**Тема:**

**Основы метапрограммирования.**

|  |  |
| --- | --- |
| Студент: | Марков А.Н. |
| Группа: | М80-208Б-18 |
| Преподаватель: | Журавлев А.А. |
| Вариант: | 17 |
| Оценка: |  |
| Дата: |  |

Москва

2019

**1. Код программы на языке C++:**

**main.cpp:**

#include <iostream>

#include <cstring>

#include <exception>

#include "rectangle.h"

#include "square.h"

#include "triangle.h"

#include "vector.h"

#include "templates.h"

template<typename T>

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

template<typename T>

void process() {

T object;

std::cout << "Input points: ";

try {

Read(std::cin, object);

}

catch (std::exception &e) {

std::cout << e.what() << std::endl;

return;

}

Print(std::cout, object);

std::cout << std::endl<< Area(object) << std::endl;

std::cout << Center(object) << std::endl;

}

int main() {

std::string command;

std::cout << "Figure / Tuple" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

while (std::cin >> command) {

std::string object\_type;

if (command == "Figure") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Triangle / Square / Rectangle" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cin >> object\_type;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

if (object\_type == "Triangle") {

process<Triangle<double>>();

}

else if (object\_type == "Square") {

process<Square<double>>();

}

else if (object\_type == "Rectangle") {

process<Rectangle<double>>();

}

else {

std::cout << "Invalid figure" << std::endl;

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Figure / Tuple" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

}

}

else if (command == "Tuple") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Triangle / Square / Rectangle" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cin >> object\_type;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

if (object\_type == "Triangle") {

process<std::tuple<vertex\_t<double>, vertex\_t<double>,

vertex\_t<double>>>();

}

else if (object\_type == "Square") {

process<std::tuple<vertex\_t<double>, vertex\_t<double>,

vertex\_t<double>, vertex\_t<double>>>();

}

else if (object\_type == "Rectangle") {

process<std::tuple<vertex\_t<double>, vertex\_t<double>,

vertex\_t<double>, vertex\_t<double>>>();

}

else {

std::cout << "Invalid figure" << std::endl;

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Figure / Tuple" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

}

}

else {

std::cout << "Invalid command" << std::endl;

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

}

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Figure / Tuple" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

return 0;

}

**templates.h:**

#ifndef D\_TEMPLATES\_H

#define D\_TEMPLATES\_H 1

#include <tuple>

#include <utility>

#include <type\_traits>

#include <exception>

#include "vector.h"

template<typename T>

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

template<typename T>

struct is\_vertex : std::false\_type {};

template<typename T>

struct is\_vertex<std::pair<T, T>> : std::true\_type {};

template<class T>

struct is\_figurelike\_tuple : std::false\_type {};

template<class Head, class... Tail>

struct is\_figurelike\_tuple<std::tuple<Head, Tail...>> :

std::conjunction<is\_vertex<Head>,

std::is\_same<Head, Tail>...> {};

template<typename T>

inline constexpr bool is\_figurelike\_tuple\_v = is\_figurelike\_tuple<T>::value;

template<typename T, typename = void>

struct has\_method\_area : std::false\_type {};

template<typename T>

struct has\_method\_area<T, decltype(std::declval<const T&>().Area())> :

std::true\_type {};

template<typename T>

inline constexpr bool has\_method\_area\_v = has\_method\_area<T>::value;

template<typename T>

std::enable\_if\_t<has\_method\_area\_v<T>, double>

Area(const T& object) {

return object.Area();

}

template<typename T, typename = void>

struct has\_method\_center : std::false\_type {};

template<typename T>

struct has\_method\_center<T, decltype(std::declval<const T&>().Center())> :

std::true\_type {};

template<typename T>

inline constexpr bool has\_method\_center\_v = has\_method\_center<T>::value;

template<typename T>

std::enable\_if\_t<has\_method\_center\_v<T>, vertex\_t<double>>

Center(const T& object) {

return object.Center();

}

template<typename T, typename = void>

struct has\_method\_print : std::false\_type {};

template<typename T>

struct has\_method\_print<T, decltype(std::declval<const T&>().Print())> :

std::true\_type {};

template<typename T>

inline constexpr bool has\_method\_print\_v = has\_method\_print<T>::value;

template<typename T>

std::enable\_if\_t<has\_method\_print\_v<T>, std::ostream &>

Print(std::ostream &os, const T& object) {

return Print(os, object);

}

template<typename T, typename = void>

struct has\_method\_read : std::false\_type {};

template<typename T>

struct has\_method\_read<T, decltype(std::declval<const T&>().Read())> :

std::true\_type {};

template<typename T>

inline constexpr bool has\_method\_read\_v = has\_method\_read<T>::value;

template<typename T>

std::enable\_if\_t<has\_method\_read\_v<T>, std::istream &>

Read(std::istream &is, T& object) {

return Read(is, object);

}

template<size\_t Id, typename T>

double compute\_area(const T &tuple) {

if constexpr(Id >= std::tuple\_size\_v<T>) {

return 0;

}

else {

const auto dx1 = std::get<Id - 0>(tuple).first - std::get<0>(tuple).first;

const auto dy1 = std::get<Id - 0>(tuple).second - std::get<0>(tuple).second;

const auto dx2 = std::get<Id - 1>(tuple).first - std::get<0>(tuple).first;

const auto dy2 = std::get<Id - 1>(tuple).second - std::get<0>(tuple).second;

const double local\_area = std::abs(dx1 \* dy2 - dy1 \* dx2) \* 0.5;

return local\_area + compute\_area<Id + 1>(tuple);

}

}

template<typename T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, double>

Area(const T& object) {

if constexpr (std::tuple\_size\_v<T> < 3){

return 0;

}

else{

return compute\_area<2>(object);

}

}

template<size\_t Id, typename T>

double recursive\_center\_x(const T &tuple) {

if constexpr (Id >= std::tuple\_size\_v<T>) {

return 0;

}

else {

return (std::get<Id>(tuple).first / std::tuple\_size\_v<T>) + recursive\_center\_x<Id + 1>(tuple);

}

}

template<size\_t Id, typename T>

double recursive\_center\_y(const T &tuple) {

if constexpr (Id >= std::tuple\_size\_v<T>) {

return 0;

}

else {

return (std::get<Id>(tuple).second / std::tuple\_size\_v<T>) + recursive\_center\_y<Id + 1>(tuple);

}

}

template<size\_t Id, typename T>

vertex\_t<double> compute\_center(const T &tuple) {

if constexpr (Id >= std::tuple\_size\_v<T>) {

return 0;

}

else {

return {recursive\_center\_x<Id>(tuple), recursive\_center\_y<Id>(tuple)} ;

}

}

template<typename T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, vertex\_t<double>>

Center(const T& object) {

return compute\_center<0>(object);

}

template<size\_t Id, typename T>

void recursive\_print(std::ostream &os, const T &tuple) {

if constexpr (Id >= std::tuple\_size\_v<T>) {

return;

}

else {

os << std::get<Id>(tuple) << " ";

recursive\_print<Id + 1>(os, tuple);

}

}

template<typename T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, void>

Print(std::ostream &os, const T& object) {

recursive\_print<0>(os, object);

}

template<typename T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, void>

Check\_triangle(T& object) {

double AB = Length(std::get<0>(object), std::get<1>(object)),

BC = Length(std::get<1>(object), std::get<2>(object)),

AC = Length(std::get<0>(object), std::get<2>(object));

if (AB >= BC + AC || BC >= AB + AC || AC >= AB + BC) {

throw std::logic\_error("Vertices must not be on the same line.");

}

}

template<typename T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, bool>

Check\_rectangle(T& object) {

Vector<decltype(std::get<0>(object).first)> AB = {std::get<0>(object), std::get<1>(object)},

BC = {std::get<1>(object), std::get<2>(object)},

CD = {std::get<2>(object), std::get<3>(object)},

DA = {std::get<3>(object), std::get<0>(object)};

if (!is\_parallel(DA, BC) || !is\_parallel(AB, CD)) {

throw std::logic\_error("Vertices must be entered clockwise or counterclockwise");

}

if (AB \* BC || BC \* CD || CD \* DA || DA \* AB) {

throw std::logic\_error("The sides should be perpendicular");

}

if (!Length(AB) || !Length(BC) || !Length(CD) || !Length(DA)) {

throw std::logic\_error("The sides must be greater than zero");

}

return true;

}

template<typename T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, void>

Check(T& object) {

if constexpr (std::tuple\_size\_v<T> == 3) {

Check\_triangle(object);

}

else if (std::tuple\_size\_v<T> == 4) {

Check\_rectangle(object);

}

}

template<size\_t Id, typename T>

void recursive\_read(std::istream &is, T &tuple) {

if constexpr (Id >= std::tuple\_size\_v<T>) {

return;

}

else {

is >> std::get<Id>(tuple);

recursive\_read<Id + 1>(is, tuple);

}

}

template<typename T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, void>

Read(std::istream &is, T& object) {

recursive\_read<0>(is, object);

Check(object);

}

#endif // D\_TEMPLATES\_H

**vertex.h:**

#ifndef VERTEX\_H

#define VERTEX\_H 1

template<typename T>

struct vertex {

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

};

template<typename T>

std::istream &operator>>(std::istream &is, std::pair<T, T> &v) {

is >> v.first >> v.second;

return is;

}

template<typename T>

std::ostream &operator<<(std::ostream &os, const std::pair<T,T> &v) {

os << "[" << v.first << ", " << v.second << "]";

return os;

}

#endif // VERTEX\_H

**vector.h:**

#ifndef VECTOR\_H

#define VECTOR\_H 1

#include <utility>

#include <cmath>

#include <iostream>

#include "vertex.h"

template<typename T>

struct Vector {

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

T p1, p2;

Vector(T x\_cord, T y\_cord) : p1{x\_cord}, p2{y\_cord} {};

Vector(vertex\_t &p1, vertex\_t &p2) : p1{p2.first - p1.first},

p2{p2.second - p1.second} {};

double operator\*(const Vector<T> &a) const {

return (p1 \* a.p1) + (p2 \* a.p2);

}

Vector<T> &operator=(const Vector<T> &a) {

p1 = a.p1;

p2 = a.p2;

return \*this;

}

};

template<typename T>

double Length(const Vector<T> &vector) {

return sqrt(vector.p1 \* vector.p1 + vector.p2 \* vector.p2);

}

template<typename T>

double Length(const std::pair<T, T> &A,

const std::pair<T, T> &B) {

return sqrt(pow((B.first - A.first), 2) +

pow((B.second - A.second), 2));

}

template<typename T>

bool is\_parallel(const Vector<T> &A, const Vector<T> &B) {

return (A.p1 \* B.p2) - (A.p2 \* B.p1) == 0;

}

#endif //VECTOR\_H

**rectangle.h:**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H 1

#include <utility>

#include <iostream>

#include "vector.h"

#include "vertex.h"

template<typename T>

struct Rectangle {

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

vertex\_t vertices[4];

};

template<typename T>

typename Rectangle<T>::vertex\_t Center(const Rectangle<T> &r);

template<typename T>

double Area(const Rectangle<T> &r);

template<typename T>

std::ostream &Print(std::ostream &os, const Rectangle<T> &r);

template<typename T>

std::istream &Read(std::istream &is, Rectangle<T> &r);

template<typename T>

std::istream &operator>>(std::istream &is, Rectangle<T> &r);

template<typename T>

std::ostream &operator<<(std::ostream &os, const Rectangle<T> &r);

template<typename T>

typename Rectangle<T>::vertex\_t Center(const Rectangle<T> &r) {

T x, y;

x = (r.vertices[0].first + r.vertices[1].first + r.vertices[2].first + r.vertices[3].first) / 4;

y = (r.vertices[0].second + r.vertices[1].second + r.vertices[2].second + r.vertices[3].second) / 4;

return std::make\_pair(x, y);

}

template<typename T>

double Area(const Rectangle<T> &r) {

double res = 0;

for (int i = 0; i <= 2; i++) {

res += (r.vertices[i].first \* r.vertices[i + 1].second -

r.vertices[i + 1].first \* r.vertices[i].second);

}

res += (r.vertices[2].first \* r.vertices[0].second -

r.vertices[0].first \* r.vertices[2].second);

res = 0.5 \* std::abs(res);

return res;

}

template<typename T>

std::ostream &Print(std::ostream &os, const Rectangle<T> &r) {

for (int i = 0; i < 4; i++) {

os << r.vertices[i];

if (i != 3) {

os << " ";

}

}

return os;

}

template<typename T>

std::istream &Read(std::istream &is, Rectangle<T> &r) {

for (int i = 0; i < 4; i++) {

is >> r.vertices[i].first >> r.vertices[i].second;

}

Vector<T> AB = {r.vertices[0], r.vertices[1]},

BC = {r.vertices[1], r.vertices[2]},

CD = {r.vertices[2], r.vertices[3]},

DA = {r.vertices[3], r.vertices[0]};

if (!is\_parallel(DA, BC)) {

std::swap(r.vertices[0], r.vertices[1]);

AB = {r.vertices[0], r.vertices[1]};

BC = {r.vertices[1], r.vertices[2]};

CD = {r.vertices[2], r.vertices[3]};

DA = {r.vertices[3], r.vertices[0]};

}

if (!is\_parallel(AB, CD)) {

std::swap(r.vertices[1], r.vertices[2]);

AB = {r.vertices[0], r.vertices[1]};

BC = {r.vertices[1], r.vertices[2]};

CD = {r.vertices[2], r.vertices[3]};

DA = {r.vertices[3], r.vertices[0]};

}

if (AB \* BC || BC \* CD || CD \* DA || DA \* AB) {

throw std::logic\_error("The sides of the Rectangle should be perpendicular");

}

if (!Length(AB) || !Length(BC) || !Length(CD) || !Length(DA)) {

throw std::logic\_error("The sides of the Rectangle must be greater than zero");

}

return is;

}

template<typename T>

std::istream &operator>>(std::istream &is, Rectangle<T> &r) {

return Read(is, r);

}

template<typename T>

std::ostream &operator<<(std::ostream &os, const Rectangle<T> &r) {

return Print(os, r);

}

#endif // RECTANGLE\_H

**square.h:**

#ifndef SQUARE\_H

#define SQUARE\_H 1

#include <utility>

#include <iostream>

#include "vector.h"

template<typename T>

struct Square {

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

vertex\_t vertices[4];

};

template<typename T>

typename Square<T>::vertex\_t Center(const Square<T> &s);

template<typename T>

double Area(const Square<T> &s);

template<typename T>

std::ostream &Print(std::ostream &os, const Square<T> &s);

template<typename T>

std::istream &Read(std::istream &is, Square<T> &s);

template<typename T>

std::istream &operator>>(std::istream &is, Square<T> &s);

template<typename T>

std::ostream &operator<<(std::ostream &os, const Square<T> &s);

template<typename T>

typename Square<T>::vertex\_t Center(const Square<T> &s) {

T x, y;

x = (s.vertices[0].first + s.vertices[1].first + s.vertices[2].first + s.vertices[3].first) / 4;

y = (s.vertices[0].second + s.vertices[1].second + s.vertices[2].second + s.vertices[3].second) / 4;

return std::make\_pair(x, y);

}

template<typename T>

double Area(const Square<T> &s) {

double res = 0;

for (int i = 0; i <= 2; i++) {

res += (s.vertices[i].first \* s.vertices[i + 1].second -

s.vertices[i + 1].first \* s.vertices[i].second);

}

res += (s.vertices[2].first \* s.vertices[0].second -

s.vertices[0].first \* s.vertices[2].second);

res = 0.5 \* std::abs(res);

return res;

}

template<typename T>

std::ostream &Print(std::ostream &os, const Square<T> &s) {

for (int i = 0; i < 4; i++) {

os << s.vertices[i];

if (i != 3) {

os << " ";

}

}

return os;

}

template<typename T>

std::istream &Read(std::istream &is, Square<T> &s) {

for (int i = 0; i < 4; i++) {

is >> s.vertices[i].first >> s.vertices[i].second;

}

Vector<T> AB = {s.vertices[0], s.vertices[1]},

BC = {s.vertices[1], s.vertices[2]},

CD = {s.vertices[2], s.vertices[3]},

DA = {s.vertices[3], s.vertices[0]};

if (!is\_parallel(DA, BC)) {

std::swap(s.vertices[0], s.vertices[1]);

AB = {s.vertices[0], s.vertices[1]};

BC = {s.vertices[1], s.vertices[2]};

CD = {s.vertices[2], s.vertices[3]};

DA = {s.vertices[3], s.vertices[0]};

}

if (!is\_parallel(AB, CD)) {

std::swap(s.vertices[1], s.vertices[2]);

AB = {s.vertices[0], s.vertices[1]};

BC = {s.vertices[1], s.vertices[2]};

CD = {s.vertices[2], s.vertices[3]};

DA = {s.vertices[3], s.vertices[0]};

}

if (AB \* BC || BC \* CD || CD \* DA || DA \* AB) {

throw std::logic\_error("The sides of the square should be perpendicular");

}

if (Length(AB) != Length(BC) || Length(BC) != Length(CD) || Length(CD) != Length(DA) || Length(DA) != Length(AB)) {

throw std::logic\_error("The sides of the square should be equal");

}

if (!Length(AB) || !Length(BC) || !Length(CD) || !Length(DA)) {

throw std::logic\_error("The sides of the square must be greater than zero");

}

return is;

}

template<typename T>

std::istream &operator>>(std::istream &is, Square<T> &s) {

return Read(is, s);

}

template<typename T>

std::ostream &operator<<(std::ostream &os, const Square<T> &s) {

return Print(os, s);

}

#endif // SQUARE\_H

**triangle.h:**

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include <utility>

#include <iostream>

#include "vector.h"

#include "vertex.h"

template<typename T>

struct Triangle {

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

vertex\_t vertices[3];

};

template<typename T>

typename Triangle<T>::vertex\_t Center(const Triangle<T> &t);

template<typename T>

double Area(const Triangle<T> &t);

template<typename T>

std::ostream &Print(std::ostream &os, const Triangle<T> &t);

template<typename T>

std::istream &Read(std::istream &is, Triangle<T> &t);

template<typename T>

std::istream &operator>>(std::istream &is, Triangle<T> &t);

template<typename T>

std::ostream &operator<<(std::ostream &os, const Triangle<T> &t);

template<typename T>

typename Triangle<T>::vertex\_t Center(const Triangle<T> &t) {

T x, y;

x = (t.vertices[0].first + t.vertices[1].first + t.vertices[2].first) / 3;

y = (t.vertices[0].second + t.vertices[1].second + t.vertices[2].second) / 3;

return std::make\_pair(x, y);

}

template<typename T>

double Area(const Triangle<T> &t) {

double res = 0;

for (int i = 0; i <= 1; i++) {

res += (t.vertices[i].first \* t.vertices[i + 1].second -

t.vertices[i + 1].first \* t.vertices[i].second);

}

res += (t.vertices[2].first \* t.vertices[0].second -

t.vertices[0].first \* t.vertices[2].second);

res = 0.5 \* std::abs(res);

return res;

}

template<typename T>

std::ostream &Print(std::ostream &os, const Triangle<T> &t) {

for (int i = 0; i < 3; i++) {

os << t.vertices[i];

if (i != 2) {

os << " ";

}

}

return os;

}

template<typename T>

std::istream &Read(std::istream &is, Triangle<T> &t) {

for (int i = 0; i < 3; i++) {

is >> t.vertices[i].first >> t.vertices[i].second;

}

double AB = Length(t.vertices[0], t.vertices[1]),

BC = Length(t.vertices[1], t.vertices[2]),

AC = Length(t.vertices[0], t.vertices[2]);

if (AB >= BC + AC || BC >= AB + AC || AC >= AB + BC) {

throw std::logic\_error("Vertices must not be on the same line.");

}

return is;

}

template<typename T>

std::istream &operator>>(std::istream &is, Triangle<T> &t) {

return Read(is, t);

}

template<typename T>

std::ostream &operator<<(std::ostream &os, const Triangle<T> &t) {

return Print(os, t);

}

#endif // TRIANGLE\_H

**2. Ссылка на репозиторий на GitHub.**

<https://github.com/Markov-A-N/oop_exercise_04.git>

**3. Набор testcases.**

**test\_00.test:**

Figure

Triangle

0 0 1 1 1 0

Figure

Square

0 0 1 0 1 1 0 1

Figure

Rectangle

0 0 2 0 2 1 0 1

**test\_01.test:**

Tuple

Triangle

0 0 1 1 2 2

Tuple

Square

-5 -5 -5 5 5 5 5 -5

Tuple

Rectangle

-5 -5 -5 10 5 10 5 -5

**4. Результаты выполнения тестов.**

**test\_00.result:**

Figure / Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Triangle / Square / Rectangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Triangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 1 1 1 0

[0, 0] [1, 1] [1, 0]

0.5

[0.666667, 0.333333]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure / Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Triangle / Square / Rectangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Square

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 1 0 1 1 0 1

[0, 0] [1, 0] [1, 1] [0, 1]

1

[0.5, 0.5]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure / Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Triangle / Square / Rectangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Rectangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 2 0 2 1 0 1

[0, 0] [2, 0] [2, 1] [0, 1]

2

[1, 0.5]

**test\_01.result:**

Figure / Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Triangle / Square / Rectangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Triangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 1 1 2 2

Vertices must not be on the same line.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure / Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Triangle / Square / Rectangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Square

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: -5 -5 -5 5 5 5 5 -5

[-5, -5] [-5, 5] [5, 5] [5, -5]

100

[0, 0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure / Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tuple

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Triangle / Square / Rectangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Rectangle

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: -5 -5 -5 10 5 10 5 -5

[-5, -5] [-5, 10] [5, 10] [5, -5]

150

[0, 2.5]

**5. Объяснение результатов работы программы.**

1) Шаблонная функция Center() возвращает вершину, первой координатой которой является деление суммы иксов всех точек данной фигуры на их количество, со второй координатой аналогично. Эта функция определена для классов фигур и tuple. Во втором случае центр вычисляется рекурсивно.

2) Функция Print() выводит координаты всех точек данной фигуры. Эта функция определена для классов фигур и tuple. Во втором случае печать выполняется рекурсивно.

3) Функция Area() вычисляет площадь данной фигуры по методу Гаусса.

**6. Вывод.**

Научился использовать шаблоны, где в качестве параметра используются скалярными данные, для работы с шаблонными клаcсами и кортежами. Узнал о применении шаблонов в метапрограммировании. Также я познакомился с полезными заголовочными файлами <tuple> и <type\_traits>.