**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 4**

Тема: Изучение основ работы с шаблонами (template) в С++

Студент: Минибаев Айдар

Группа: 80-201Б-18

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

Вариант 9: треугольник, квадрат, прямоугольник.

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения. Для хранения координат фигур необходимо использовать шаблон std::pair.

Создать набор шаблонов, создающих функции, реализующие:

1. Вычисление геометрического центра фигуры.

2. Вывод в стандартный поток вывода std::cout координат вершин фигуры.

3. Вычисление площади фигуры.

Параметром шаблона должен являться тип класса фигуры ( например Square<int>). Помимо самого класса фигуры, шаблонная функция должна уметь работать с tuple.

Создать программу, которая позволяет:

• Вводить из стандартного ввода фигуры, согласно варианту задания (как в виде класса, так и в виде std::tuple).

• Вызывать для нее шаблонные функции (1-3).

1. **Описание программы**

Программа состоит из 2 файлов:

1. main.cpp
2. figures.h - фигуры и их функции

Реализованы шаблонные функции для трех фигур, а именно: вычисление площади, вычисление центра координат, вывод координат.

1. **Набор testcases**

Пользователь вводит фигуру в формате “имя тип”, и как хранить фигуру: в кортеже или как объект класса. Например, “Tri int tuple”, после чего вводит координаты. Если координаты введены неверно, то выдается соответствующая ошибка. После задания фигуры выводится площадь, координаты вершин и координаты геометрического центра фигуры.

**test\_\_01.txt**

Tri int tuple

0 0 3 5 7 9

Squ double class

1.1 1.1 5.5 5.5

Rec int class

0 0 4 0 4 3 0 3

tuple\_work

Проверка проверки ввода команд, задания координат квадрата, функций для него. Проверка работы с заданными фигурами через кортеж.

**test\_\_02.txt**

Tri double tuple

0 0 5.6 7.8 9.1 4.3

bgeog

Squ double class

0 0 4.3 6.5

Проверка проверки координат прямоугольника, обработки нецелых чисел при задании координат фигуры с целочисленными координатами, вывод формул для фигуры.

**test\_\_03.txt**

Rec int tuple

0 0 4 0 6 4 -2 4 // задан параллепипед

Проверка неверно заданного прямоугольника

1. **Результаты выполнения тестов.**

**test\_01.txt**

Enter a word for action:

1) Create Triangle[Tri]

2) Create Square [Squ]

3) Create Rectangle [Rec]

4) And then enter word to select data type: [int] or [double]

5) Next choose where to write a figure: [tuple] or [class] object

6) Demonstration of a tuple with figures[tuple\_work]

7) Print Menu[Menu]

8) Exit[Ex]

Tri int tuple

Choose data type: int or double

choose the type of shape input: through the [class] or [tuple]

cords point A: 0 0

cords point B: 3 5

cords point C: 7 9

1 Figure: Triangle integer

Tuple. Cords of Triangle: <0, 0>, <3, 5>, <7, 9>

Area:4

Center:<10, 14>

Squ double class

Choose data type: int or double

choose the type of shape input: through the [class] or [tuple]

cords lower left point A: 1.1 1.1

cords top right point C: 5.5 5.5

2 Figure: Square double

Cords of Square: <1.1, 1.1>, <5.5, 1.1>, <5.5, 5.5>, <1.1, 5.5>

Figure area19.36

The coordinates of the center of the figure<3.3, 3.3>

Rec int class

Choose data type: int or double

choose the type of shape input: through the [class] or [tuple]

cords point A: 0 0

cords point B: 4 0

cords point C: 4 3

cords point D: 0 3

3 Figure: Rectangle integer

Cords of Rectangle: <0, 0>, <4, 0>, <4, 3>, <0, 3>

Figure area12

The coordinates of the center of the figure<2, 1>

tuple\_work

Example with tuple usage:

libc++abi.dylib: terminating with uncaught exception of type std::overflow\_error: The given coordinates of the triangle lie on one straight line

Tuple. Cords of Square: <1.1, 1.1>, <3.3, 1.1>, <3.3, 3.3>, <1.1, 3.3>

Area: 4.84

<<Center of the figure:<2.2, 2.2>

Area:

Process finished with exit code 6

**test\_02.txt**

Enter a word for action:

1) Create Triangle[Tri]

2) Create Square [Squ]

3) Create Rectangle [Rec]

4) And then enter word to select data type: [int] or [double]

5) Next choose where to write a figure: [tuple] or [class] object

6) Demonstration of a tuple with figures[tuple\_work]

7) Print Menu[Menu]

8) Exit[Ex]

Tri double tuple

Choose data type: int or double

choose the type of shape input: through the [class] or [tuple]

cords point A: 0 0

cords point B: 5.6 7.8

cords point C: 9.1 4.3

1 Figure: Triangle double

Tuple. Cords of Triangle: <0, 0>, <5.6, 7.8>, <9.1, 4.3>

Area:23.45

Center:<14.7, 12.1>

bgeog

You did not select an action or you were mistaken in entering the name of the action

Squ double class

Choose data type: int or double

choose the type of shape input: through the [class] or [tuple]

cords lower left point A: 0 0

cords top right point C: 4.3 6.5

libc++abi.dylib: terminating with uncaught exception of type std::overflow\_error: Entered the coordinates of the vertices do not belong to the square.

Process finished with exit code 6

**test\_03.txt**

Enter a word for action:

1) Create Triangle[Tri]

2) Create Square [Squ]

3) Create Rectangle [Rec]

4) And then enter word to select data type: [int] or [double]

5) Next choose where to write a figure: [tuple] or [class] object

6) Demonstration of a tuple with figures[tuple\_work]

7) Print Menu[Menu]

8) Exit[Ex]

Rec int tuple

Choose data type: int or double

choose the type of shape input: through the [class] or [tuple]

cords point A: 0 0

cords point B: 4 0

cords point C: 6 4

cords point D: -2 4

1 Figure: Rectangle integer

libc++abi.dylib: terminating with uncaught exception of type std::overflow\_error: The entered coordinates of the vertices do not belong to the rectangle.

Process finished with exit code 6

Все тесты успешно пройдены, программа выдает правильные ответы.

1. **Листинг программы**

**main.cpp**

/\*

Написать программу по вычислению центра и площади квадрата, прямоугольника или треугольника, которые

задает пользователь, и по выводу их координат. Использовать вместо наследования шаблоны.

Сделать функции, которые принимают на вход кортежи.

Минибаев Айдар

М8О-201Б-18

\*/

#include "figures.h"

template <class T>

void Triangle\_read\_cords(std::pair<T, T> &A, std::pair<T, T> &B, std::pair<T, T> &C){

std::cout << "cords point A: ";

std::cin >> A;

std::cout << "cords point B: ";

std::cin >> B;

std::cout << "cords point C: ";

std::cin >> C;

}

template <class T>

void Square\_read\_cords(std::pair<T, T> &A, std::pair<T, T> &C){

std::cout << "cords lower left point A: ";

std::cin >> A;

std::cout << "cords top right point C: ";

std::cin >> C;

}

template <class T>

void Rectangle\_read\_cords(std::pair<T, T> &A, std::pair<T, T> &B, std::pair<T, T> &C, std::pair<T, T> &D){

std::cout << "cords point A: ";

std::cin >> A;

std::cout << "cords point B: ";

std::cin >> B;

std::cout << "cords point C: ";

std::cin >> C;

std::cout << "cords point D: ";

std::cin >> D;

}

template <class T>

void actions(const T &figure) {

Coord\_print(figure);

std::cout << "Figure area" << calc\_area(figure) << std::endl;

std::cout << "The coordinates of the center of the figure"

<< calc\_center(figure) << std::endl;

}

int main(){

using int\_pair = std::pair<int, int>;

using double\_pair = std::pair<double, double>;

std::tuple<int\_pair, int\_pair, int\_pair> tuple\_intTri;

std::tuple<double\_pair, double\_pair, double\_pair> tuple\_doubleTri;

std::tuple<int\_pair, int\_pair> tuple\_intSqu;

std::tuple<double\_pair, double\_pair> tuple\_doubleSqu;

std::tuple<int\_pair, int\_pair, int\_pair, int\_pair> tuple\_intRec;

std::tuple<double\_pair, double\_pair, double\_pair, double\_pair > tuple\_doubleRec;

int id = 0;

std::string option = "Menu";

while(option != "Ex") {

if (option == "Tri") {

id++;

std::cout << "Choose data type: int or double" << std::endl;

std::cin >> option;

if (option == "int") {

std::cout << "choose the type of shape input: through the [class] or [tuple]" << std::endl;

std::cin >> option;

if (option == "class") {

std::pair<int, int> A, B, C;

Triangle\_read\_cords(A, B, C);

Triangle<int> tri\_int(A, B, C);

std::cout << id << " Figure: Triangle integer\n";

actions(tri\_int);

} else if (option == "tuple") {

std::pair<int, int> A, B, C;

Triangle\_read\_cords(A, B, C);

std::get<0>(tuple\_intTri) = A;

std::get<1>(tuple\_intTri) = B;

std::get<2>(tuple\_intTri) = C;

std::cout << id << " Figure: Triangle integer\n";

Coord\_print(tuple\_intTri);

std::cout << "Area:" << calc\_area(tuple\_intTri) << std::endl;

std::cout << "Center:" << calc\_center(tuple\_intTri) << std::endl;

}

} else if (option == "double") {

std::cout << "choose the type of shape input: through the [class] or [tuple]" << std::endl;

std::cin >> option;

if (option == "class") {

std::pair<double, double> A, B, C;

Triangle\_read\_cords(A, B, C);

Triangle<double> tri\_double(A, B, C);

std::cout << id << " Figure: Triangle double\n";

actions(tri\_double);

} else if (option == "tuple") {

std::pair<double, double> A, B, C;

Triangle\_read\_cords(A, B, C);

std::get<0>(tuple\_doubleTri) = A;

std::get<1>(tuple\_doubleTri) = B;

std::get<2>(tuple\_doubleTri) = C;

std::cout << id << " Figure: Triangle double\n";

Coord\_print(tuple\_doubleTri);

std::cout << "Area:" << calc\_area(tuple\_doubleTri) << std::endl;

std::cout << "Center:" << calc\_center(tuple\_doubleTri) << std::endl;

}

}

} else if (option == "Squ") {

id++;

std::cout << "Choose data type: int or double" << std::endl;

std::cin >> option;

if (option == "int") {

std::cout << "choose the type of shape input: through the [class] or [tuple]" << std::endl;

std::cin >> option;

if (option == "class") {

std::pair<int, int> A, C;

Square\_read\_cords(A, C);

Square<int> squ\_int(A, C);

std::cout << id << " Figure: Square integer\n";

actions(squ\_int);

} else if (option == "tuple") {

std::pair<int, int> A, C;

Square\_read\_cords(A, C);

std::get<0>(tuple\_intSqu) = A;

std::get<1>(tuple\_intSqu) = C;

std::cout << id << " Figure: Square integer\n";

Coord\_print(tuple\_intSqu);

std::cout << "Area:" << calc\_area(tuple\_intSqu) << std::endl;

std::cout << "Center:" << calc\_center(tuple\_intSqu) << std::endl;

}

} else if (option == "double") {

std::cout << "choose the type of shape input: through the [class] or [tuple]" << std::endl;

std::cin >> option;

if (option == "class") {

std::pair<double, double> A, C;

Square\_read\_cords(A, C);

Square<double> squ\_double(A, C);

std::cout << id << " Figure: Square double\n";

actions(squ\_double);

} else if (option == "tuple") {

std::pair<double, double> A, C;

Square\_read\_cords(A, C);

std::get<0>(tuple\_doubleSqu) = A;

std::get<1>(tuple\_doubleSqu) = C;

std::cout << id << " Figure: Square double\n";

Coord\_print(tuple\_doubleSqu);

std::cout << "Area:" << calc\_area(tuple\_doubleSqu) << std::endl;

std::cout << "Center:" <<calc\_center(tuple\_doubleSqu) << std::endl;

}

}

} else if (option == "Rec") {

id++;

std::cout << "Choose data type: int or double" << std::endl;

std::cin >> option;

if (option == "int") {

std::cout << "choose the type of shape input: through the [class] or [tuple]" << std::endl;

std::cin >> option;

if (option == "class") {

std::pair<int, int> A, B, C, D;

Rectangle\_read\_cords(A, B, C, D);

Rectangle<int> rec\_int(A, B, C, D);

std::cout << id << " Figure: Rectangle integer\n";

actions(rec\_int);

} else if (option == "tuple") {

std::pair<int, int> A, B, C, D;

Rectangle\_read\_cords(A, B, C, D);

std::get<0>(tuple\_intRec) = A;

std::get<1>(tuple\_intRec) = B;

std::get<2>(tuple\_intRec) = C;

std::get<3>(tuple\_intRec) = D;

std::cout << id << " Figure: Rectangle integer\n";

Coord\_print(tuple\_intRec);

std::cout << "Area:" << calc\_area(tuple\_intRec) << std::endl;

std::cout << "Center:" <<calc\_center(tuple\_intRec) << std::endl;

}

} else if (option == "double") {

std::cout << "choose the type of shape input: through the [class] or [tuple]" << std::endl;

std::cin >> option;

if (option == "class") {

std::pair<double, double> A, B, C, D;

Rectangle\_read\_cords(A, B, C, D);

Rectangle<double> rec\_double(A, B, C, D);

std::cout << id << " Figure: Rectangle double\n";

actions(rec\_double);

} else if (option == "tuple") {

std::pair<double, double> A, B, C, D;

Rectangle\_read\_cords(A, B, C, D);

std::get<0>(tuple\_doubleRec) = A;

std::get<1>(tuple\_doubleRec) = B;

std::get<2>(tuple\_doubleRec) = C;

std::get<3>(tuple\_doubleRec) = D;

std::cout << id << " Figure: Rectangle double\n";

Coord\_print(tuple\_doubleSqu);

std::cout << "Area:" << calc\_area(tuple\_doubleRec) << std::endl;

std::cout << "Center:" << calc\_center(tuple\_doubleRec) << std::endl;

}

}

} else if (option == "Menu") {

std::cout << "Enter a word for action:" << std::endl;

std::cout << "1) Create Triangle[Tri]" <<

std::endl << "2) Create Square [Squ]" <<

std::endl << "3) Create Rectangle [Rec]" <<

std::endl << "4) And then enter word to select data type: [int] or [double]" <<

std::endl << "5) Next choose where to write a figure: [tuple] or [class] object" <<

std::endl << "6) Demonstration of a tuple with figures[tuple\_work]" <<

std::endl << "7) Print Menu[Menu]" <<

std::endl << "8) Exit[Ex]" << std::endl;

} else if(option == "tuple\_work") {

std::cout << "\nExample with tuple usage:\n\n";

std::tuple<double\_pair, double\_pair> exm\_tup\_squ({1.1, 1.1}, {3.3,3.3});

Coord\_print(exm\_tup\_squ);

std::cout << "Area: "<< calc\_area(exm\_tup\_squ) << std::endl;

std::cout << "<<Center of the figure:" << calc\_center(exm\_tup\_squ) << std::endl;

std::tuple<int\_pair, int\_pair, int\_pair> exm\_tup\_tri({0,0}, {1,1}, {2,2});

std::cout << "Area: "<< calc\_area(exm\_tup\_tri) << std::endl;

}else {

std::cout << "You did not select an action or you were mistaken in entering the name of the action" << std::endl;

}

std::cin >> option;

}

return 0;

}

**figures.h**

#ifndef \_\_FIGURES\_H\_\_

#define \_\_FIGURES\_H\_\_

#include <type\_traits> // for enable\_if

#include <string>

#include <tuple>

#include <cmath>

#include <stdexcept>

#include <iostream>

template <class T>

std::pair<T, T> operator-(const std::pair<T, T> &p1, const std::pair<T, T> &p2){

return {p1.first-p2.first, p1.second-p2.second};

}

template <class T>

std::ostream& operator<<(std::ostream &o, const std::pair<T, T> &p){

o << "<" << p.first << ", " << p.second << ">";

return o;

}

template <class T>

bool operator==(const std::pair<T, T> &a, const std::pair<T, T> &b){

return (a.first == b.first) && (a.second == b.second);

}

bool isNumber(const std::string& s){

return !s.empty() && s.find\_first\_not\_of("-.0123456789") == std::string::npos; }

template <class T>

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

std::string checker;

is >> checker;

if(isNumber(checker) == false){

throw std::overflow\_error("Is not a number");

}

p.first = static\_cast<T>(std::stod(checker));

is >> checker;

if(isNumber(checker) == false){

throw std::overflow\_error("Is not a number");

}

p.second = static\_cast<T>(std::stod(checker));

return is;

}

template <class T>

bool collinear(const std::pair<T, T> &a, const std::pair<T, T> &b, const std::pair<T, T> &c, const std::pair<T, T> &d){

return (b.second-a.second)\*(d.first-c.first) - (d.second-c.second)\*(b.first-a.first) <= 1e-9;

}

template <int>

bool collinear(std::pair<int, int> &a, std::pair<int, int> &b, std::pair<int, int> &c, const std::pair<int, int> &d){

return (b.second-a.second)\*(d.first-c.first) == (d.second-c.second)\*(b.first-a.first);

}

template <class T>

bool perpendic(const std::pair<T, T> &a, const std::pair<T, T> &b, const std::pair<T, T> &d){

using vect = std::pair<T, T>;

vect AB = b-a;

vect AD = d-a;

T dotProduct = AB.first\*AD.first + AB.second\*AD.second;

if(dotProduct <= 1e-9 && dotProduct >= -1e-9) return true;

else return false;

}

template <class T>

double dist(const std::pair<T,T> &a, const std::pair<T,T> &b){

return sqrt( ((b.first - a.first) \* (b.first - a.first)) + ((b.second - a.second) \* (b.second - a.second)));

}

template <class T>

class Triangle{

public:

using type = T;

using point = std::pair<T, T>;

point A, B, C;

Triangle() = default;

Triangle(point &a, point &b, point &c){

/\*double AB = dist(a, b); //1

double AC = dist(a, c); //3

double BC = dist(b, c); //2

(AB + AC) > BC ) && ((AB + BC) > AC) && ((BC + AC) > AB);\*/

//(a.first-c.first)\*(b.second-c.second) == (a.second-c.second)\*(b.first-c.first)

if ( ((a.first-c.first)\*(b.second-c.second)) != ((a.second-c.second)\*(b.first-c.first)) ){

A = a;

B = b;

C = c;

}else{

throw std::overflow\_error("The given coordinates of the triangle lie on one straight line");

}

}

};

//Шаблон, который возвращает истинное значение value, если контейнер хранит фигуру rectangle

template <class T>

struct is\_Triangle{

static const bool value = false;

};

template <class T>

struct is\_Triangle<Triangle<T>>{

static const bool value = true;

};

template <class T>

using Triangle\_center = typename std::enable\_if<is\_Triangle<T>::value, std::pair<typename T::type, typename T::type> >::type;

template <class T>

using Triangle\_cords = typename std::enable\_if<is\_Triangle<T>::value, void >::type;

template <class T>

using Triangle\_area = typename std::enable\_if<is\_Triangle<T>::value, typename T::type >::type;

template <class T>

Triangle\_center<T> calc\_center(const T &tri){

typename T::type x\_center, y\_center;

x\_center = tri.A.first + tri.B.first + tri.C.first;

y\_center = tri.A.second + tri.B.second + tri.C.second;

return {x\_center, y\_center};

}

template <class T>

std::pair<T, T> calc\_center(const std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T> > &tp){

std::pair<T,T> a, b, c;

a = std::get<0>(tp);

b = std::get<1>(tp);

c = std::get<2>(tp);

Triangle<T> tri(a, b, c);

return calc\_center(tri);

}

template <class T>

Triangle\_cords<T> Coord\_print(const T &tri){

std::cout << "Cords of Triangle: " << tri.A << ", " << tri.B << ", " << tri.C << std::endl;

}

template <class T>

void Coord\_print(const std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T> > &tp){

std::pair<T,T> a, b, c;

a = std::get<0>(tp);

b = std::get<1>(tp);

c = std::get<2>(tp);

Triangle<T> tri(a, b, c);

std::cout << "Tuple. ";

Coord\_print(tri);

}

template <class T>

Triangle\_area<T> calc\_area(const T &tri) {

auto AB = dist(tri.A, tri.B);

auto AC = dist(tri.A, tri.C);

auto BC = dist(tri.B, tri.C);

auto p = (AB + AC + BC) / 2;

auto S = sqrt(p \* (p - AB) \* (p - AC) \* (p - BC));

return S;

}

template <class T>

T calc\_area(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> a, b, c;

a = std::get<0>(tp);

b = std::get<1>(tp);

c = std::get<2>(tp);

Triangle<T> tri(a, b, c);

return calc\_area(tri);

}

//for square

template <class T>

class Square{

public:

using type = T; // вот это нужно когда мы проверяем в enable if, если тот тип фигуры то даем тип данной фигуры что то такое

using point = std::pair<T, T>;

point A, B, C, D;

Square() = default;

Square(point &a, point &c){ // по ссылке обращение

point d(a.first, c.second);

point b(c.first, a.second);

auto AB = dist(a,b);

auto AD = dist(a,d);

if(a == c || AB != AD){

throw std::overflow\_error("Entered the coordinates of the vertices do not belong to the square.");

}else{

if(a.first > b.first){

C = a; A = c;

B = d; D = b;

}else{

A = a; C = c;

D = d; B = b;

}

}

}

};

template <class T>

struct is\_Square{

static const bool value = false;

};

template <class T>

struct is\_Square<Square<T>>{

static const bool value = true;

};

template <class T>

using Square\_center = typename std::enable\_if<is\_Square<T>::value, std::pair<typename T::type, typename T::type> >::type;

template <class T>

using Square\_cords = typename std::enable\_if<is\_Square<T>::value, void >::type;

template <class T>

using Square\_area = typename std::enable\_if<is\_Square<T>::value, typename T::type >::type;

template <class T>

Square\_center<T> calc\_center(const T &squ){

typename T::type x\_center, y\_center;

x\_center = (squ.A.first + squ.C.first)/2;

y\_center = (squ.A.second + squ.C.second)/2;

return {x\_center, y\_center};

}

template <class T>

std::pair<T, T> calc\_center(const std::tuple< std::pair<T, T>, std::pair<T, T> > &tp){

std::pair<T,T> a, c;

a = std::get<0>(tp);

c = std::get<1>(tp);

Square<T> squ(a, c);

return calc\_center(squ);

}

template <class T>

Square\_cords<T> Coord\_print(const T &squ){

std::cout << "Cords of Square: " << squ.A << ", " << squ.B << ", " << squ.C << ", " << squ.D << std::endl;

}

template <class T>

void Coord\_print(const std::tuple< std::pair<T, T>, std::pair<T, T> > &tp){

std::pair<T,T> a, c;

a = std::get<0>(tp);

c = std::get<1>(tp);

Square<T> squ(a, c);

std::cout << "Tuple. ";

Coord\_print(squ);

}

template <class T>

Square\_area<T> calc\_area(const T &squ) {

auto d = dist(squ.A, squ.C);

return (d\*d)/2;

}

template <class T>

T calc\_area(const std::tuple<std::pair<T, T>, std::pair<T, T> > &tp){

std::pair<T,T> a, c;

a = std::get<0>(tp);

c = std::get<1>(tp);

Square<T> squ(a, c);

return calc\_area(squ);

}

template <class T>

class Rectangle{

public:

using type = T;

using point = std::pair<T, T>;

point A, B, C, D;

Rectangle() = default;

Rectangle(point &a, point &b, point &c, point &d){

auto AB = dist(a,b);

auto AD = dist(a,d);

if(a == b || a == c || b == c || a == d ||

AB == AD || !(perpendic(a, b, d)) || !collinear(a, d, c, b) || !collinear(a, b, d, c)){

throw std::overflow\_error("The entered coordinates of the vertices do not belong to the rectangle.");

}else{

A = a;

B = b;

C = c;

D = d;

}

}

};

template <class T>

struct is\_Rectangle{

static const bool value = false;

};

template <class T>

struct is\_Rectangle<Rectangle<T>>{

static const bool value = true;

};

template <class T>

using Rectangle\_center = typename std::enable\_if<is\_Rectangle<T>::value, std::pair<typename T::type, typename T::type> >::type;

template <class T>

using Rectangle\_cords = typename std::enable\_if<is\_Rectangle<T>::value, void >::type;

template <class T>

using Rectangle\_area = typename std::enable\_if<is\_Rectangle<T>::value, typename T::type >::type;

template <class T>

Rectangle\_center<T> calc\_center(const T &rec){

typename T::type Ox, Oy;

Ox = (rec.A.first + rec.B.first + rec.C.first + rec.D.first)/4;

Oy = (rec.A.second + rec.B.second + rec.C.second + rec.D.second)/4;

return {Ox, Oy};

}

template <class T>

std::pair<T, T> calc\_center(const std::tuple< std::pair<T, T>, std::pair<T, T> , std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> a, b, c, d;

a = std::get<0>(tp);

b = std::get<1>(tp);

c = std::get<2>(tp);

d = std::get<3>(tp);

Rectangle<T> rec(a, b, c, d);

return calc\_center(rec);

}

template <class T>

Rectangle\_cords<T> Coord\_print(const T &rec){

std::cout << "Cords of Rectangle: " << rec.A << ", " << rec.B << ", " << rec.C << ", " << rec.D << std::endl;

}

template <class T>

void Coord\_print(const std::tuple< std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T> > &tp){

std::pair<T,T> a, b, c, d;

a = std::get<0>(tp);

b = std::get<1>(tp);

c = std::get<2>(tp);

d = std::get<3>(tp);

Rectangle<T> rec(a, b, c, d);

std::cout << "Tuple. ";

Coord\_print(rec);

}

template <class T>

Rectangle\_area<T> calc\_area(const T &rec) {

auto AB = dist(rec.A, rec.B);

auto AD = dist(rec.A, rec.D);

return AB\*AD;

}

template <class T>

T calc\_area(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T> > &tp){

std::pair<T,T> a, b, c, d;

a = std::get<0>(tp);

b = std::get<1>(tp);

c = std::get<2>(tp);

d = std::get<3>(tp);

Rectangle<T> rec(a, b, c, d);

return calc\_area(rec);

}

#endif

1. **Вывод**

Научился использовать шаблоны, где в качестве параметра используются скалярными данные, для работы с шаблонными клаcсами и кортежами.

**Список литературы**

1. Шилдт, Герберт. С++: базовый курс, 3-е изд. : Пер. с англ. - М. : ООО “И.Д. Вильямс”, 2018. - 624 с. : ил. - Парал. тит. англ.
2. Справочник по языку С++ [Электронный ресурс]. URL: [http://www.cplusplus.com/reference](http://www.cplusplus.com/reference/deque/) (дата обращения: 25.10.2019).
3. Справочник по языку С++ [Электронный ресурс]. URL:

<https://en.cppreference.com/w/> (дата обращения: 27.10.2019).