**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 4**

Тема: Основы метапрограммирования

Студент: Кочисов Алексей

Группа: М8О-201Б-20

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Создать программу на C++, которая позволяет:

* Создает набор фигур согласно варианту задания (как минимум по одной фигуре каждого типа с координатами типа int и координатами типа double).
* Сохраняет фигуры в std::tuple
* Печатает на экран содержимое std::tuple с помощью шаблонной функции print.
* Вычисляет суммарную площадь фигур в std::tuple и выводит значение на экран.

Использовать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь только публичные поля. В классах не должно быть методов, только поля. Фигуры являются фигурами вращения (равнобедренными), за исключением трапеции и прямоугольника. Для хранения координат фигур необходимо использовать шаблон std::pair.

**Вариант №13:**

|  |  |  |
| --- | --- | --- |
| **Фигура №1** | **Фигура №2** | **Фигура №3** |
| Ромб | 5-угольник | 6-угольник |

Все фигуры выпуклые и равносторонние.

**2. Листинг программы**

**Pentagon.h**

#ifndef PENTAGON\_H\_

#define PENTAGON\_H\_

#include <iostream>

#include <cmath>

#include "point.h"

template<class T>

struct pentagon {

Point<T> p[5];

pentagon(std::istream& is);

double area() const;

Point<T> center() const;

void print(std::ostream& os) const;

};

template<class T>

double pentagon<T>::area() const {

const T d1 = triangle\_height(p[0], p[1], p[2]) \* sqrt((p[2].x - p[1].x) \* (p[2].x - p[1].x) + (p[2].y - p[1].y) \* (p[2].y - p[1].y)) / 2;

const T d2 = triangle\_height(p[0], p[2], p[3]) \* sqrt((p[3].x - p[2].x) \* (p[3].x - p[2].x) + (p[3].y - p[2].y) \* (p[3].y - p[2].y)) / 2;

const T d3 = triangle\_height(p[0], p[3], p[4]) \* sqrt((p[4].x - p[3].x) \* (p[4].x - p[3].x) + (p[4].y - p[3].y) \* (p[4].y - p[3].y)) / 2;

return d1 + d2 + d3;

}

template<class T>

Point<T> pentagon<T>::center() const {

Point<T> res;

res.x = (p[0].x + p[1].x + p[2].x + p[3].x + p[4].x) / 5;

res.y = (p[0].y + p[1].y + p[2].y + p[3].y + p[4].y) / 5;

return res;

}

template<class T>

void pentagon<T>::print(std::ostream& os) const{

for(int i = 0; i < 5; ++i){

os << p[i];

if(i + 1 != 5){

os << ' ';

}

}

}

template<class T>

pentagon<T>::pentagon(std::istream& is) {

for(int i = 0; i < 5; ++i){

is >> p[i];

}

}

#endif

**Point.h**

#ifndef Point\_H\_

#define Point\_H\_

#include <iostream>

#include <cmath>

#include <limits>

template<class T>

struct Point {

T x;

T y;

};

template<class T>

Point<T> operator+(const Point<T>& A, const Point<T>& B) {

Point<T> res;

res.x = A.x + B.x;

res.y = A.y + B.y;

return res;

}

template<class T>

Point<T> operator/=(Point<T>& A, const double B) { //деление на число

A.x /= B;

A.y /= B;

return A;

}

template<class T>

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

is >> p.x >> p.y;

return is;

}

template<class T>

std::ostream& operator<< (std::ostream& os, const Point<T>& p) {

os << '(' << p.x << ';' << p.y << ')' ;

return os;

}

template <class T>

double triangle\_height(Point<T> p1, Point<T> p2, Point<T> p3) {

double A = p2.y - p3.y;

double B = p3.x - p2.x;

double C = p2.x\*p3.y - p3.x\*p2.y;

return (std::abs(A\*p1.x + B\*p1.y + C) / std::sqrt(A\*A + B\*B));

}

#endif

**Hexagon.h**

#ifndef D\_HEXAGON\_H

#define D\_HEXAGON\_H

#include <iostream>

#include <cmath>

#include "point.h"

template<class T>

struct hexagon {

Point<T> p[6];

hexagon(std::istream& is);

double area() const;

Point<T> center() const;

void print(std::ostream& os) const;

};

template<class T>

double hexagon<T>::area() const {

return (-0.5) \* ((p[0].x\*p[1].y + p[1].x\*p[2].y + p[2].x\*p[3].y + p[3].x\*p[4].y + p[4].x\*p[5].y + p[5].x\*p[0].y) - ( p[0].y\*p[1].x + p[1].y\*p[2].x + p[2].y\*p[3].x + p[3].y\*p[4].x + p[4].y\*p[5].x + p[5].y\*p[0].x ));

}

template<class T>

Point<T> hexagon<T>::center() const {

Point<T> res;

res.x = (p[0].x + p[1].x + p[2].x + p[3].x + p[4].x + p[5].x) / 6;

res.y = (p[0].y + p[1].y + p[2].y + p[3].y + p[4].y + p[5].y) / 6;

return res;

}

template<class T>

void hexagon<T>::print(std::ostream& os) const{

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

os << p[i];

if(i + 1 != 6){

os << ' ';

}

}

}

template<class T>

hexagon<T>::hexagon(std::istream& is) {

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

is >> p[i];

}

}

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H\_

#define RHOMBUS\_H\_

#include <iostream>

#include <cmath>

#include "point.h"

template<class T>

struct rhombus {

Point<T> p[4];

rhombus(std::istream& is);

double area() const;

Point<T> center() const;

void print(std::ostream& os) const;

};

template<class T>

rhombus<T>::rhombus(std::istream& is) {

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

is >> p[i];

}

double a, b, c, d;

a = sqrt((p[1].x - p[0].x) \* (p[1].x - p[0].x) + (p[1].y - p[0].y) \* (p[1].y - p[0].y));

b = sqrt((p[2].x - p[1].x) \* (p[2].x - p[1].x) + (p[2].y - p[1].y) \* (p[2].y - p[1].y));

c = sqrt((p[2].x - p[3].x) \* (p[2].x - p[3].x) + (p[2].y - p[3].y) \* (p[2].y - p[3].y));

d = sqrt((p[3].x - p[0].x) \* (p[3].x - p[0].x) + (p[3].y - p[0].y) \* (p[3].y - p[0].y));

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

throw std::logic\_error("Wrong coordinates. It's not a rhombus.");

}

template<class T>

double rhombus<T>::area() const {

const T d1 = sqrt((p[0].x - p[2].x) \* (p[0].x - p[2].x) + (p[0].y - p[2].y) \* (p[0].y - p[2].y));

const T d2 = sqrt((p[1].x - p[3].x) \* (p[1].x - p[3].x) + (p[1].y - p[3].y) \* (p[1].y - p[3].y));

return d1 \* d2 / 2;

}

template<class T>

Point<T> rhombus<T>::center() const {

Point<T> res;

res.x = (p[0].x + p[1].x + p[2].x + p[3].x) / 4;

res.y = (p[0].y + p[1].y + p[2].y + p[3].y) / 4;

return res;

}

template<class T>

void rhombus<T>::print(std::ostream& os) const {

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

os << p[i];

if(i + 1 != 4){

os << ' ';

}

}

}

#endif

**Templates.h**

#ifndef TEMPLATES\_H\_

#define TEMPLATES\_H\_

#include <tuple>

#include <type\_traits>

#include "point.h"

template<class T>

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

template<class T>

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

template<class T>

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

template<class T, class = void>

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

template<class T>

struct has\_method\_area<T, std::void\_t<decltype(std::declval<const T&>().area())>> : std::true\_type {};

template<class T>

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

template<class T>

std::enable\_if\_t<has\_method\_area\_v<T>, double> area(const T& object) {

return object.area();

}

template<class T, class = void>

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

template<class T>

struct has\_method\_center<T, std::void\_t<decltype(std::declval<const T&>().center())>> : std::true\_type {};

template<class T>

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

template<class T>

std::enable\_if\_t<has\_method\_center\_v<T>, Point<double>> center(const T& object) {

return object.center();

}

template<class T, class = void>

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

template<class T>

struct has\_method\_print<T, std::void\_t<decltype(std::declval<const T&>().print(std::cout))>> : std::true\_type {};

template<class T>

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

template<class T>

std::enable\_if\_t<has\_method\_print\_v<T>, void> print(std::ostream& os, const T& object) {

object.print(os);

}

template<size\_t Id, class T>

double compute\_area(const T& tuple) {

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

return 0;

}else{

const auto x1 = std::get<Id - 0>(tuple).x - std::get<0>(tuple).x;

const auto y1 = std::get<Id - 0>(tuple).y - std::get<0>(tuple).y;

const auto x2 = std::get<Id - 1>(tuple).x - std::get<0>(tuple).x;

const auto y2 = std::get<Id - 1>(tuple).y - std::get<0>(tuple).y;

const double local\_area = std::abs(x1 \* y2 - y1 \* x2) \* 0.5;

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

}

}

template<class T>

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

area(const T& object) {

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

throw std::logic\_error("It`s not a figure");

}else{

return compute\_area<2>(object);

}

}

template<size\_t Id, class T>

Point<double> tuple\_center(const T& object) {

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

return Point<double> {0, 0};

} else {

Point<double> res = std::get<Id>(object);

return res + tuple\_center<Id+1>(object);

}

}

template<class T>

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

Point<double> res{0, 0};

res = tuple\_center<0>(tuple);

res /= std::tuple\_size\_v<T>;

return res;

}

template<class T>

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

center(const T& object) {

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

throw std::logic\_error("It`s not a figure");

}else{

return compute\_center(object);

}

}

template<size\_t Id, class T>

void step\_print(const T& object, std::ostream& os) {

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

std::cout << "\n";

} else {

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

step\_print<Id + 1>(object, os);

}

}

template<class T>

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

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

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

throw std::logic\_error("It`s not a figure");

}else{

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

}

}

#endif

**Main.cpp**

#include <iostream>

#include <tuple>

#include "point.h"

#include "hexagon.h"

#include "pentagon.h"

#include "rhombus.h"

#include "templates.h"

template<class T>

void init(std::istream& is, std::ostream& os) {

if constexpr (is\_figurelike\_tuple<T>::value) {

int arg;

std::cin >> arg;

std::cout << "Input coordinates: " << std::endl;

if (arg == 4) {

Point<double> A, B, C, D;

is >> A >> B >> C >> D;

auto object = std::make\_tuple(A, B, C, D);

print(os, object);

os << "Area: " << area(object) << std::endl;

os << "Center: " << center(object) << std::endl;

}else if(arg == 5){

Point<double> A, B, C, D, F;

is >> A >> B >> C >> D >> F;

auto object = std::make\_tuple(A, B, C, D, F);

print(os, object);

os << "Area: " << area(object) << std::endl;

os << "Center: " << center(object) << std::endl;

}else if(arg == 6){

Point<double> A, B, C, D, F, G;

is >> A >> B >> C >> D >> F >> G;

auto object = std::make\_tuple(A, B, C, D, F, G);

print(os, object);

os << "Area: " << area(object) << std::endl;

os << "Center: " << center(object) << std::endl;

}

}else {

T object(is);

print(os, object);

os << '\n' <<"Area: " << area(object) << std::endl;

os << "Center: " << center(object) << std::endl;

}

}

int main() {

char obj\_type;

std:: cout<< " Available input:\n1 - input pentagon\n2 - input Rhombus\n3 - input hexagon\n4 - Tuple\n5 - Exit" <<std::endl;

while (std::cin >> obj\_type){

if(obj\_type == '4') {

std::cout << "Input number of vertices: " << std::endl;

init<std::tuple<Point<double>>> (std::cin, std::cout);

}else if(obj\_type == '1'){

std::cout << "Input pentagon coordinates: " << std::endl;

init<pentagon<double>>(std::cin, std::cout);

}else if(obj\_type == '2'){

std::cout << "Input Rhombus coordinates: " << std::endl;

init<rhombus<double>>(std::cin, std::cout);

}else if(obj\_type == '3'){

std::cout << "Input hexagon coordinates: " << std::endl;

init<hexagon<double>>(std::cin, std::cout);

}else if(obj\_type == '5'){

return 0;

}else{

std::cout << "Try another key" << std::endl;

}

}

}

1. **Набор тестов**

**Test1.test**

2

0 0 0 2 2 2 2 0

1

1 1 0 2 1 3 2 3 3 2

3

**Test2.test**

2

0 0 1 1 2 2 3 3

1. **Результаты выполнения тестов**

**Test1.result**

Available input:

1 - input pentagon

2 - input Rhombus

3 - input hexagon

4 - Tuple

5 - Exit

Input Rhombus coordinates:

(0;0) (0;2) (2;2) (2;0)

Area: 4

Center: (1;1)

Input pentagon coordinates:

(1;1) (0;2) (1;3) (2;3) (3;2)

Area: 3.5

Center: (1.4;2.2)

Input hexagon coordinates:

(2;0) (1;2) (2;4) (4;4) (5;2) (4;0)

Area: 12

Center: (3;2)

**Test2.result**

Available input:

1 - input pentagon

2 - input Rhombus

3 - input hexagon

4 - Tuple

5 - Exit

2

Input Rhombus coordinates:

0 0 1 1 2 2 3 3

terminate called after throwing an instance of 'std::logic\_error'

what(): Wrong coordinates. It's not a rhombus.

Aborted (core dumped)

1. **Объяснение результатов работы программы**

1) Шаблонная функция center() возвращает точку с x –деление суммы иксов всех точек данной фигуры на их количество, у – аналогично х. Она определена для моих фигур и tuple.

2) Функция print() печатает координаты всех точек данной фигуры или кортежа. Она определена для моих фигур и tuple.

3) Функция area() вычисляет площадь данной фигуры или совокупности точек и возвращает это значение.

**6. Выводы**

Выполняя данную лабораторную я получил опыт работы с шаблонами в С++. Узнал о применении шаблонов в метапрограммировании. Также я познакомился с полезными заголовочными файлами и освоил enable\_if, decltype и базовые вещи для работы с tuple. Данная лабораторная работа показала многогранность и мощь языка C++