**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: Гребенков Дмитрий Игоревич

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Разработать 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.

Необходимо реализовать две шаблонных функции:

1. Функция **print** печати фигур на экран std::cout (печататься должны координаты вершин фигур). Функция должна принимать на вход std::tuple с фигурами, согласно варианту задания (минимум по одной каждого класса).

2. Функция **square** вычисления суммарной площади фигур. Функция должна принимать на вход std::tuple с фигурами, согласно варианту задания (минимум по одной каждого класса).

Создать программу, 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.

Вариант задания: 12. Трапеция, ромб, пятиугольник.

1. Описание программы

Созданы шаблонные классы фигур по варианту задания. Эти классы содержат только публичные поля для хранения фигур и конструкторы для создания определённых объектов этих классов.

Для хранения всех фигур используется std::tuple.

Реализованы шаблонные функции для вывода всех вершин каждой фигуры и для подсчета общей площади фигур, хранящихся в tuple.

Взаимодействие с пользователем для ввода новых фигур не предусмотрено, так как все вычисления в программе происходят на этапе компиляции.

Примечание: для ускорения вычислений и упрощения хранения на вводимые фигуры установлены некоторые ограничения. Во-первых, основания вводимых трапеций должны быть параллельны оси Ox. Во-вторых, большее основание трапеции должно быть снизу. В-третьих, диагонали вводимых ромбов должны быть параллельны осям Ox и Oy.

Трапеция задается координатами трех вершин.

Ромб задается координатами двух вершин.

Пятиугольник задается центром описанной окружности и радиусом.

1. Набор тестов и результаты их выполнения

Фигуры в наборе тестов:

Trapezoid <int> tr1 (1, 2, 0, 0, 4, 0);

Trapezoid <double> tr2 (3.25, 4.25, 1.25, 1.25, 8.25, 1.25);

Rhombus <int> rh1 (-1, 0, 0, -1);

Rhombus <double> rh2 (-0.75, 1.0, 2.0, -0.5);

Pentagon <int> pg1 (5, 4, 9);

Pentagon <double> pg2 (3.72, 8.04, 1.36);

Результат работы программы:

Figure [0]:

(1, 2)

(0, 0)

(4, 0)

(3, 2)

Figure [1]:

(3.25, 4.25)

(1.25, 1.25)

(8.25, 1.25)

(6.25, 4.25)

Figure [2]:

(-1, 0)

(0, -1)

(1, 0)

(0, 1)

Figure [3]:

(-0.75, 1)

(2, -0.5)

(4.75, 1)

(2, 2.5)

Figure [4]:

(-3.55951, 6.78116)

(5, 13)

(13.5595, 6.78116)

(10.2901, -3.28116)

(-0.290063, -3.28116)

Figure [5]:

(2.42656, 8.46026)

(3.72, 9.4)

(5.01344, 8.46026)

(4.51939, 6.93974)

(2.92061, 6.93974)

Sum of squares of all figures in tuple: 227.648

1. Листинг программы

/\*

Гребенков Д. И.

Группа М8О-207Б-19

Вариант 12

Разработать 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.

Необходимо реализовать две шаблонных функции:

1. Функция print печати фигур на экран std::cout (печататься должны координаты вершин фигур). Функция должна

принимать на вход std::tuple с фигурами, согласно варианту задания (минимум по одной каждого класса).

2. Функция square вычисления суммарной площади фигур. Функция должна принимать на вход std::tuple с фигурами,

согласно варианту задания (минимум по одной каждого класса).

Создать программу, 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.

Вариант задания: Трапеция, ромб, пятиугольник.

\*/

#include <iostream>

#include <tuple>

#include <cmath>

const double pi = 3.1415926;

template <typename T>

class Trapezoid {

public:

using type = T;

std::pair <T, T> left\_top, left\_bottom, right\_bottom;

Trapezoid (T ltx, T lty, T lbx, T lby, T rbx, T rby) {

left\_top.first = ltx;

left\_top.second = lty;

left\_bottom.first = lbx;

left\_bottom.second = lby;

right\_bottom.first = rbx;

right\_bottom.second = rby;

};

};

template <typename T>

class Rhombus {

public:

using type = T;

std::pair <T, T> left, bottom;

Rhombus (T lx, T ly, T bx, T by) {

left.first = lx;

left.second = ly;

bottom.first = bx;

bottom.second = by;

};

};

template <typename T>

class Pentagon {

public:

using type = T;

std::pair <T, T> center;

T radius;

Pentagon (T cx, T cy, T r) {

center.first = cx;

center.second = cy;

radius = r;

};

};

template <int i = 0, class... Ts>

void vertices (std::tuple <Ts ...> t) {

if constexpr (i < std::tuple\_size <decltype (t)>::value) {

std::cout << std::endl;

std::cout << " Figure [" << i << "]:" << std::endl;

verticesN (std::get <i> (t));

vertices < i + 1, Ts ... > (t);

}

}

template <int i = 0, class... Ts>

void squares (std::tuple <Ts ...> t, double sq) {

if constexpr (i < std::tuple\_size <decltype (t)>::value){

sq += figure\_square (std::get <i> (t));

squares < i + 1, Ts ... > (t, sq);

}

else {

std::cout << std::endl;

std::cout << "Sum of squares of all figures in tuple: " << sq << std::endl;

}

}

template <typename T>

T figure\_square (Trapezoid <T> tr) {

return ((tr.right\_bottom.first - tr.left\_top.first) \* (tr.left\_top.second - tr.left\_bottom.second));

}

template <typename T>

T figure\_square (Rhombus <T> rh) {

return (2 \* (rh.bottom.first - rh.left.first) \* (rh.left.second - rh.bottom.second));

}

template <typename T>

T figure\_square (Pentagon <T> pg) {

return (5 \* pow (pg.radius, 2) \* sin (72 \* pi / 180.0) / 2.0);

}

template <typename T>

void verticesN (Trapezoid <T> tr) {

std::cout << "(" << tr.left\_top.first << ", " << tr.left\_top.second << ")" << std::endl;

std::cout << "(" << tr.left\_bottom.first << ", " << tr.left\_bottom.second << ")" << std::endl;

std::cout << "(" << tr.right\_bottom.first << ", " << tr.right\_bottom.second << ")" << std::endl;

std::cout << "(" << tr.right\_bottom.first - tr.left\_top.first + tr.left\_bottom.first <<

", " << tr.left\_top.second << ")" << std::endl;

}

template <typename T>

void verticesN (Rhombus <T> rh) {

std::cout << "(" << rh.left.first << ", " << rh.left.second << ")" << std::endl;

std::cout << "(" << rh.bottom.first << ", " << rh.bottom.second << ")" << std::endl;

std::cout << "(" << 2 \* rh.bottom.first - rh.left.first << ", " << rh.left.second << ")" << std::endl;

std::cout << "(" << rh.bottom.first << ", " << 2 \* rh.left.second - rh.bottom.second << ")" << std::endl;

}

template <typename T>

void verticesN (Pentagon <T> pg) {

std::cout << "(" << pg.center.first - pg.radius \* sin (72 \* pi / 180)

<< ", " << pg.center.second + pg.radius \* cos (72 \* pi / 180) << ")" << std::endl;

std::cout << "(" << pg.center.first << ", " << pg.center.second + pg.radius << ")" << std::endl;

std::cout << "(" << pg.center.first + pg.radius \* sin (72 \* pi / 180)

<< ", " << pg.center.second + pg.radius \* cos (72 \* pi / 180) << ")" << std::endl;

std::cout << "(" << pg.center.first + pg.radius \* sin (36 \* pi / 180)

<< ", " << pg.center.second - pg.radius \* cos (36 \* pi / 180) << ")" << std::endl;

std::cout << "(" << pg.center.first - pg.radius \* sin (36 \* pi / 180)

<< ", " << pg.center.second - pg.radius \* cos (36 \* pi / 180) << ")" << std::endl;

}

int main () {

Trapezoid <int> tr1 (1, 2, 0, 0, 4, 0);

Trapezoid <double> tr2 (3.25, 4.25, 1.25, 1.25, 8.25, 1.25);

Rhombus <int> rh1 (-1, 0, 0, -1);

Rhombus <double> rh2 (-0.75, 1.0, 2.0, -0.5);

Pentagon <int> pg1 (5, 4, 9);

Pentagon <double> pg2 (3.72, 8.04, 1.36);

std::tuple < Trapezoid <int>, Trapezoid <double>, Rhombus <int>, Rhombus <double>,

Pentagon <int>, Pentagon <double> > tupl {tr1, tr2, rh1, rh2, pg1, pg2};

vertices (tupl);

squares (tupl, 0);

return 0;

}

1. Вывод

Изучены основы работы с шаблонами в C++. Получены базовые навыки работы с шаблонами std::pair и std::tuple.

Литература:

1. Шаблоны классов в C++ | Уроки C++ - Ravesli [Электронный ресурс] URL: <https://ravesli.com/urok-175-shablony-klassov/> (дата обращения 01.11.2020).
2. С++ | Шаблон класса [Электронный ресурс] URL: <https://metanit.com/cpp/tutorial/9.1.php> (дата обращения 01.11.2020).