**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: Бирюков Виктор Владимирович

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:

* + 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).
  + 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 17: треугольник, квадрат, прямоугольник.

1. Описание программы

Класс Triangle хранит треугольник как координаты его левой вершины (в виде пары) и длину стороны. Класс Square хранит квадрат как координаты его левой нижней вершины (в виде пары) и длину стороны. Класс Rectangle хранит прямоугольник как координаты его нижней левой вершины (в виде пары) и длины сторон. Все поля классов имеют тип (или пару этого типа), переданный как параметр шаблона, который также сохранен как поле type.

Метафункция is\_tuple проверяет, что переданный тип является std::tuple. В таком случае значение поля value равно true, иначе — false

Шаблонная функция square может принимать на вход как std::tuple, так и Triangle, Square и Rectangle. Варианты этой функции для фигур проверяют, что тип аргумента — нужная фигура, при помощи std::enable\_if и std::is\_same. Версия для std::tuple использует std::enable\_if и is\_tuple. Площадь треугольника вычисляется по формуле: половина синуса угла (60°) на произведение (квадрат) сторон, квадрата — квадрат стороны, прямоугольника — произведение сторон. Если аргумент — кортеж, в функцию также передается шаблонный параметр index типа size\_t, по умолчанию равный 0. Если index меньше чем длина кортежа, вычисляется площадь элемента с этим индексом, затем рекурсивно вызывается функция с большим на единицу значением index.

Шаблонная функция print также принимает на вход std::tuple, Triangle, Square и Rectangle. Проверка типов производится аналогично. Координаты фигур выводятся по часовой стрелке. Вместе с кортежем в функцию передается параметр шаблона index, по умолчанию равный 0. Если index меньше чем длина кортежа, выводятся координаты элемента с этим индексом, затем рекурсивно вызывается функция с большим на единицу значением index.

1. Набор тестов

Программа принимает на вход данные о шести фигурах: треугольника, квадрата и прямоугольника с целыми координатами и квадрата, прямоугольника и треугольника с вещественными координатами. Треугольник задается тремя числами: координатами левой вершины и длины стороны. Квадрат задается тремя числами: координатами левой нижней вершины и длины стороны. Прямоугольник задается четырьмя числами: координатами левой нижней вершины и длины сторон.

Программа выводит координаты всех фигур в кортеже и их суммарную площадь.

Тест 1:

Triangle(int): 0 0 1

Square(int): 0 0 2

Rectangle(int): 0 0 5 4

Square(double): -10 -10 2.2

Rectangle(double): 0 5 2.5 1.5

Triangle(double): 1 1 0.5

Coordinates:

(0, 0) (0.5, 0.866025) (1, 0)

(0, 0) (0, 2) (2, 2) (2, 0)

(0, 0) (0, 4) (5, 4) (5, 0)

(-10, -10) (-10, -7.8) (-7.8, -7.8) (-7.8, -10)

(0, 5) (0, 6.5) (2.5, 6.5) (2.5, 5)

(1, 1) (1.25, 1.43301) (1.5, 1)

Total square: 33.1313

Тест 2 (одинаковые площади):

Triangle(int): 1 2 3

Square(int): 0 0 2

Rectangle(int): 5 5 4 1

Square(double): -4.5 -1.5 2

Rectangle(double): 1.1 0.4 1.5 2.666666667

Triangle(double): 10 -5 3.0393427

Coordinates:

(1, 2) (2.5, 4.59808) (4, 2)

(0, 0) (0, 2) (2, 2) (2, 0)

(5, 5) (5, 6) (9, 6) (9, 5)

(-4.5, -1.5) (-4.5, 0.5) (-2.5, 0.5) (-2.5, -1.5)

(1.1, 0.4) (1.1, 3.06667) (2.6, 3.06667) (2.6, 0.4)

(10, -5) (11.5197, -2.36785) (13.0393, -5)

Total square: 23.8971

1. Листинг программы

#include <iostream>

#include <tuple>

#include <type\_traits>

#include <cmath>

template <class... Ts>

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

template <class... Ts>

struct is\_tuple<std::tuple<Ts...>> : std::true\_type {};

template <typename T>

class Triangle {

public:

using type = T;

// левая вершина и длина стороны

std::pair<T,T> x;

T a;

Triangle(T x1, T x2, T a) : x(x1,x2), a(a) {

if (a <= 0) {

throw std::invalid\_argument("Invalid triangle parameters");

}

}

};

template <typename T>

class Square {

public:

using type = T;

// левая нижняя грань и длина стороны

std::pair<T,T> x;

T a;

Square(T x1, T x2, T a) : x(x1,x2), a(a) {

if (a <= 0) {

throw std::invalid\_argument("Invalid square parameters");

}

}

};

template <typename T>

class Rectangle {

public:

using type = T;

// левая нижняя грань и длины сторон

std::pair<T,T> x;

T a;

T b;

Rectangle(T x1, T x2, T a, T b) : x(x1,x2), a(a), b(b) {

if (a <= 0 || b <= 0) {

throw std::invalid\_argument("Invalid rectangle parameters");

}

}

};

template <class T, size\_t index = 0>

typename std::enable\_if<is\_tuple<T>::value, void>::type print(T &tup) {

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

print(std::get<index>(tup));

print<T, index+1>(tup);

} else {

return;

}

}

template <class T>

typename std::enable\_if<std::is\_same<T, Triangle<typename T::type>>::value, void>::type print(T &tr) {

std::cout << "(" << tr.x.first << ", " << tr.x.second << ") " <<

"(" << tr.x.first + 1.0 / 2 \* tr.a << ", " <<

tr.x.second + sqrt(3) / 2 \* tr.a << ") " <<

"(" << tr.x.first + tr.a << ", " << tr.x.second << ")" << std::endl;

}

template <class T>

typename std::enable\_if<std::is\_same<T, Square<typename T::type>>::value, void>::type print(T &sq) {

std::cout << "(" << sq.x.first << ", " << sq.x.second << ") " <<

"(" << sq.x.first << ", " << sq.x.second + sq.a << ") " <<

"(" << sq.x.first + sq.a << ", " << sq.x.second + sq.a << ") " <<

"(" << sq.x.first + sq.a << ", " << sq.x.second << ")" << std::endl;

}

template <class T>

typename std::enable\_if<std::is\_same<T, Rectangle<typename T::type>>::value, void>::type print(T &rect) {

std::cout << "(" << rect.x.first << ", " << rect.x.second << ") " <<

"(" << rect.x.first << ", " << rect.x.second + rect.b << ") " <<

"(" << rect.x.first + rect.a << ", " << rect.x.second + rect.b << ") " <<

"(" << rect.x.first + rect.a << ", " << rect.x.second << ")" << std::endl;

}

template <class T, size\_t index = 0>

typename std::enable\_if<is\_tuple<T>::value, double>::type square(T &tup) {

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

double value = square(std::get<index>(tup));

value += square<T, index+1>(tup);

return value;

} else {

return 0;

}

}

template <class T>

typename std::enable\_if<std::is\_same<T, Triangle<typename T::type>>::value, double>::type square(const T &tr) {

return sqrt(3) / 4 \* tr.a \* tr.a;

}

template <class T>

typename std::enable\_if<std::is\_same<T, Square<typename T::type>>::value, typename T::type>::type square(const T &sq) {

return sq.a \* sq.a;

}

template <class T>

typename std::enable\_if<std::is\_same<T, Rectangle<typename T::type>>::value, typename T::type>::type square(const T &rect) {

return rect.a \* rect.b;

}

int main() {

int ix = 0, iy = 0, ia = 0, ib = 0;

double dx = 0, dy = 0, da = 0, db = 0;

try {

std::cout << "Triangle(int): ";

std::cin >> ix >> iy >> ia;

Triangle<int> tr1(ix, iy, ia);

ix = iy = ia = 0;

std::cout << "Square(int): ";

std::cin >> ix >> iy >> ia;

Square<int> sq1(ix, iy, ia);

ix = iy = ia = 0;

std::cout << "Rectangle(int): ";

std::cin >> ix >> iy >> ia >> ib;

Rectangle<int> rect1(ix, iy, ia, ib);

std::cout << "Square(double): ";

std::cin >> dx >> dy >> da;

Square<double> sq2(dx, dy, da);

dx = dy = da = 0;

std::cout << "Rectangle(double): ";

std::cin >> dx >> dy >> da >> db;

Rectangle<double> rect2(dx, dy, da, db);

dx = dy = da = db = 0;

std::cout << "Triangle(double): ";

std::cin >> dx >> dy >> da;

Triangle<double> tr2(dx, dy, da);

std::tuple<Triangle<int>,Square<int>,Rectangle<int>,

Square<double>,Rectangle<double>,Triangle<double>>

tup{tr1, sq1, rect1, sq2, rect2, tr2};

std::cout << "Coordinates:" << std::endl;

print(tup);

double ts = square(tup);

std::cout << "Total square: " << ts << std::endl;

} catch (std::invalid\_argument& ex) {

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

}

}

1. Выводы

В ходе лабораторной работы я познакомился с созданием шаблонных классов и функций в языке C++, а также с использованием типа “кортеж” и метафункций из библиотеки type\_traits.

Литература

1. Справочник по языку C++ [Электронный ресурс]. URL: https://ru.cppreference.com