**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: Терво Александр Александрович

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 координатоми 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.

При реализации шаблонных функций допускается использование вспомогательных шаблонов std::enable\_if, std::tuple\_size, std::is\_same.

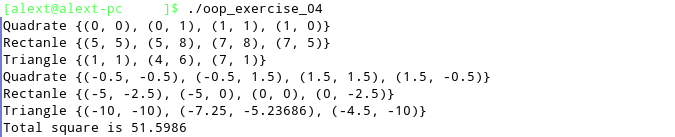
Вариант №1, фигуры: треугольник, квадрат, прямоугольник.

1. Описание программы

Для каждой фигуры будем хранить координаты левого нижнего угла в видеstd::pair<T, T>, для которого переопределим операцию вывода. Квадрат имеет поле длины стороны, прямоугольник – ширину и высоту, треугольник – длину стороны. Опишем шаблонные функции для вычисления площади и печати координат вершин каждой фигуры. Фигуры будем хранить в std::tuple, который создаётся std::make\_tuple. Печать фигур и вычисление площади реализуем с помощью выражение constexpr и std::get.

1. Вывод программы

Программа выводит на экран результат выполнения шаблонной функции PrintTuple и TotalSquare.



1. Листинг программы

main.cpp:

#include "quad.hpp"

#include "rectangle.hpp"

#include "triangle.hpp"

using IntVertex = std::pair<int, int>;

using DoubleVertex = std::pair<long double, long double>;

template<class T1, class T2>

std::ostream & operator << (std::ostream & out, const std::pair<T1, T2> & p) {

out << "(" << p.first << ", " << p.second << ")";

return out;

}

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

void PrintTuple(T & tup) {

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

std::cout << std::get<index>(tup) << std::endl;

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

}

}

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

long double TotalSquare(T & tup) {

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

return (long double)CalcSquare(std::get<index>(tup)) + TotalSquare<T, index + 1>(tup);

} else {

return 0;

}

}

int main() {

Quad<int> quadInt(IntVertex(0, 0), 1);

Rectangle<int> rectangleInt(DoubleVertex(5, 5), 2, 3);

Triangle<int> triangleInt(IntVertex(1, 1), 6);

Quad<long double> quadDouble = {DoubleVertex(-0.5, -0.5), 2};

Rectangle<long double> rectangleDouble(DoubleVertex(-5.0, -2.5), 5.0, 2.5);

Triangle<long double> triangleDouble(DoubleVertex(-10.0, -10.0), 5.5);

auto tup = std::make\_tuple(quadInt, rectangleInt, triangleInt, quadDouble, rectangleDouble, triangleDouble);

PrintTuple(tup);

std::cout << "Total square is " << TotalSquare(tup) << std::endl;

return 0;

}

quad.hpp:

#ifndef QUAD\_HPP

#define QUAD\_HPP

#include <iostream>

#include <tuple>

#include <vector>

template<class T>

struct Quad {

std::pair<T, T> Cord;

T Side;

Quad(const std::pair<T, T> & cord, T side) : Cord(cord), Side(side) {}

};

template<class T>

T CalcSquare(const Quad<T> & Sq) {

return Sq.Side \* Sq.Side;

}

template<class T>

std::ostream & operator << (std::ostream & out, const Quad<T> & sq) {

out << "Quadrate {";

out << std::pair<T, T>(sq.Cord.first, sq.Cord.second) << ", ";

out << std::pair<T, T>(sq.Cord.first, sq.Cord.second + sq.Side) << ", ";

out << std::pair<T, T>(sq.Cord.first + sq.Side, sq.Cord.second + sq.Side) << ", ";

out << std::pair<T, T>(sq.Cord.first + sq.Side, sq.Cord.second);

out << "}";

return out;

}

#endif

rectangle.hpp:

#ifndef RECTANGLE\_HPP

#define RECTANGLE\_HPP

#include <iostream>

#include <tuple>

#include <vector>

template<class T>

struct Rectangle {

std::pair<T, T> Cord;

T Width, Height;

Rectangle(const std::pair<T, T> & cord, T width, T height) : Cord(cord), Width(width), Height(height) {}

};

template<class T>

T CalcSquare(const Rectangle<T> & rectangle) {

return rectangle.Width \* rectangle.Height;

}

template<class T>

std::ostream & operator << (std::ostream & out, const Rectangle<T> & rectangle) {

out << "Rectanle {";

out << std::pair<T, T>(rectangle.Cord.first, rectangle.Cord.second) << ", ";

out << std::pair<T, T>(rectangle.Cord.first, rectangle.Cord.second + rectangle.Height) << ", ";

out << std::pair<T, T>(rectangle.Cord.first + rectangle.Width, rectangle.Cord.second + rectangle.Height) << ", ";

out << std::pair<T, T>(rectangle.Cord.first + rectangle.Width, rectangle.Cord.second);

out << "}";

return out;

}

#endif

triangle.hpp:

#ifndef TRIANGLE\_HPP

#define TRIANGLE\_HPP

#include <iostream>

#include <tuple>

#include <vector>

#include <cmath>

template<class T>

struct Triangle {

std::pair<T, T> Cord;

T Side;

Triangle(const std::pair<T, T> & cord, T side) : Cord(cord), Side(side) {}

};

template<class T>

T CalcSquare(const Triangle<T> & Tri) {

return Tri.Side \* Tri.Side \* (std::pow(3, 0.5) / 4);

}

template<class T>

std::ostream & operator << (std::ostream & out, const Triangle<T> & tri) {

out << "Triangle {";

out << std::pair<T, T>(tri.Cord.first, tri.Cord.second) << ", ";

out << std::pair<T, T>(tri.Cord.first + 0.5 \* tri.Side, tri.Cord.second + tri.Side \* (std::pow(3, 0.5) / 2)) << ", ";

out << std::pair<T, T>(tri.Cord.first + tri.Side, tri.Cord.second);

out << "}";

return out;

}

#endif

1. Выводы

В ходе работы мной были изучены принципы работы с шаблонами функций и классов в С++. Также были изучены шаблоны std::pair и std::tuple, с помощью которых удалось реализовать динамическую коллекцию геометрических фигур.

1. Список литературы

1. Основы шаблонов С++: шаблоны функций / Хабр — HabrURL: https://habr.com/ru/post/436880/

2. Шаблоны в C++ —ВикипедияURL: https://ru.wikipedia.org/wiki/Шаблоны\_C%2B%2B