**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: Назарова Анастасия Игоревна

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:09.11.20

Оценка:

Москва, 2020

1. Постановка задачи

Необходимо изучить основы работы с шаблонами (template) в С++, изучить шаблоны std::pair, std::tuple и приобрести навыка работы со специализацией шаблонов и идиомой SFINAE.

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).

1. Описание программы

Создаем набор фигур согласно варианту(8-угольник, прямоугольник и квадрат).Прописываем координаты в программе. Сохраняем фигуры в std::tuple.Печатаем на экран содержимое std::tuple с помощью шаблонной функции print, вычисляем суммарную площадь фигур в std::tuple и выводим значение на экран.

1. Набор тестов

В наборе тестов прописываем координаты каждой фигуры

|  |  |
| --- | --- |
| Входные данные | Результат |
| 0,0 4,0 4,4 0,4  0,0 10,0 5,4  2,0 4,0 6,2 6,4 4,6 2,6 0,4 0,2 | (0,0),(4,0),(4,4),(0,4)  (0,0),(10,0),(5,4)  (2,0) (4,0) (6,2) (6,4) (4,6) (2,6) (0,4) (0,2)  55 |
| 0,0 4,0 4,4 0,4  0,0 6,0 6,6 0,6  0,0 10,0 5,4  2,0 4,0 6,2 6,4 4,6 2,6 0,4 0,2 | (0,0),(4,0),(4,4),(0,4)  0,0 6,0 6,6 0,6  (0,0),(10,0),(5,4)  (2,0) (4,0) (6,2) (6,4) (4,6) (2,6) (0,4) (0,2)  91 |

1. Результаты выполнения тестов

Координаты вершин квадрата (0,0),(4,0),(4,4),(0,4)

Координаты вершин треугольника (0,0),(10,0),(5,4)

Координаты вершин восьмиугольника (2,0) (4,0) (6,2)

(6,4) (4,6) (2,6) (0,4) (0,2)

Площадь:55

Координаты вершин квадрата (0,0),(4,0),(4,4),(0,4)

Координаты вершин квадрата (0,0),(6,0),(6,6),(0,6)

Координаты вершин треугольника (0,0),(10,0),(5,4)

Координаты вершин восьмиугольника (2,0) (4,0) (6,2)

(6,4) (4,6) (2,6) (0,4) (0,2)

Площадь:91

1. Листинг программы

/\*24 вариант

https://github.com/Lostsunnydust/oop\_exercise\_04

Создаем набор фигур согласно варианту(8-угольник, прямоугольник и квадрат).Прописываем координаты в программе. Сохраняем фигуры в std::tuple.Печатаем на экран содержимое std::tuple с помощью шаблонной функции print, вычисляем суммарную площадь фигур в std::tuple и выводим значение на экран.\*/

#include <iostream>

#include <tuple>

#include <cmath>

template <class T> struct Tringle {

using type = T;

std::pair<T, T> t1, t2, t3;

};

template <class T> struct Squared {

using type = T;

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

};

template <class T, size\_t SIZE> struct Octagon {

using type = T;

std::pair<T, T> array[SIZE];

std::pair<T, T>\* begin() {

return &array[0];

}

std::pair<T, T>\* end() {

return &array[SIZE];

}

};

//печать на экран tuple

template <class T, size\_t index> typename std::enable\_if < index >= std::tuple\_size<T>::value, void>::type print\_tuple(T&) {

std::cout << std::endl;

}

template <class T, size\_t index> typename std::enable\_if < index<std::tuple\_size<T>::value, void>::type print\_tuple(T& tuple) {

auto figure = std::get<index>(tuple);

print(figure);

print\_tuple<T, index + 1>(tuple);

}

//печать на экран координат

template <class T> typename std::enable\_if<(sizeof(T::a) > 0), void>::type print(T& vertex) {

std::cout << "Координаты вершин квадрата";

std::cout << "(" << vertex.a.first << "," << vertex.a.second << ")" << "," << "(" << vertex.b.first << "," << vertex.b.second << ")" << "," << "(" << vertex.c.first << "," << vertex.c.second << ")";

std::cout << "," << "(" << vertex.d.first << "," << vertex.d.second << ")";

std::cout << std::endl;

}

template <class T> typename std::enable\_if<(sizeof(T::t1) > 0), void>::type print(T& vertex1) {

std::cout << "Координаты вершин треугольника";

std::cout << "(" << vertex1.t1.first << "," << vertex1.t1.second << ")" << "," << "(" << vertex1.t2.first << "," << vertex1.t2.second << ")" << "," << "(" << vertex1.t3.first << "," << vertex1.t3.second << ")";

std::cout << std::endl;

}

template <class T> typename std::enable\_if<(sizeof(T::array) > 0), void>::type print(T& octagon) {

std::cout << "Координаты вершин восьмиугольника";

for (auto vertex : octagon)

std::cout << "(" << vertex.first << "," << vertex.second << ")" << " ";

std::cout << std::endl;

}

template <class T> typename std::enable\_if<(sizeof(T::a) > 0), typename T::type>::type square(T& vertex) {

return static\_cast<typename T::type>((vertex.b.first - vertex.a.first) \* (vertex.b.first - vertex.a.first));

}

template <class T> typename std::enable\_if<(sizeof(T::t3) > 0), typename T::type>::type square(T& vertex1) {

return static\_cast<typename T::type>(((vertex1.t3.second - vertex1.t1.second) \* (vertex1.t2.first - vertex1.t1.first)) / 2);

}

template <class T> typename std::enable\_if<(sizeof(T::array) > 0), typename T::type>::type square(T& octagon) {

return static\_cast<typename T::type>((octagon.array[1].first - octagon.array[0].first) \* (octagon.array[1].first - octagon.array[0].first) \* 2 \* (1 + sqrt(2)));

}

//считаем площадь tuple

template <class T, size\_t index> double square\_tuple(T& tuple) {

auto item = std::get<index>(tuple);

double value = square(item);

if constexpr ((index + 1) < std::tuple\_size<T>::value) {

return value + square\_tuple<T, index + 1>(tuple);

}

return value;

}

int main()

{

setlocale(LC\_ALL, "Russian");

Squared<int> squared1;

squared1.a = { 0,0 };

squared1.b = { 4,0 };

squared1.c = { 4,4 };

squared1.d = { 0,4 };

Squared<int> squared2;

squared2.a = { 0,0 };

squared2.b = { 6,0 };

squared2.c = { 6,6 };

squared2.d = { 0,6 };

Tringle<int> tringle1;

tringle1.t1 = { 0,0 };

tringle1.t2 = { 10,0 };

tringle1.t3 = { 5,4 };

Octagon<int, 8> o1;

o1.array[0] = { 2,0 };

o1.array[1] = { 4,0 };

o1.array[2] = { 6,2 };

o1.array[3] = { 6,4 };

o1.array[4] = { 4,6 };

o1.array[5] = { 2,6 };

o1.array[6] = { 0,4 };

o1.array[7] = { 0,2 };

std::tuple<decltype(squared1),decltype(squared2), decltype(tringle1), decltype(o1) >tuple{ squared1,squared2,tringle1,o1 };

print\_tuple<decltype(tuple), 0>(tuple);

std::cout << "Площадь:" << square\_tuple<decltype(tuple), 0>(tuple) << std::endl;

return 1;

}

1. Вывод

Я изучила основы работы с шаблонами (template) в С++, изучила шаблоны std::pair, std::tuple и приобрела навыки работы со специализацией шаблонов и идиомой SFINAE. Разработала шаблоны классов согласно варианту задания.

1. Список используемых источников
2. Работа с кортежами [Электронный ресурс]. URL:https://habr.com/ru/post/318236/(Дата обращения: 1.11.2020)
3. Вычисление площади фигур [Электронный ресурс]. URL:https://tehtab.ru/guide/guidemathematics/perimsqvolgradrad/squaresofplainfigures/flatfigurescalculation/Дата обращения: 18.10.2020)