**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: Волков

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.

Например:

template <class T>

struct Square{

using vertex\_t = std::pair<T,T>;

vertex\_t a,b,c,d;

};

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.

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

*Вариант: Прямоугольник, трапеция, ромб*

1. Описание программы

Есть 3 класса: *Rectangle, Rhombus, Trapeze.* В них содержатся данные, как и в 3-ей лабораторной, то есть координаты точки левой вершины и стороны. Для каждого класса прописан свой конструктор.

В самом верху написана самостоятельная проверка на тип *tuple.* Далее с помощью метапрограммирования написаны функции print и area. Они печатают информацию о фигуре и находят площадь соответственно.

В *main* здесь происходит тестирование вышеописанных классов и метапрограммирования.

1. Наборы и Результаты выполнения тестов

*Тест 1*

Rectangle<int>. There are 4 parameters:

coord point

2 edges

1 1 -1 3

Invalid Rectangle parameters

Process finished with exit code 1

*Тест 2*

*Rectangle<int>. There are 4 parameters:*

*coord point*

*2 edges*

*1 1 2 2*

*Rhombus<int>. There are 4 parameters:*

*coord point*

*2 edges*

*2 2 8 1*

*Trapeze<int>. There are 5 parameters:*

*coord points*

*3 edges*

*4 8 8 1 2*

*Rectangle<double>. There are 4 parameters:*

*coord point*

*2 edges*

*3 2 2 2*

*Rhombus<double>. There are 4 parameters:*

*coord point*

*2 edges*

*3 4 5 6*

*Trapeze<double>. There are 5 parameters:*

*coord points*

*3 edges*

*7 8 8 10 11*

*Coord:*

*Point coord: (1, 1)*

*Edges: a=2 b=2*

*Point coord: (2, 2)*

*Edges: d1=8 d2=1*

*Point coord: (4, 8)*

*Edges: a=8 b=1 c=2*

*Point coord: (3, 2)*

*Edges: a=2 b=2*

*Point coord: (3, 4)*

*Edges: d1=5 d2=6*

*Point coord: (7, 8)*

*Edges: a=8 b=10 c=11*

*Total area: 138.515*

*Process finished with exit code 0*

*Тест 3*

Rectangle<int>. There are 4 parameters:

coord point

2 edges

1 1 1 1

Rhombus<int>. There are 4 parameters:

coord point

2 edges

1 1 1 1

Trapeze<int>. There are 5 parameters:

coord points

3 edges

1 1 1 1 1

Rectangle<double>. There are 4 parameters:

coord point

2 edges

f f f f f

Invalid Rectangle parameters

Process finished with exit code 1

1. Листинг программы

/\*Волков Матвей Андреевич

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.

Например:

template <class T>

struct Square{

using vertex\_t = std::pair<T,T>;

vertex\_t a,b,c,d;

};

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.

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

Вариант: Прямоугольник, трапеция, ромб

\*/

#include <iostream>

#include <type\_traits>

#include <tuple>

#include <cmath>

template<class... Ts>

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

};

template<class... Ts>

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

};

template<class T>

class Rectangle {

public:

using type = T;

//левая нижняя вершина и 2 стороны

std::pair<T, T> point;

T a, b;

Rectangle(T x, T y, T A, T B) : point(x, y), a(A), b(B) {

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

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

}

}

};

template<class T>

class Rhombus {

public:

using type = T;

//левая вершина и 2 диагонали, причем ромб расположен так, что диагонали параллельны осям XOY

std::pair<T, T> point;

T d1, d2;

Rhombus(T x, T y, T D1, T D2) : point(x, y), d1(D1), d2(D2) {

if (d1 <= 0 || d2 <= 0) {

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

}

}

};

template<class T>

class Trapeze {

public:

using type = T;

//левая нижняя вершина и основания

std::pair<T, T> point;

T a, b, c;

Trapeze(T x, T y, T A, T B, T C) : point(x, y), a(A), b(B), c(C) {

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

throw std::invalid\_argument("Invalid Trapeze 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\_fig(std::get<index>(tup));

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

}

else {

return;

}

}

template<class T>

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

std::cout << "Point coord: (" << req.point.first << ", " << req.point.second << ')' << std::endl

<< "Edges: a=" << req.a << " b=" << req.b << '\n' << std::endl;

}

template<class T>

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

std::cout << "Point coord: (" << rho.point.first << ", " << rho.point.second << ')' << std::endl

<< "Edges: d1=" << rho.d1 << " d2=" << rho.d2 << '\n' << std::endl;

}

template<class T>

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

std::cout << "Point coord: (" << tra.point.first << ", " << tra.point.second << ')' << std::endl

<< "Edges: a=" << tra.a << " b=" << tra.b << " c=" << tra.c << '\n' << std::endl;

}

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

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

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

double value = area\_fig(std::get<index>(tup));

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

return value;

} else {

return 0;

}

}

template <class T>

typename std::enable\_if<std::is\_same<T, Rectangle<typename T::type>>::value, double>::type area\_fig(const T &req){

return req.a \* req.b;

}

template <class T>

typename std::enable\_if<std::is\_same<T, Rhombus<typename T::type>>::value, double>::type area\_fig(const T &rum){

return rum.d1 \* rum.d2 / 2;

}

template <class T>

typename std::enable\_if<std::is\_same<T, Trapeze<typename T::type>>::value, double>::type area\_fig(const T &tra){

double diff = (std::fabs(tra.a - tra.b)) / 2.0;

double height = std::sqrt(std::fabs(tra.c \* tra.c - diff \* diff));

return height \* (tra.a + tra.b) / 2.0;

}

int main() {

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

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

try{

std::cout << "Rectangle<int>. There are 4 parameters:\ncoord point\n2 edges" << std::endl;

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

Rectangle<int> req1(ix, iy, ia, ib);

ix = iy = ia = ib = 0;

std::cout << "Rhombus<int>. There are 4 parameters:\ncoord point\n2 edges" << std::endl;

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

Rhombus<int> rum1(ix, iy, ia, ib);

ix = iy = ia = ib = 0;

std::cout << "Trapeze<int>. There are 5 parameters:\ncoord points\n3 edges" << std::endl;

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

Trapeze<int> tra1(ix, iy, ia, ib, ic);

std::cout << "Rectangle<double>. There are 4 parameters:\ncoord point\n2 edges" << std::endl;

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

Rectangle<double> req2(dx, dy, da, db);

dx = dy = da = db = 0;

std::cout << "Rhombus<double>. There are 4 parameters:\ncoord point\n2 edges" << std::endl;

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

Rhombus<double> rum2(dx, dy, da, db);

dx = dy = da = db = 0;

std::cout << "Trapeze<double>. There are 5 parameters:\ncoord points\n3 edges" << std::endl;

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

Trapeze<double> tra2(dx, dy, da, db, dc);

std::tuple< Rectangle<int>, Rhombus<int>, Trapeze<int>,

Rectangle<double>, Rhombus<double>, Trapeze<double> >

tup{req1, rum1, tra1, req2, rum2, tra2};

std::cout << "Coord:" << std::endl;

print(tup);

double ts = area(tup);

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

}

catch (std::invalid\_argument& e) {

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

return 1;

}

}

1. **Вывод**

Благодаря проделанной работе, я узнал что такое *template*, познал дзен метапрограммирования и понял, что это очень сложно. Узнал о существовании контейнера *tuple* и о встроенной функции проверки типа *enable\_if.*

**Список литературы:**

1. Основы шаблонов С++: шаблоны функций / Хабр — Habr

URL: <https://habr.com/ru/post/436880/> (дата обращения 30.10.2020).

1. Шаблоны в C++ — Википедия

URL: [https://ru.wikipedia.org/wiki/Шаблоны\_C%2B%2B](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_C%2B%2B) (дата обращения 30.10.2020).