**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: Лагуткина Мария Сергеевна

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Вариант 10: квадрат, прямоугольник, трапеция.

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.

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

1. Описание программы

На вход программе подаются координаты фигур: квадрата, прямоугольника и трапеции, для каждой фигуры в двух типа integer и double и производится проверка введенных координат на корректность: проверяется там, где это необходимо для соответствия введенной фигуры ее определению, на равенство сторон (стороны считаются равными, если разность между ними меньше значения машинного эпсилон). Если фигура введена некорректно, программа завершается с кодом 2 и выдается в поток ошибок сообщение о том, какая из фигур введена некорректно.

Если введенные координаты прошли проверку на корректность, то создается класс одной из указанных фигур с этими координатами. Далее создается кортеж из всех шести заданных классов.

Для каждой фигуры реализованы функции нахождения площади и вывода координат. Площадь фигур вычисляется по методу Гаусса. Для кортежа реализованы печать координат вершин всех фигур и подсчет суммарной площади фигур.

1. Набор тестов и результаты выполнения

Вводятся координаты квадрата, прямоугольника, трапеции типа int, затем типа double.

Тест 1:

0 0 0 1 1 1 1 0

0.5 0.5 0.5 1.5 1.5 1.5 1.5 0.5

1 3 3 3 3 1 1 1

1.2 2 1.2 5 4 5 4 2

0 0 3 4 6 4 9 0

-3 -2 0 2 2.5 2 5.5 -2

Результат:

Enter squares of integer:

0 0 0 1 1 1 1 0

Enter squares of double:

0.5 0.5 0.5 1.5 1.5 1.5 1.5 0.5

Enter rectangle of integer:

1 3 3 3 3 1 1 1

Enter rectangle of double:

1.2 2 1.2 5 4 5 4 2

Enter trapezoid of integer:

0 0 3 4 6 4 9 0

Enter trapezoid of double:

-3 -2 0 2 2.5 2 5.5 -2

(0,0)

(0,1)

(1,1)

(1,0)

(0.5,0.5)

(0.5,1.5)

(1.5,1.5)

(1.5,0.5)

(1,3)

(3,3)

(3,1)

(1,1)

(1.2,2)

(1.2,5)

(4,5)

(4,2)

(0,0)

(3,4)

(6,4)

(9,0)

(-3,-2)

(0,2)

(2.5,2)

(5.5,-2)

Total area: 60.4

Тест 2:

0 2 2 0 0 -2 -2 0

0 0 0 1.3 1.3 1.3 1 2 2

Результат:

Error input: square

Тест 3:

0 2 2 0 0 -2 -2 0

0 0 0 1.3 1.3 1.3 1.3 0

2 2 2 4 4 4 4 2

1.2 2 1.2 5 4 5 4 2

-3 -2 0 2 3 2 6 -2

-3 1.5 1 4.5 2 4.5 6 1.5

Результат:

Enter squares of integer:

0 2 2 0 0 -2 -2 0

Enter squares of double:

0 0 0 1.3 1.3 1.3 1.3 0

Enter rectangle of integer:

2 2 2 4 4 4 4 2

Enter rectangle of double:

1.2 2 1.2 5 4 5 4 2

Enter trapezoid of integer:

-3 -2 0 2 3 2 6 -2

Enter trapezoid of double:

-3 1.5 1 4.5 2 4.5 6 1.5

(0,2)

(2,0)

(0,-2)

(-2,0)

(0,0)

(0,1.3)

(1.3,1.3)

(1.3,0)

(2,2)

(2,4)

(4,4)

(4,2)

(1.2,2)

(1.2,5)

(4,5)

(4,2)

(-3,-2)

(0,2)

(3,2)

(6,-2)

(-3,1.5)

(1,4.5)

(2,4.5)

(6,1.5)

1. Листинг программы

#include <iostream>

#include <cmath>

#include <tuple>

#include <fstream>

#include <string>

#include <utility>

**using** **namespace** std;

**template** <**class** T>

**using** vertex\_t = pair<T, T>;

**template** <**class** T>

istream& **operator**>> (istream& input, vertex\_t<T>& v) {

input >> v.first >> v.second;

**return** input;

}

**template**<**class** T>

ostream& **operator**<< (ostream& output, **const** vertex\_t<T> v) {

output << "(" << v.first << "," << v.second << ")" << '\n';

**return** output;

}

**template** <**class** T>

**class** Square{

**public**:

vertex\_t<T> a, b, c, d;

};

**template** <**class** T>

**class** Rectangle {

**public**:

vertex\_t<T> a, b, c, d;

};

**template** <**class** T>

**class** Trapezoid {

**public**:

vertex\_t<T> a, b, c, d;

};

**template** <**class** T>

**double** area(**const** Square<T>& sq) {

**return** abs(sq.a.first\*sq.b.second + sq.b.first\*sq.c.second +

sq.c.first\*sq.d.second + sq.d.first\*sq.a.second -

sq.b.first\*sq.a.second - sq.c.first\*sq.b.second -

sq.d.first\*sq.c.second - sq.a.first\*sq.d.second)/2;

}

**template** <**class** T>

**double** area(**const** Rectangle<T>& re) {

**return** abs(re.a.first\*re.b.second + re.b.first\*re.c.second +

re.c.first\*re.d.second + re.d.first\*re.a.second -

re.b.first\*re.a.second - re.c.first\*re.b.second -

re.d.first\*re.c.second - re.a.first\*re.d.second)/2;

}

**template** <**class** T>

**double** area(**const** Trapezoid<T>& tr) {

**return** abs(tr.a.first\*tr.b.second + tr.b.first\*tr.c.second +

tr.c.first\*tr.d.second + tr.d.first\*tr.a.second -

tr.b.first\*tr.a.second - tr.c.first\*tr.b.second -

tr.d.first\*tr.c.second - tr.a.first\*tr.d.second)/2;

}

**template** <**class** T>

istream& **operator**>> (istream& input, Square<T>& sq) {

input >> sq.a >> sq.b >> sq.c >> sq.d;

**return** input;

}

**template** <**class** T>

istream& **operator**>> (istream& input, Rectangle<T>& re) {

input >> re.a >> re.b >> re.c >> re.d;

**return** input;

}

**template** <**class** T>

istream& **operator**>> (istream& input, Trapezoid<T>& tr) {

input >> tr.a >> tr.b >> tr.c >> tr.d;

**return** input;

}

**template** <**class** T>

**void** print(**const** Square<T>& sq) {

cout << sq.a << ' ' << sq.b << ' ' << sq.c << ' ' << sq.d << '\n';

**return**;

}

**template** <**class** T>

**void** print(**const** Rectangle<T>& re) {

cout << re.a << ' ' << re.b << ' ' << re.c << ' ' << re.d << '\n';

**return**;

}

**template** <**class** T>

**void** print(**const** Trapezoid<T>& tr) {

cout << tr.a << ' ' << tr.b << ' ' << tr.c << ' ' << tr.d << '\n';

**return**;

}

**template** <**class**... Elements>

**void** print(**const** tuple<Elements...> t) {

**if** (tuple\_size<tuple<Elements...>>::value != 6) {

cerr << "Incorrect number of figures entered.\n";

**return**;

}

print(get<0>(t));

print(get<1>(t));

print(get<2>(t));

print(get<3>(t));

print(get<4>(t));

print(get<5>(t));

**return**;

}

**template**<**class**... Elements>

**void** area(**const** tuple<Elements...> t) {

**if** (tuple\_size<tuple<Elements...>>::value != 6) {

cerr << "Incorrect number of figures entered.\n";

**return**;

}

**double** sum\_area = 0;

sum\_area += area(get<0>(t));

sum\_area += area(get<1>(t));

sum\_area += area(get<2>(t));

sum\_area += area(get<3>(t));

sum\_area += area(get<4>(t));

sum\_area += area(get<5>(t));

cout <<"\nTotal area: "<< sum\_area;

**return**;

}

**template**<**class** T>

**double** length(vertex\_t<T>& a, vertex\_t<T>& b) { //вычисление длины стороны

**return** sqrt((b.first - a.first)\*(b.first - a.first) +

((b.second - a.second)\*(b.second - a.second)));

}

**template**<**class** T>

**int** CorrectInput(pair<T, T>& a, pair<T, T>& b,

pair<T, T>& c, pair<T, T>& d, **int** f) {

//проверка корректности введенной фигуры

**if** ((a.first == c.first && a.second == c.second) || (b.first == d.first && b.second == d.second)) {

**return** 1;

}

**double** len\_ab = length(a, b);

**double** len\_bc = length(b, c);

**double** len\_cd = length(c, d);

**double** len\_da = length(d, a);

**double** len\_ac = length(a, c);

**double** len\_bd = length(b, d);

//проверка коректности квадрата: равенсво строн и диагоналей

**if** (f == 1) {

**if** (abs(len\_cd - len\_ab)<numeric\_limits<**double**>::epsilon() &&

abs(len\_cd - len\_bc)<numeric\_limits<**double**>::epsilon() &&

abs(len\_cd - len\_da)<numeric\_limits<**double**>::epsilon() &&

abs(len\_bd - len\_ac)<numeric\_limits<**double**>::epsilon()) {

**return** 0;

}

**return** 1;

}

//проверка корректности прямоугольника: равенство противоположных

сторон и диагоналей

**if** (f == 2) {

**if** (abs(len\_ab - len\_cd)<numeric\_limits<**double**>::epsilon() &&

abs(len\_bc - len\_da)<numeric\_limits<**double**>::epsilon() &&

abs(len\_ac - len\_bd)<numeric\_limits<**double**>::epsilon()) {

**return** 0;

}

**return** 1;

}

//проверка корректности равнобедренной трапеции: равенство боковых

сторон и диагоналей

**if** (f == 3) {

**if** (abs(len\_ab - len\_cd)<numeric\_limits<**double**>::epsilon() &&

abs(len\_ac - len\_bd)<numeric\_limits<**double**>::epsilon()) {

**return** 0;

}

**return** 1;

}

**return** 0;

}

**int** main() {

pair<**int**, **int**> a\_int, b\_int, c\_int, d\_int;

pair<**double**, **double**> a\_dob, b\_dob, c\_dob, d\_dob;

cout << "Enter squares of integer:" << '\n';

cin >> a\_int >> b\_int >> c\_int >> d\_int;

**int** f = CorrectInput(a\_int, b\_int, c\_int, d\_int, 1);

**if** (f == 1) { cerr << "Error input: square"; **return** 2; }

Square<**int**> s\_i{a\_int,b\_int,c\_int,d\_int };

cout << "Enter squares of double:" << '\n';

cin >> a\_dob >> b\_dob >> c\_dob >> d\_dob;

f = CorrectInput(a\_dob, b\_dob, c\_dob, d\_dob, 1);

**if** (f == 1) { cerr << "Error input: square"; **return** 2; }

Square<**double**> s\_d{ a\_dob,b\_dob,c\_dob,d\_dob };

cout << "Enter rectangle of integer:" << '\n';

cin >> a\_int >> b\_int >> c\_int >> d\_int;

f = CorrectInput(a\_int, b\_int, c\_int, d\_int, 2);

**if** (f == 1) { cerr << "Error input: square"; **return** 2; }

Rectangle<**int**> r\_i{ a\_int,b\_int,c\_int,d\_int };

cout << "Enter rectangle of double:" << '\n';

cin >> a\_dob >> b\_dob >> c\_dob >> d\_dob;

f = CorrectInput(a\_dob, b\_dob, c\_dob, d\_dob, 2);

**if** (f == 1) { cerr << "Error input: square"; **return** 2; }

Rectangle<**double**> r\_d{ a\_dob,b\_dob,c\_dob,d\_dob };

cout << "Enter trapezoid of integer:" << '\n';

cin >> a\_int >> b\_int >> c\_int >> d\_int;

f = CorrectInput(a\_int, b\_int, c\_int, d\_int, 3);

**if** (f == 1) { cerr << "Error input: square"; **return** 2; }

Trapezoid<**int**> t\_i{ a\_int,b\_int,c\_int,d\_int };

cout << "Enter trapezoid of double:" << '\n';

cin >> a\_dob >> b\_dob >> c\_dob >> d\_dob;

f = CorrectInput(a\_dob, b\_dob, c\_dob, d\_dob, 3);

**if** (f == 1) { cerr << "Error input: square"; **return** 2; }

Trapezoid<**double**> t\_d{ a\_dob,b\_dob,c\_dob,d\_dob };

**auto** tup = make\_tuple(s\_i, s\_d, r\_i, r\_d, t\_i, t\_d);

print(tup);

area(tup);

**return** 0;

}

1. Выводы

Я узнала как работать с кортежем, также узнала, что в стандартной библиотеке C++ есть класс Pair, позволяющий определять пару значений.

1. Литература

1. Курс «Основы разработки на С++: белый пояс». [Электронный ресурс] URL:<https://www.coursera.org/learn/c-plus-plus-white> (дата обращения 20.10.2020).

2. Документация Microsoft по С++. [Электронный ресурс] URL:[https://docs.microsoft.com/ru-ru/?view=msvc-16](https://docs.microsoft.com/ru-ru/?view=msvc-160) (дата обращения 28.10.2020).