**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: Шараковский Юрий Дмитриевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Разработать шаблоны классов для квадрата, треугольника и прямоугольника (Вариант 25). 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. Для хранения координат фигур необходимо использовать шаблон std::pair.

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 шаблона ProcessTuple.

• вычисляет суммарную площадь фигур в std::tuple и выводит значение на экран.

1. Описание программы

В программе реализованы шаблоны структур Triangle<T>, Square<T> и Rectangle<T> а также шаблон для ввода этих структур Read<T, F<T>> и шаблоны для вывода координат фигуры на экран PrintFigure<T, F<T>>, шаблон GetFigureArea<T, F<T>> возвращает площадь фигуры, а шаблон ProcessTuple<F, size\_t index> обрабатывает кортеж фигур (выводит их на экран и считает площадь). Если в шаблон поступает не фигура то на экран выводится: Bad figure!

1. Набор тестов

testfile\_01 - содержит информацию для ввода 3 фигур с целочисленными координатами и 3 фигур с вещественными координатами:

1

0 0 100 100

0 0 100 100

0 0 100 100

2

0 0 100 100

0 0 100 100

0 0 100 100

-1

testfile\_02 - показывает то как программа справляется с ошибкой ввода:

2

1 1 2 2

1 1 2 2

1 1 2 2

1

adasdasd

1

10 20

10 20

10 20

-1

1. Результаты выполнения тестов

testfile\_01:

1. Целочисленные координаты

2. Вещественные координаты

Введите координаты для квадрата (центр и вершина):

Введите координаты для треугольника (центр и вершина):

Введите координаты для прямоугольника (центр и вершина):

Square: [100,100] [-100,100] [-100,-100] [100,-100]

Figure Area: 40000

Triangle: [100,100] [-137,37] [36,-137]

Figure Area: 25991.6

Rectangle: [100,100] [-100,100] [-100,-100] [100,-100]

Figure Area: 40000

Total area: 105992

1. Целочисленные координаты

2. Вещественные координаты

Введите координаты для квадрата (центр и вершина):

Введите координаты для треугольника (центр и вершина):

Введите координаты для прямоугольника (центр и вершина):

Square: [100,100] [-100,100] [-100,-100] [100,-100]

Figure Area: 40000

Triangle: [100,100] [-136.603,36.6025] [36.6025,-136.603]

Figure Area: 25980.8

Rectangle: [100,100] [-100,100] [-100,-100] [100,-100]

Figure Area: 40000

Total area: 105981

1. Целочисленные координаты

2. Вещественные координаты

Goodbye!

testfile\_02:

1. Целочисленные координаты

2. Вещественные координаты

Введите координаты для квадрата (центр и вершина):

Введите координаты для треугольника (центр и вершина):

Введите координаты для прямоугольника (центр и вершина):

Square: [2,2] [1.7949e-09,2] [-3.58979e-09,3.58979e-09] [2,-5.38469e-09]

Figure Area: 4

Triangle: [2,2] [-0.366025,1.36603] [1.36603,-0.366025]

Figure Area: 2.59808

Rectangle: [2,2] [1.7949e-09,2] [-3.58979e-09,3.58979e-09] [2,-5.38469e-09]

Figure Area: 4

Total area: 10.5981

1. Целочисленные координаты

2. Вещественные координаты

Введите координаты для квадрата (центр и вершина):

Введите координаты для треугольника (центр и вершина):

Введите координаты для прямоугольника (центр и вершина):

Square: [0,0] [0,0] [0,0] [0,0]

Figure Area: 0

Triangle: [0,0] [0,0] [0,0]

Figure Area: 0

Rectangle: [0,0] [0,0] [0,0] [0,0]

Figure Area: 0

Total area: 0

1. Листинг программы

#include <iostream>

#include <utility>

#include <tuple>

#include <cmath>

namespace figs {

const double PI = 3.14159265;

template<typename T>

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

template<typename T>

struct Triangle {

vertex<T> vertices[3];

};

template<typename T>

struct Square {

vertex<T> vertices[4];

};

template<typename T>

struct Rectangle {

vertex<T> vertices[4];

};

template<typename T, template<typename> typename F>

void Read(F<T> &f) {

double alpha = 0;

int k = 0;

if (std::is\_same<F<T>, Square<T>>::value) {

alpha = 90.0 / 180.0 \* PI;

k = 4;

} else if (std::is\_same<F<T>, Triangle<T>>::value) {

alpha = 120.0 / 180.0 \* PI;

k = 3;

} else if (std::is\_same<F<T>, Rectangle<T>>::value) {

alpha = 90.0 / 180.0 \* PI;

k = 4;

} else {

std::cout << "Bad figure!";

}

vertex<double> center;

std::cin >> center.first >> center.second >> f.vertices[0].first >> f.vertices[0].second;

for (int i = 1; i < k; ++i) {

double a = -sin(alpha) \* (static\_cast<double>(f.vertices[i - 1].second) - center.second)

+ cos(alpha) \* (static\_cast<double>(f.vertices[i - 1].first) - center.first) + center.first;

double b = cos(alpha) \* (static\_cast<double>(f.vertices[i - 1].second) - center.second)

+ sin(alpha) \* (static\_cast<double>(f.vertices[i - 1].first) - center.first) + center.second;

if constexpr (std::is\_same<T, int>::value) {

f.vertices[i].first = static\_cast<T>(std::round(a));

f.vertices[i].second = static\_cast<T>(std::round(b));

} else {

f.vertices[i].first = a;

f.vertices[i].second = b;

}

}

}

template<typename T>

T dist(const vertex<T> &a, const vertex<T> &b) {

return static\_cast<T>(sqrt(

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

}

template<typename T, template<typename> typename F>

typename std::enable\_if<std::is\_same<F<T>, Rectangle<T>>::value>::type PrintFigure(F<T> &a) {

std::cout << "Rectangle: ";

for (int i = 0; i < 4; ++i) {

std::cout << '[' << a.vertices[i].first << ',' << a.vertices[i].second << ']' << ' ';

}

std::cout << std::endl;

}

template<typename T, template<typename> typename F>

typename std::enable\_if<std::is\_same<F<T>, Square<T>>::value>::type PrintFigure(F<T> &a) {

std::cout << "Square: ";

for (int i = 0; i < 4; ++i) {

std::cout << '[' << a.vertices[i].first << ',' << a.vertices[i].second << ']' << ' ';

}

std::cout << std::endl;

}

template<typename T, template<typename> typename F>

typename std::enable\_if<std::is\_same<F<T>, Triangle<T>>::value>::type PrintFigure(F<T> &a) {

std::cout << "Triangle: ";

for (int i = 0; i < 3; ++i) {

std::cout << '[' << a.vertices[i].first << ',' << a.vertices[i].second << ']' << ' ';

}

std::cout << std::endl;

}

template<typename T>

void PrintFigure(T &a) {

std::cout << "Bad Figure!\n";

};

template<typename T, template<typename> typename F>

typename std::enable\_if<std::is\_same<F<T>, Rectangle<T>>::value, double>::type GetFigureArea(F<T> &a) {

return dist(a.vertices[0], a.vertices[1]) \* dist(a.vertices[2], a.vertices[3]);

}

template<typename T, template<typename> typename F>

typename std::enable\_if<std::is\_same<F<T>, Square<T>>::value, double>::type GetFigureArea(F<T> &a) {

return dist(a.vertices[0], a.vertices[1]) \* dist(a.vertices[2], a.vertices[3]);

}

template<typename T, template<typename> typename F>

typename std::enable\_if<std::is\_same<F<T>, Triangle<T>>::value, double>::type GetFigureArea(F<T> &a) {

T A = dist(a.vertices[0], a.vertices[1]);

T B = dist(a.vertices[1], a.vertices[2]);

T C = dist(a.vertices[2], a.vertices[0]);

double p = (A + B + C) / 2.0;

double x = p \* (p - A) \* (p - B) \* (p - C);

return sqrt(x);

}

template<typename T>

double GetFigureArea(T &a) {

*//std::cout << "Bad Figure: ";*

return -1;

}

template<typename F, size\_t *index*>

typename std::enable\_if<*index* < 0, double>::type ProcessTuple(F &t) {

return 0;

}

template<typename F, size\_t *index*>

typename std::enable\_if<*index* >= std::tuple\_size<F>::value, double>::type ProcessTuple(F &t) {

return 0;

}

template<typename F, size\_t *index* = 0>

typename std::enable\_if<*index* < std::tuple\_size<F>::value, double>::type ProcessTuple(F &t) {

auto v = std::get<*index*>(t);

PrintFigure(v);

double area = GetFigureArea(v);

if (area >= 0) {

std::cout << "\tFigure Area: " << area << '\n';

} else {

std::cout << "\tError!\n";

}

return area + ProcessTuple<F, *index* + 1>(t);

}

}

int main() {

bool f = true;

while (f) {

f = false;

std::cout << "1. Целочисленные координаты\n2. Вещественные координаты" << std::endl;

int option;

std::cin >> option;

if (option == 1) {

figs::Square<int> a;

figs::Triangle<int> b;

figs::Rectangle<int> c;

std::cout << "Введите координаты для квадрата (центр и вершина): \n";

Read(a);

std::cout << "Введите координаты для треугольника (центр и вершина): \n";

Read(b);

std::cout << "Введите координаты для прямоугольника (центр и вершина): \n";

Read(c);

auto t = std::make\_tuple(a, b, c);

double area = figs::ProcessTuple(t);

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

} else if (option == 2) {

figs::Square<double> a;

figs::Triangle<double> b;

figs::Rectangle<double> c;

std::cout << "Введите координаты для квадрата (центр и вершина): \n";

Read(a);

std::cout << "Введите координаты для треугольника (центр и вершина): \n";

Read(b);

std::cout << "Введите координаты для прямоугольника (центр и вершина): \n";

Read(c);

auto t = std::make\_tuple(a, b, c);

double area = figs::ProcessTuple(t);

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

} else {

std::cout << "Goodbye!" << std::endl;

break;

}

if (!std::cin.fail()) f = true;

*//std::tuple<figs::Square<int>, figs::Triangle<int>, figs::Rectangle<int>> t;*

}

return 0;

}

1. Выводы

С помощью шаблонов и метапрограммирования мы можем создавать более эффективные программы на языке C++, во многом благодаря тому что шаблон - это указание по генерации кода для компилятора, то есть мы сбавляем нагрузку на вычислительную машину за счет того что упрощаем работу программы во время выполнения. Однако, код, содержащий много шаблонов и языковых конструкций для работы с типами, становится трудночитаемым и трудно понимаемым. Также увеличивается время, затраченное на сборку программы (компилятору приходится проводить сложный анализ шаблонов).

Список литературы

1. C++ Primer (5th Edition) Stanley B. Lippman, Josee Lajoie, E. Moo. – Addison-Wesley, 2019. ISBN-13: 978-0321714114, ISBN-10: 9780321714114.

2. Effective Modern C++, Scott Meyers. ISBN-13: 978-1491903995

ISBN-10: 9781491903995