**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: Киреев Александр Константинович

Группа: 80-206

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 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**

В файле lab4.cpp представлен интерактивный ввод кортежа, обработка невалидных данных. В файлах octagon.hpp, triangle.hpp, square.hpp задаются шаблонные классы фигур, а также несколько конструкторов и проверка фигуры на существование в данных координатах. В файле tuple\_handler.hpp представлены шаблонные функции printTuple и getTotalArea, которые используют инструмент if constexpr для отсечения ненужного кода во время компиляции, decltype для определения типа шаблона. Функции printTuple и getTotalArea работают на принципе рекурсии: getTotalArea вычисляет площадь очередной фигуры в шаблоне при помощи шаблонный функции getArea, прибавляет площадь к общему результату и запускает getTotalArea для следующей фигуры; printTuple, пока индекс меньше размера кортежа, выводит фигуру в кортеже при помощи шаблонной функции print и запускает printTuple от следующей фигуры в кортеже.

1. **Набор тестов**

Тест 1:

0 0 2 2

0 0 2 2

2 2 4 4

0 0 0 1

0 0 0 1

0 0 0 1

1

2

4

Тест 1 проверяет работоспособность программы в целом, а также показывает, что в целых координатах не существует правильного треугольника с центром в (0, 0) и с вершиной в (0, 1).

Тест 2:

0 0 1 1

0 0 2.5 1

0 0 2 2

0 0 0 2.5

0 0 0 1

0 0 0 2.3

1

2

4

В тесте 2 мы показываем, что не существует правильного квадрата с координатами типа size\_t с центром в точке (0, 0) и с вершиной в (2, 2), а также показывает, что в целых координатах не существует правильного треугольника с центром в (0, 0) и с вершиной в (0, 1).

1. **Результаты выполнения тестов**

Тест 1:

MacBook-Air-K:oop\_exercise\_04 AK$ ./lab4 < test/test1.txt

Command -- Description

1 -- Display figures in tuple

2 -- Calculate total area of figures in tuple

3 -- Display help

4 -- End program

Enter center and vertex of Square<int>:

Enter center and vertex of Square<double>:

Enter center and vertex of Square<size\_t>:

Enter center and vertex of Triangle<double>:

Enter center and vertex of Triangle<int>:

Enter center and vertex of Octagon<double>:

Cmd: =============================TUPLE\_START=============================

[ (2, 2) (-2, 2) (-2, -2) (2, -2) ]

[ (2.000, 2.000) (-2.000, 2.000) (-2.000, -2.000) (2.000, -2.000) ]

[ (4, 4) (0, 4) (0, 0) (4, 0) ]

[ (0.000, 1.000) (-0.866, -0.500) (0.866, -0.500) ]

Invalid figure: Triangle<i>

[ (0.000, 1.000) (-0.707, 0.707) (-1.000, 0.000) (-0.707, -0.707) (-0.000, -1.000) (0.707, -0.707) (1.000, -0.000) (0.707, 0.707) ]

==============================TUPLE\_END==============================

Cmd: Total tuple area is: 52.127

Тест 2:

MacBook-Air-K:oop\_exercise\_04 AK$ ./lab4 < test/test2.txt

Command -- Description

1 -- Display figures in tuple

2 -- Calculate total area of figures in tuple

3 -- Display help

4 -- End program

Enter center and vertex of Square<int>:

Enter center and vertex of Square<double>:

Enter center and vertex of Square<size\_t>:

Enter center and vertex of Triangle<double>:

Enter center and vertex of Triangle<int>:

Enter center and vertex of Octagon<double>:

Cmd: =============================TUPLE\_START=============================

[ (1, 1) (-1, 1) (-1, -1) (1, -1) ]

[ (2.500, 1.000) (-1.000, 2.500) (-2.500, -1.000) (1.000, -2.500) ]

Invalid figure: Square<m>

[ (0.000, 2.500) (-2.165, -1.250) (2.165, -1.250) ]

Invalid figure: Triangle<i>

[ (0.000, 2.300) (-1.626, 1.626) (-2.300, 0.000) (-1.626, -1.626) (-0.000, -2.300) (1.626, -1.626) (2.300, -0.000) (1.626, 1.626) ]

==============================TUPLE\_END==============================

Cmd: Total tuple area is: 41.581

1. **Листинг программы**

**lab4.cpp**

// Киреев А.К. 206

#include <iostream>

#include <utility>

#include <tuple>

#include <sstream>

#include <string>

#include "octagon.hpp"

#include "square.hpp"

#include "triangle.hpp"

#include "tuple\_handler.hpp"

//

// мануал

//

void help() {

std::cout << "Command -- Description\n" <<

"1 -- Display figures in tuple\n" <<

"2 -- Calculate total area of figures in tuple \n" <<

"3 -- Display help\n" <<

"4 -- End program" << std::endl;

}

//

// интерактивный ввод

//

std::tuple<Square<int>, Square<double>, Square<size\_t>, Triangle<double>, Triangle<int>, Octagon<double>>

tupleCreat() {

std::cout << "Enter center and vertex of Square<int>:\n";

int c1\_x, v1\_x, c1\_y, v1\_y;

if (!(std::cin >> c1\_x >> c1\_y >> v1\_x >> v1\_y)) {

std::cout << "Input error.\n";

}

std::cout << "Enter center and vertex of Square<double>:\n";

double c2\_x, v2\_x, c2\_y, v2\_y;

if (!(std::cin >> c2\_x >> c2\_y >> v2\_x >> v2\_y)) {

std::cout << "Input error.\n";

}

std::cout << "Enter center and vertex of Square<size\_t>:\n";

size\_t c3\_x, v3\_x, c3\_y, v3\_y;

if (!(std::cin >> c3\_x >> c3\_y >> v3\_x >> v3\_y)) {

std::cout << "Input error.\n";

}

std::cout << "Enter center and vertex of Triangle<double>:\n";

double c4\_x, v4\_x, c4\_y, v4\_y;

if (!(std::cin >> c4\_x >> c4\_y >> v4\_x >> v4\_y)) {

std::cout << "Input error.\n";

}

std::cout << "Enter center and vertex of Triangle<int>:\n";

int c5\_x, v5\_x, c5\_y, v5\_y;

if (!(std::cin >> c5\_x >> c5\_y >> v5\_x >> v5\_y)) {

std::cout << "Input error.\n";

}

std::cout << "Enter center and vertex of Octagon<double>:\n";

double c6\_x, v6\_x, c6\_y, v6\_y;

if (!(std::cin >> c6\_x >> c6\_y >> v6\_x >> v6\_y)) {

std::cout << "Input error.\n";

}

std::tuple<Square<int>, Square<double>, Square<size\_t>, Triangle<double>, Triangle<int>, Octagon<double>> t =

{Square<int>({c1\_x, c1\_y}, {v1\_x, v1\_y}), Square<double>({c2\_x, c2\_y}, {v2\_x, v2\_y}),

Square<size\_t>({c3\_x, c3\_y}, {v3\_x, v3\_y}), Triangle<double>({c4\_x, c4\_y}, {v4\_x, v4\_y}),

Triangle<int>({c5\_x, c5\_y}, {v5\_x, v5\_y}), Octagon<double>({c6\_x, c6\_y}, {v6\_x, v6\_y})};

return t;

}

void mainLoop() {

int command = 0;

std::string s;

// создаем tuple из шести фигур разного типа

std::tuple<Square<int>, Square<double>, Square<size\_t>, Triangle<double>, Triangle<int>, Octagon<double>> t = tupleCreat();

// парсер команд

while (std::cout << "Cmd: " && std::cin >> s) {

if (s.length() > 1) {

std::cout << "Invalid command." << std::endl;

continue;

}

std::stringstream ss(s);

ss >> command;

if (command == 4)

break;

switch (command) {

case 1:

std::cout << "=============================TUPLE\_START=============================\n";

printTuple<decltype(t),0>(t);

std::cout << "==============================TUPLE\_END==============================\n";

break;

case 2:

std::cout << "Total tuple area is: " << getTotalArea<decltype(t),0>(t) << std::endl;

break;

case 3:

help();

break;

default:

std::cout << "Invalid command!" << std::endl;

break;

}

}

}

int main() {

help();

mainLoop();

return 0;

}

**octagon.hpp**

// Киреев А.К. 206

#pragma once

#include <utility>

#include <iostream>

#include <cmath>

#include <vector>

#define \_USE\_MATH\_DEFINES

#define EPSILON 0.01

template <class T>

struct Octagon {

using type = T;

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

vertex center;

vertex v;

bool valid;

std::vector<vertex> path;

Octagon();

Octagon(vertex center, vertex v);

Octagon(std::istream& is);

~Octagon() = default;

};

template <class T>

Octagon<T>::Octagon(): center(), v(), valid(false) {}

template <class T>

Octagon<T>::Octagon(vertex center, vertex v): center(center), v(v) {

valid = true;

validCheck(\*this);

}

template <class T>

Octagon<T>:: Octagon(std::istream& in) {

T center\_X, center\_Y, v\_X, v\_Y;

in >> center\_X >> center\_Y >> v\_X >> v\_Y;

center = std::make\_pair(center\_X, center\_Y);

v = std::make\_pair(v\_X, v\_Y);

// valid check

valid = true;

validCheck(\*this);

}

template <class T>

void validCheck(Octagon<T>& f) {

double r = sqrt((f.v.first - f.center.first) \* (f.v.first - f.center.first) +

(f.v.second - f.center.second) \* (f.v.second - f.center.second));

double lenX = f.v.first - f.center.first;

double lenY = f.v.second - f.center.second;

double phi = acos(lenX / r);

if (lenX < 0) {

if (lenY < 0)

phi = -M\_PI + phi;

} else {

if (lenY < 0)

phi = -phi;

}

for (int i = 0; i < 8; ++i) {

double angle = 2.0 \* M\_PI \* i / 8.0;

double next\_X = r \* cos(angle + phi) + f.center.first;

double next\_Y = r \* sin(angle + phi) + f.center.second;

if (abs(next\_X - static\_cast<T>(next\_X)) > EPSILON || abs(next\_Y - static\_cast<T>(next\_Y)) > EPSILON) {

f.valid = false;

}

f.path.push\_back(std::make\_pair(static\_cast<T>(next\_X),static\_cast<T>(next\_Y)));

}

}

**triangle.hpp**

// Киреев А.К. 206

#pragma once

#include <utility>

#include <iostream>

#include <cmath>

#define \_USE\_MATH\_DEFINES

#define EPSILON 0.01

template <class T>

struct Triangle {

using type = T;

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

vertex center;

vertex v;

std::vector<vertex> path;

bool valid;

Triangle();

Triangle(vertex center, vertex v);

Triangle(std::istream& is);

~Triangle() = default;

};

template <class T>

Triangle<T>::Triangle(): center(), v(), valid(false) {}

template <class T>

Triangle<T>::Triangle(vertex center, vertex v): center(center), v(v) {

valid = true;

validCheck(\*this);

}

template <class T>

Triangle<T>:: Triangle(std::istream& in) {

T center\_X, center\_Y, v\_X, v\_Y;

in >> center\_X >> center\_Y >> v\_X >> v\_Y;

center = std::make\_pair(center\_X, center\_Y);

v = std::make\_pair(v\_X, v\_Y);

// valid check

valid = true;

validCheck(\*this);

}

template <class T>

void validCheck(Triangle<T>& f) {

double r = sqrt((f.v.first - f.center.first) \* (f.v.first - f.center.first) +

(f.v.second - f.center.second) \* (f.v.second - f.center.second));

double lenX = f.v.first - f.center.first;

double lenY = f.v.second - f.center.second;

double phi = acos(lenX / r);

if (lenX < 0) {

if (lenY < 0)

phi = -M\_PI + phi;

} else {

if (lenY < 0)

phi = -phi;

}

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

double angle = 2.0 \* M\_PI \* i / 3.0;

double next\_X = r \* cos(angle + phi) + f.center.first;

double next\_Y = r \* sin(angle + phi) + f.center.second;

if (abs(next\_X - static\_cast<T>(next\_X)) > EPSILON || abs(next\_Y - static\_cast<T>(next\_Y)) > EPSILON) {

f.valid = false;

}

f.path.push\_back(std::make\_pair(static\_cast<T>(next\_X), static\_cast<T>(next\_Y)));

}

}

**square.hpp**

// Киреев А.К. 206

#pragma once

#include <utility>

#include <iostream>

#include <cmath>

#define \_USE\_MATH\_DEFINES

#define EPSILON 0.01

template <class T>

struct Square {

using type = T;

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

vertex center;

vertex v;

std::vector<vertex> path;

bool valid;

Square();

Square(vertex center, vertex v);

Square(std::istream& is);

~Square() = default;

};

template <class T>

Square<T>::Square(): center(), v(), valid(false) {}

template <class T>

Square<T>::Square(vertex center, vertex v): center(center), v(v) {

valid = true;

validCheck(\*this) ;

}

template <class T>

Square<T>:: Square(std::istream& in) {

T center\_X, center\_Y, v\_X, v\_Y;

in >> center\_X >> center\_Y >> v\_X >> v\_Y;

center = std::make\_pair(center\_X, center\_Y);

v = std::make\_pair(v\_X, v\_Y);

// valid check

valid = true;

validCheck(\*this) ;

}

template <class T>

void validCheck(Square<T>& f) {

double r = sqrt((f.v.first - f.center.first) \* (f.v.first - f.center.first) +

(f.v.second - f.center.second) \* (f.v.second - f.center.second));

double lenX = f.v.first - f.center.first;

double lenY = f.v.second - f.center.second;

double phi = acos(lenX / r);

if (lenX < 0) {

if (lenY < 0)

phi = -M\_PI + phi;

} else {

if (lenY < 0)

phi = -phi;

}

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

double angle = 2.0 \* M\_PI \* i / 4.0;

double next\_X = r \* cos(angle + phi) + f.center.first;

double next\_Y = r \* sin(angle + phi) + f.center.second;

if (abs(next\_X - static\_cast<T>(next\_X)) > EPSILON || abs(next\_Y - static\_cast<T>(next\_Y)) > EPSILON) {

f.valid = false;

}

f.path.push\_back(std::make\_pair(static\_cast<T>(next\_X), static\_cast<T>(next\_Y)));

}

}

**tuple\_handler.hpp**

// Киреев А.К. 206

#include <iostream>

#include <tuple>

#include <utility>

#include <cmath>

#include <string>

#include <typeinfo>

#define \_USE\_MATH\_DEFINES

#include "octagon.hpp"

#include "square.hpp"

#include "triangle.hpp"

template <class T>

double getArea(T& figure) {

using Type = typename T::type;

double res = 0.0;

if (!figure.valid) {

return 0.0;

}

double r = sqrt((figure.v.first - figure.center.first) \* (figure.v.first - figure.center.first) + (figure.v.second - figure.center.second) \* (figure.v.second - figure.center.second));

if constexpr (std::is\_same<decltype(figure), Triangle<Type>&>::value) {

res += 3.0 \* sqrt(3.0) \* r \* r / 4.0;

} else if constexpr (std::is\_same<decltype(figure), Square<Type>&>::value) {

res += 2.0 \* r \* r;

} else if constexpr (std::is\_same<decltype(figure), Octagon<Type>&>::value) {

res += 2.0 \* sqrt(2.0) \* r \* r;

} else {

return 0.0;

}

return res;

}

template <class T, int index>

double getTotalArea(T& tuple) {

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

double res = getArea(figure);

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

return res + getTotalArea<T, index + 1>(tuple);

}

return res;

}

template <class T>

void print(T& figure) {

using Type = typename T::type;

// проверка фигуры на валидность (она может не существовать в каких-то координатах)

if (!figure.valid) {

if constexpr (std::is\_same<decltype(figure), Triangle<Type>&>::value) {

std::cout << "Invalid figure: Triangle<" << typeid(Type).name() << ">" << std::endl;

} else if constexpr (std::is\_same<decltype(figure), Square<Type>&>::value) {

std::cout << "Invalid figure: Square<" << typeid(Type).name() << ">" << std::endl;

} else if constexpr (std::is\_same<decltype(figure), Octagon<Type>&>::value){

std::cout << "Invalid figure: Octagon<" << typeid(Type).name() << ">" << std::endl;

} else {

std::cout << "Invalid undefined figure." << std::endl;

}

return;

} // конец проверки

// вывод координат вершин фигуры

std::cout.precision(3);

std::cout << "[ ";

if constexpr

((std::is\_same<decltype(figure), Triangle<Type>&>::value) ||

(std::is\_same<decltype(figure), Square<Type>&>::value) ||

(std::is\_same<decltype(figure), Octagon<Type>&>::value)) {

for (int i = 0; i < figure.path.size(); ++i) {

std::cout << "(" << std::fixed << figure.path[i].first << ", " << std::fixed << figure.path[i].second << ") ";

}

}

std::cout << "]" << std::endl; // конец вывода координат вершин фигуры

}

template <class T,size\_t index>

typename std::enable\_if<index >= std::tuple\_size<T>::value, void>::type

printTuple(T& tuple){

std::cout << std::endl;

}

template <class T,size\_t index>

typename std::enable\_if<index < std::tuple\_size<T>::value, void>::type

printTuple(T& tuple){

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

print(figure);

printTuple<T, index + 1>(tuple);

}

1. **Вывод**

Метапрограммирование хорошо развито в языке C++. С его помощью можно перенести некоторые вещи из runtime в compile time, повысив эффективность программы. Также метапрограммирование - сильнейший инструмент абстракции над данными, который позволяет писать максимально обобщенный код. С помощью шаблонов можно изменять поведение программы в зависимости от шаблонного типа. Однако метапрограммирование сложно воспринимается в коде, его сложно поддерживать и развивать такой код. Поэтому нужно пользоваться метапрограммированием с умом.

**Список литературы**

1. “Язык программирования C++. Краткий курс”, Бьярне Страуструп, 2-е издание, (перевод Игоря Красикова), бумага офсетная-белая, твердый переплет, 320 стр., ISBN 978-5-907144-12-5.