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: Шубин Григорий Сергеевич
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

a. Ознакомиться с теоретическим материалом по шаблонам и метапрограммированию в языке C++.
b. Разработать шаблоны классов “Rhombus”, “Pentagon”, “Hexagon” для работы с ромбами, пятиугольниками и шестиугольниками соответственно. 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. Все фигуры являются правильными фигурами. Для хранения координат фигур необходимо использовать шаблон std::pair.
c. Необходимо реализовать две шаблонных функции:
    i. Функция print для 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).
    ii. Функция square вычисления суммарной площади фигур. Функция должна принимать на вход std::tuple с фигурами, согласно варианту задания (минимум по одной каждого класса).
d. Создать программу, которая 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.
e. Настроить CMake файл для сборки программы.
f. Подготовить наборы тестовых данных.
g. Загрузить файлы лабораторной работы в репозиторий GitHub.
h. Подготовить отчёт по лабораторной работе.

2. Описание программы

Программа имеет многофайловую структуру. Описания классов выделены в отдельные заголовочные файлы.

Rhombus.h

В файле Rhombus.h описан шаблон класса Rhombus для работы с ромбами. Параметр шаблона - тип данных для координат. Ромб задаётся координатами центра - точки пересечения диагоналей и длинами диагоналей. Диагонали ромба параллельны осям координат: первая диагональ параллельна оси абсцисс, вторая - оси ординат.

Pentagon.h

В файле Pentagon.h описан шаблон класса Pentagon для работы с пятиугольниками. Параметр шаблона - тип данных для координат. Пятиугольник задаётся координатами центра и радиусом. Одна из сторон пятиугольника параллельна оси абсцисс.

Hexagon.h

В файле Hexagon.h описан шаблон класса Hexagon для работы с шестиугольниками. Параметр шаблона - тип данных для координат. Шестиугольник задаётся координатами центра и радиусом (равному длине стороны). Две стороны шестиугольника параллельны оси абсцисс.

main.cpp

В начале функции main описываются фигуры, которые будут обрабатываться программой. Для хранения фигур используется std::tuple.

Для печати фигур на экран используется шаблонная рекурсивная функция print_tuple, принимающая в качестве аргумента tuple с фигурами. При помощи шаблона std::enable_if проверяется, лежит ли данный индекс в кортеже или нет. Если лежит, то вызывается шаблонная функция print, вычисляющая и выводящая на экран координаты фигуры, и рекурсивно вызывается print_tuple с инкрементированным индексом. Если индекс лежит за пределами кортежа, то печатается перевод строки.

Для каждой фигуры реализованы функции print. Компилятор определяет нужную функцию по наличию специальных полей в классах фигур: если у фигуры есть поле diag1, то это ромб, если есть radius - пятиугольник, если есть side - шестиугольник.

Аналогичным образом реализована рекурсивная шаблонная функция total_square, вычисляющая общую площадь фигур, и функции square, вычисляющие площади определенных фигур. Для ромба площадь считается как полусумма длин диагоналей, для остальных фигур - как площадь правильного n-угольника.

В функции main последовательно вызываются функции print_tuple и total_square.

3. Тестирование программы

Суть лабораторной работы в том, что большая часть вычислений производится на стадии компиляции, поэтому интерактивный ввод фигур не предусмотрен. В функции main описаны 6 фигур: 2 ромба, 2 пятиугольника и 2 шестиугольника. Для тестирования программы будем менять характеристики этих фигур в программе.

Тест 1

```
Rhombus<int> r1;
r1.center = {2, 1};
r1.diag1 = 10;
r1.diag2 = 6;

Rhombus<double> r2;
r2.center = {0.5, -3.2};
r2.diag1 = 3.14;
r2.diag2 = 2.72;
```

```
Pentagon<int> p1;
p1.center = {1, 0};
p1.radius = 1;

Pentagon<double> p2;
p2.center = {1.55, 4.1};
p2.radius = 9.5;

Hexagon<int> h1;
h1.center = {1, -3};
h1.side = 1;

Hexagon<double> h2;
h2.center = {13.37, -3.5};
h2.side = 6.66;
```

Результаты теста 1

```
Rhombus {(7; 1), (2; 4), (-3; 1), (2; -2)}
Pentagon {(2; 0.31), (1; 1), (0.049; 0.31), (0.41; -0.81), (1.6; -0.81)}
Hexagon {(2; -3), (1.5; -2.1), (0.5; -2.1), (0; -3), (0.5; -3.9), (1.5;
-3.9)}
Rhombus {(2.1; -3.2), (0.5; -1.8), (-1.1; -3.2), (0.5; -4.6)}
Pentagon {(11; 7), (1.6; 14), (-7.5; 7), (-4; -3.6), (7.1; -3.6)}
Hexagon {(20; -3.5), (17; 2.3), (10; 2.3), (6.7; -3.5), (10; -9.3), (17;
-9.3)}

Total square: 345.73
```

Тест 2

```
Rhombus<int> r1;
r1.center = {3, 5};
r1.diag1 = 100;
r1.diag2 = 1;

Rhombus<double> r2;
r2.center = {13.37, -36.2};
r2.diag1 = 10;
r2.diag2 = 2.74;

Pentagon<int> p1;
p1.center = {9, -9};
p1.radius = 9;

Pentagon<double> p2;
p2.center = {0.01, 0.01};
p2.radius = 0.25;

Hexagon<int> h1;
h1.center = {-5, -10};
h1.side = 7.5;
```

```
Hexagon<double> h2;
h2.center = {12.34, -5.3};
h2.side = 0.01;
```

Результаты теста 2

```
Rhombus {(53; 5), (3; 5.5), (-47; 5), (3; 4.5)}
Pentagon {(18; -6.2), (9; 0), (0.44; -6.2), (3.7; -16), (14; -16)}
Hexagon {(2.5; -10), (-1.2; -3.5), (-8.7; -3.5), (-12; -10), (-8.8; -16),
(-1.2; -16)}
Rhombus {(18; -36), (13; -35), (8.4; -36), (13; -38)}
Pentagon {(0.25; 0.087), (0.01; 0.26), (-0.23; 0.087), (-0.14; -0.19), (0.16;
-0.19)}
Hexagon {(12; -5.3), (12; -5.3), (12; -5.3), (12; -5.3), (12; -5.3), (12;
-5.3)}

Total square: 395.75
```

4. Листинг программы

Rhombus.h

```
#ifndef OOP_EXERCISE_04_RHOMBUS_H
#define OOP_EXERCISE_04_RHOMBUS_H

template<class T>
class Rhombus {
public:
    std::pair<T, T> center;
    double diag1;
    double diag2;
};


#endif //OOP_EXERCISE_04_RHOMBUS_H
```

Pentagon.h

```
#ifndef OOP_EXERCISE_04_PENTAGON_H
#define OOP_EXERCISE_04_PENTAGON_H

template<class T>
class Pentagon {
public:
    std::pair<T, T> center;
    double radius;
};


#endif //OOP_EXERCISE_04_PENTAGON_H
```

Hexagon.h

```
#ifndef OOP_EXERCISE_04_HEXAGON_H
#define OOP_EXERCISE_04_HEXAGON_H

template<class T>
class Hexagon {
public:
    std::pair<T, T> center;
    double side;
};


#endif //OOP_EXERCISE_04_HEXAGON_H
```

main.cpp

```
#include <iostream>
#include <tuple>
#include <cmath>

#include "Rhombus.h"
#include "Pentagon.h"
#include "Hexagon.h"


// prints a tuple
template<class T, int index>
typename std::enable_if<index >= std::tuple_size<T>::value, void>::type
print_tuple(T &tuple) {
    std::cout << std::endl;
}

template<class T, int 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);
}


// prints a rhombus
template<class T>
typename std::enable_if<(sizeof(T::diag1) > 0), void>::type print(T &r) {
    std::cout.precision(2);
    std::cout << "Rhombus {(" << r.center.first + r.diag1 * 0.5 << "; " <<
r.center.second << "), (";
    std::cout << r.center.first << "; " << r.center.second + r.diag2 * 0.5 << "),
(";
    std::cout << r.center.first - r.diag1 * 0.5 << "; " << r.center.second << "),
(";
    std::cout << r.center.first << "; " << r.center.second - r.diag2 * 0.5 << ")}";
    std::cout << std::endl;
}

// prints a pentagon
template<class T>
typename std::enable_if<(sizeof(T::radius) > 0), void>::type print(T &p) {
    std::cout << "Pentagon {";
    double pi = acos(-1);
    for (int i = 0; i < 5; ++i) {
        double angle = 2 * pi * i / 5;
        std::cout.precision(2);
        std::cout << "(" << p.center.first + p.radius * cos(angle + pi / 10) << ";
```

```
"
                  << p.center.second + p.radius * sin(angle + pi / 10) << ")";
        if (i != 4) {
            std::cout << ", ";
        }
    }
    std::cout << "}" << std::endl;
}

// prints a hexagon
template<class T>
typename std::enable_if<(sizeof(T::side) > 0), void>::type print(T &h) {
    std::cout << "Hexagon {";
    double pi = acos(-1);
    for (int i = 0; i < 6; ++i) {
        double angle = pi * i / 3;
        std::cout.precision(2);
        std::cout << "(" << h.center.first + h.side * cos(angle) << "; "
                  << h.center.second + h.side * sin(angle) << ")";
        if (i != 5) {
            std::cout << ", ";
        }
    }
    std::cout << "}" << std::endl;
}


// counts total square of figures in tuple
template<class T, int index>
typename std::enable_if<index >= std::tuple_size<T>::value, double>::type
total_square(T &tuple) {
    return 0;
}

template<class T, int index>
typename std::enable_if<index < std::tuple_size<T>::value, double>::type
total_square(T &tuple) {
    auto figure = std::get<index>(tuple);
    double cur_square = square(figure);
    return cur_square + total_square<T, index + 1>(tuple);
}

// counts a square of rhombus
template<class T>
typename std::enable_if<(sizeof(T::diag1) > 0), double>::type square(T &r) {
    return (r.diag1 + r.diag2) * 0.5;
}

// counts a square of pentagon
template<class T>
typename std::enable_if<(sizeof(T::radius) > 0), double>::type square(T &p) {
    double pi = acos(-1);
    double side = p.radius * cos(13 * pi / 10) - p.radius * cos(17 * pi / 10);
    return sqrt(25 + 10 * sqrt(5)) * pow(side, 2) * 0.25;
}

// counts a square of hexagon
template<class T>
typename std::enable_if<(sizeof(T::side) > 0), double>::type square(T &h) {
    return pow(h.side, 2) * 3 * sqrt(3) * 0.5;
}

int main() {
    // creating objects with figures
    Rhombus<int> r1;
    r1.center = {3, 5};
```

```
    r1.diag1 = 100;
    r1.diag2 = 1;

    Rhombus<double> r2;
    r2.center = {13.37, -36.2};
    r2.diag1 = 10;
    r2.diag2 = 2.74;

    Pentagon<int> p1;
    p1.center = {9, -9};
    p1.radius = 9;

    Pentagon<double> p2;
    p2.center = {0.01, 0.01};
    p2.radius = 0.25;

    Hexagon<int> h1;
    h1.center = {-5, -10};
    h1.side = 7.5;

    Hexagon<double> h2;
    h2.center = {12.34, -5.3};
    h2.side = 0.01;


    // creating tuple
    std::tuple<decltype(r1), decltype(p1), decltype(h1), decltype(r2),
decltype(p2), decltype(h2)>
            tuple{r1, p1, h1, r2, p2, h2};

    print_tuple<decltype(tuple), 0>(tuple);
    std::cout << std::fixed << "Total square: " << total_square<decltype(tuple),
0>(tuple);
}
```

5. Выводы

В данной лабораторной работе были изучены шаблоны и метафункции в языке C++. При помощи шаблонов и метапрограммирования можно добиться того, чтобы компилятор сам генерировал код, необходимый для некоторых функций и классов. В стандартной библиотеке имеется много шаблонных структур, которые упрощают метапрограммирование.

В данной лабораторной работе была применена техника SFINAE - Substitution Failure Is Not An Error. Она основывается на том, что компилятор, пытаясь вывести тип параметра для параметра шаблона, встречая ошибку в конкретной специализации, не выдает ошибку пользователю, а анализирует все возможные варианты.

Список используемых источников

1. Руководство по языку C++ [Электронный ресурс]. URL: https://www.cplusplus.com/ (дата обращения 28.10.2020).
2. Шаблоны классов в C++ [Электронный ресурс]. URL: http://cppstudio.com/post/5188/ (дата обращения 29.10.2020).