Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 4: «Основы метапрограммирования»

| | |
|---|---|
| Группа: | М8О-206Б-18, №14 |
| Студент: | Орозбакиев Э.Д. |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: | |
| Дата: | |

Москва, 2019

## 1. Задание

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться oop_exercise_04 (в случае использования Windows oop_exercise_04.exe)

Репозиторий должен содержать файлы:

• main.cpp //файл с заданием работы

• CMakeLists.txt // файл с конфигураций CMake

• test_xx.txt // файл с тестовыми данными. Где xx – номер тестового набора 01, 02 , … Тестовых наборов

• report.doc // отчет о лабораторной работе

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип

данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются

Создать набор шаблонов, создающих функции, реализующие:

1. Вычисление геометрического центра фигуры;

2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;

3. Вычисление площади фигуры;

Параметром шаблона должен являться тип класса фигуры ( например Square<int>). Помимо самого класса фигуры, шаблонная функция должна уметь работать с tuple. Например, std::tuple<std::pair<int,int>, std::pair<int,int>, std::pair<int,int>> должен интерпретироваться как треугольник. std::tuple<std::pair<int,int>, std::pair<int,int>, std::pair<int,int>, std::pair<int,int>> - как квадрат. Каждый std::pair<int,int> - соответствует координатам вершины фигуры вращения.

Создать программу, которая позволяет:

• Вводить из стандартного ввода std::cin фигуры, согласно варианту задания (как в виде класса, так и в виде

std::tuple).

• Вызывать для нее шаблонные функции (1-3).

При реализации шаблонных функций допускается использование вспомогательных шаблонов std::enable_if, std::tuple_size, std::is_same.

## 2. Адрес репозитория на GitHub

https://github.com/p0kemo4ik/oop_exercise_04

## 3. Код программы на C++

*main.cpp*

```
#include <iostream>
#include <tuple>
#include "vertex.hpp"
#include "hexagon.hpp"
#include "octagon.hpp"
#include "pentagon.hpp"
#include "templates.hpp"
template<class T>
void process() {
    T object;
    read(std::cin, object);
    print(std::cout, object);
    std::cout << square(object) << std::endl;
    std::cout << center(object) << std::endl;
}
int main() {
    std::cout << "How you want to declare your figure: " << std::endl;
    std::cout << "1. Tuple" << std::endl;
    std::cout << "2. Class" << std::endl;
    int menu, angles, figure;
    std::cin >> menu;
    std::cout << "How many angles (5, 6, 8): " << std::endl;
    std::cin >> angles;
    switch (menu) {
        case 1 :
            switch (angles) {
                case 5:
                    process<std::tuple<Vertex<int>, Vertex<int>, Vertex<int>,
Vertex<int>, Vertex<int>>>();
                    break;
                case 6:
                    process<std::tuple<Vertex<int>, Vertex<int>, Vertex<int>,
Vertex<int>, Vertex<int>, Vertex<int>>>();
                    break;
                case 8:
                    process<std::tuple<Vertex<int>, Vertex<int>, Vertex<int>,
Vertex<int>, Vertex<int>, Vertex<int>, Vertex<int>, Vertex<int>>>();
                    break;
```

```
            }
            break;
        case 2:
            switch (angles) {
                case 5:
                    process<Pentagon<int>>();
                    break;
                case 6:
                    process<Hexagon<int>>();
                    break;
                case 8:
                    process<Hexagon<int>>();
                    break;
            }
            break;
    }
    return 0;
}
```

*pentagon.hpp*

```
#pragma once
#include<iostream>
#include"vertex.hpp"
template <class T> class Pentagon{
private:
    Vertex<T> Vertexs[5];
public:
    using vertex_type = Vertex<T>;
    Pentagon();
    Pentagon(std::istream& in);
    Vertex<T> center() const;
    double square() const;
    void read(std::istream& in);
    void print(std::ostream& os) const;
};
template<class T>
Pentagon<T>::Pentagon() {}
template<class T> Pentagon<T>::Pentagon(std::istream& in) {
    for (int i = 0; i < 5; i++)
        in >> Vertexs[i];
}
template<class T> double Pentagon<T>::square() const {
    double Area = 0;
```

```
    for (int i = 0; i < 5; i++) {
        Area += (Vertexs[i].x) * (Vertexs[(i + 1) % 5].y) - (Vertexs[(i + 1) %
5].x)*(Vertexs[i].y);
    }
    Area *= 0.5;
    return abs(Area);
}
template<class T> void Pentagon<T>::print(std::ostream& os) const {
    std::cout << "Pentagon: ";
    for (int i = 0; i < 5; i++)
        std::cout << Vertexs[i] << ' ';
    std::cout << '\n';
}
template<class T> Vertex<T> Pentagon<T>::center() const {
    Vertex<T> res = Vertex<T>();
    for (int i = 0; i < 5; i++)
        res += Vertexs[i];
    return res / 5;
}
template <class T> void Pentagon<T>::read(std::istream& in) {
    Pentagon<T> res = Pentagon(in);
    *this = res;
}
```

*hexagon.hpp*

```
#pragma once
#include<iostream>
#include"vertex.hpp"
template <class T> class Hexagon{
private:
    Vertex<T> Vertexs[6];
public:
    using vertex_type = Vertex<T>;
    Hexagon();
    Hexagon(std::istream& in);
    Vertex<T> center() const;
    double square() const;
    void read(std::istream& in);
    void print(std::ostream& os) const;
};
template<class T>
Hexagon<T>::Hexagon() {}
template<class T> Hexagon<T>::Hexagon(std::istream& in) {
```

```
    for (int i = 0; i < 6; i++)
        in >> Vertexs[i];
}
template<class T> double Hexagon<T>::square() const {
    double Area = 0;
    for (int i = 0; i < 6; i++) {
        Area += (Vertexs[i].x) * (Vertexs[(i + 1) % 6].y) - (Vertexs[(i + 1) %
6].x)*(Vertexs[i].y);
    }
    Area *= 0.5;
    return abs(Area);
}
template<class T> void Hexagon<T>::print(std::ostream& os) const {
    std::cout << "Hexagon: ";
    for (int i = 0; i < 6; i++)
        std::cout << Vertexs[i] << ' ';
    std::cout << '\n';
}
template<class T> Vertex<T> Hexagon<T>::center() const {
    Vertex<T> res = Vertex<T>();
    for (int i = 0; i < 6; i++)
        res += Vertexs[i];
    return res / 6;
}
template <class T> void Hexagon<T>::read(std::istream& in) {
    Hexagon<T> res = Hexagon(in);
    *this = res;
}
```

*octagon.hpp*

```
#pragma once
#include<iostream>
#include"vertex.hpp"
template <class T> class Octagon{
private:
    Vertex<T> Vertexs[8];
public:
    using vertex_type = Vertex<T>;
    Octagon();
    Octagon(std::istream& in);
    Vertex<T> center() const;
    double square() const;
    void read(std::istream& in);
```

```
    void print(std::ostream& os) const;
};
template<class T>
Octagon<T>::Octagon(){}
template<class T> Octagon<T>::Octagon(std::istream& in) {
    for (int i = 0; i < 8; i++)
        in >> Vertexs[i];
}
template<class T> double Octagon<T>::square() const {
    double Area = 0;
    for (int i = 0; i < 8; i++) {
        Area += (Vertexs[i].x) * (Vertexs[(i + 1) % 8].y) - (Vertexs[(i + 1) %
8].x)*(Vertexs[i].y);
    }
    Area *= 0.5;
    return abs(Area);
}
template<class T> void Octagon<T>::print(std::ostream& os) const {
    std::cout << "Octagon: ";
    for (int i = 0; i < 8; i++)
        std::cout << Vertexs[i] << ' ';
    std::cout << '\n';
}
template<class T> Vertex<T> Octagon<T>::center() const {
    Vertex<T> res = Vertex<T>();
    for (int i = 0; i < 8; i++)
        res += Vertexs[i];
    return res / 8;
}
template <class T> void Octagon<T>::read(std::istream& in) {
    Octagon<T> res = Octagon(in);
    *this = res;
}
```

## 4. Объяснение результатов работы программы

Программа просит у пользователя выбрать каким образом хранить фигуру, заданную несколькими вершинами, затем выводится меню для вычисления площадей фигур. Отличительной чертой данной программы является то, что функции, которые конструируют, выводят или считают площади могут одновременно работать как с обычными фигурами, заданными через экземпляры класса, так и tuple различной длины.

## 5. Вывод

С помощью навыков полученных в ходе выполнения данной лабораторной работы программист может в будущем реализовать различные шаблонные функции с особенными спецификациями, которые будут способны работать с большим кол-вом параметров и аргументов, кроме того, несложно убедиться, что благодаря шаблонам программист получает в свой руки инструментарий логического программирования, который при умелом использовании может помочь решить ряд специфических задач.