

**Московский авиационный институт
(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 4

Тема: Метапрограммирование C++

Студент: Пермяков Никита
Александрович

Группа: 80-208

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`.

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 координатоми 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`.

Вариант: 25

- Треугольник
- Квадрат
- Прямоугольник

Цель:

- Изучение основ работы с шаблонами (template) в C++;
- Изучение шаблонов std::pair, std::tuple
- Получение навыка работы со специализацией шаблонов и идиомой SFINAE

2. Описание программы

1. Программа выполняет определённые действия по введённым командам:
 - a. 0 — выход из программы;
 - b. 1,2,3 - создание фигуры (Квадрат, Прямоугольник, Треугольник соответственно), обновление вершин через ввод и проверка;
 - c. 4 - вывод данных вершин;
 - d. 5 - вывод площади
2. Функция Print() возвращает содержимое кортежа. Она определена для tuple.
3. Функция Square() вычисляет площадь данной фигуры или совокупности точек в кортеже в зависимости от типа фигуры по методу Гаусса (формула землемера, метод шунтирования) и возвращает это значение.

3. Набор тестов

Пояснение:

- 1) На ввод подается число выбора действия из текстового интерфейса
- 2) При обновлении координат фигуры – выводится номер пары координат фигуры
- 3) Выводится информация о состоянии tuple фигур, происходит проверка тестами

Test 1

4

5

2

6.4 -5.4

0.55 3.7

8.7 609

-44.35 -3.413

4

5

0

Test 2

4

5

3

954.4 -4.54

55 3.8

0.35 1.413

4

5

0

Test 3

4

5

1

64 -54

55 3

87 609

4435 3413

4

5

0

4. Результаты выполнения тестов

Test 1

'1' - Create rectangle

'2' - Create square

'3' - Create triangle

'4' - Print coords

'5' - Total Square

'0' - Exit

4

Coords of class Rectangle<int>: (0, 0) (0, 10) (10, 10) (10, 0)

Coords of class Quadrate<double>: (3.56, 0.6) (3.56, 1.6) (3.96, 1.6) (3.96, 0.6)

Coords of class Triangle<double>: (-1, 8.1) (-3.56, 8.1) (-3.56, 1.6)

5

Total square: 121

3

enter 1 pair: 954.4 -4.54

enter 2 pair: 55 3.8

enter 3 pair: 0.35 1.413

4

Coords of class Rectangle<int>: (0, 0) (0, 10) (10, 10) (10, 0)

Coords of class Quadrate<double>: (3.56, 0.6) (3.56, 1.6) (3.96, 1.6) (3.96, 0.6)

Coords of class Triangle<double>: (954.4, -4.54) (55, 3.8) (0.35, 1.413)

5

Total square: 290.5

0

Test 2

'1' - Create rectangle

'2' - Create square

'3' - Create triangle

'4' - Print coords

'5' - Total Square

'0' - Exit

4

Coords of class Rectangle<int>: (0, 0) (0, 10) (10, 10) (10, 0)

Coords of class Quadrate<double>: (3.56, 0.6) (3.56, 1.6) (3.96, 1.6) (3.96, 0.6)

Coords of class Triangle<double>: (-1, 8.1) (-3.56, 8.1) (-3.56, 1.6)

5

Total square: 121

3

enter 1 pair: 954.4 -4.54

enter 2 pair: 55 3.8

enter 3 pair: 0.35 1.413

4

Coords of class Rectangle<int>: (0, 0) (0, 10) (10, 10) (10, 0)

Coords of class Quadrate<double>: (3.56, 0.6) (3.56, 1.6) (3.96, 1.6) (3.96, 0.6)

Coords of class Triangle<double>: (954.4, -4.54) (55, 3.8) (0.35, 1.413)

5

Total square: 290.5

0

Test 3

'1' - Create rectangle

'2' - Create square

'3' - Create triangle

'4' - Print coords

'5' - Total Square

'0' - Exit

4

Coords of class Rectangle<int>: (0, 0) (0, 10) (10, 10) (10, 0)

Coords of class Quadrate<double>: (3.56, 0.6) (3.56, 1.6) (3.96, 1.6) (3.96, 0.6)

Coords of class Triangle<double>: (-1, 8.1) (-3.56, 8.1) (-3.56, 1.6)

5

Total square: 121

1

enter 1 pair: 64 -54

enter 2 pair: 55 3

enter 3 pair: 87 609

enter 4 pair: 4435 3413

4

Coords of class Rectangle<int>: (64, -54) (55, 3) (87, 609) (4435, 3413)

Coords of class Quadrate<double>: (3.56, 0.6) (3.56, 1.6) (3.96, 1.6) (3.96, 0.6)

Coords of class Triangle<double>: (-1, 8.1) (-3.56, 8.1) (-3.56, 1.6)

5

Total square: 8.75387e+06

0

5. Листинг программы

main.cpp

```
#include <iostream>

#include <tuple>
#include "rectangle.h"
#include "quadrature.h"
#include "triangle.h"

void printHelp() {

    std::cout << "'1'          - Create rectangle" << std::endl;
    std::cout << "'2'          - Create square" << std::endl;
    std::cout << "'3'          - Create triangle" << std::endl;
    std::cout << "'4'          - Print coords" << std::endl;
    std::cout << "'5'          - Total Square" << std::endl;
    std::cout << "'0'          - Exit" << std::endl;
}

enum {
    quit,
    add_rectangle,
    add_square,
    add_triangle,
    all_print,
    sum_area
};

template<typename T, uint8_t index>
typename std::enable_if<index == std::tuple_size<T>::value,
void>::type Print(T&) {
    std::cout << std::endl;
}
```



```

template<typename T, uint8_t index>
typename std::enable_if<index < std::tuple_size<T>::value,
void>::type Print(T& tuple) {
    auto item = std::get<index>(tuple);
    std::cout << "Coords of " << typeid(item).name() << ":\t";
    itemPrint(item);
    Print<T, index + 1>(tuple);
}

```

```

template<typename T>
typename std::enable_if<(sizeof(T) / sizeof(T::P[0]) > 0),
void>::type itemPrint(T& vertex) {
    for (auto v : vertex.P)
        std::cout << " ( " << v.coord.first << ", " <<
v.coord.second << " ) ";
    std::cout << std::endl;
}

```

```

template <class T, uint8_t index>
double Square(T& tuple) {
    auto item = std::get<index>(tuple);
    double value = itemSquare(item);

    if constexpr ((index + 1) < std::tuple_size<T>::value) {
        return value + Square<T, index + 1>(tuple);
    }
    return value;
}

```

```

template <class T>
auto itemSquare(T& vertex) {
    auto area = 0;
    for (int i = 0; i < (sizeof(T) / sizeof(T::P[0])) - 1; ++i)
        area += (vertex.P[i + 1].coord.first -
vertex.P[i].coord.first) * (vertex.P[i + 1].coord.second +
vertex.P[i].coord.second);
    return std::abs(area) / 2.0;
}

```

```

int main() {
    int cmd;

```

```

double left, right;
unsigned int i;

Rectangle<int> rec;
rec.P[0].coord = { 0, 0 };
rec.P[1].coord = { 0, 10 };
rec.P[2].coord = { 10, 10 };
rec.P[3].coord = { 10, 0 };

Quadrangle<double> qua;
qua.P[0].coord = { 3.56, 0.6 };
qua.P[1].coord = { 3.56, 1.6 };
qua.P[2].coord = { 3.96, 1.6 };
qua.P[3].coord = { 3.96, 0.6 };

Triangle<double> tri;
tri.P[0].coord = { -1, 8.1 };
tri.P[1].coord = { -3.56, 8.1 };
tri.P[2].coord = { -3.56, 1.6 };
std::tuple<Rectangle<int>, Quadrangle<double>, Triangle<double>>
basic_set{rec, qua, tri};

while (true) {
    printHelp();
    std::cin >> cmd;
    std::cout << std::endl;
    switch (cmd) {
        case add_rectangle: {
            Rectangle<int> rec;
            for (i = 0; i < 4; ++i) {
                std::cout << std::endl << "enter " << i + 1 << "
pair: ";

                std::cin >> left >> right;
                rec.P[i].coord = { left, right };
            }
            std::get<0>(basic_set) = rec;
            break;
        }
        case add_square: {
            Quadrangle<double> que;
            for (i = 0; i < 4; ++i) {
                std::cout << std::endl << "enter " << i + 1 << "
pair: ";

```

```

        std::cin >> left >> right;
        que.P[i].coord = { left, right };
    }
    std::get<1>(basic_set) = que;
    break;
}
case add_triangle: {
    Triangle<double> tri;
    for (i = 0; i < 3; ++i) {
        std::cout << std::endl << "enter " << i + 1 << "
pair: ";
        std::cin >> left >> right;
        tri.P[i].coord = { left, right };
    }
    std::get<2>(basic_set) = tri;
    break;
}
case all_print: {
    Print<decltype(basic_set), 0>(basic_set);
    break;
}
case sum_area: {
    std::cout << "Total square: " <<
Square<decltype(basic_set), 0>(basic_set) << std::endl << std::endl;
    break;
}
case quit: {
    return 0;
}
default: {
    std::cout << "Undefined cmd" << std::endl;
}
}
}
return 0;
}

```

point.h

```
#pragma once
```

```
#include <iostream>
```

```
#include <cmath>
```

```
template<typename T>
```

```
class Point {
```

```
public:
```

```
    Point();
```

```
    Point(T a, T b);
```

```
    Point(const Point& other);
```

```
    virtual ~Point() {};
```

```
    Point<T> operator+ (const Point<T>& a);
```

```
    Point<T> operator- (const Point<T>& a);
```

```
    Point<T> operator* (T&& a);
```

```
    Point<T> operator/ (T&& a);
```

```
    void Print(const Point<T>& a);
```

```
    friend std::istream& operator>> (std::istream& in, Point& a);
```

```
    std::pair<T, T> coord;
```

```
};
```

```
template<class T>
```

```
Point<T>::Point() {}
```

```
template<class T>
```

```
Point<T>::Point(T a, T b) : coord{ a, b } {}
```

```
template<class T>
```

```
Point<T>::Point(const Point& other) : coord{ other.coord } {}
```

```
template<class T>
```

```

Point<T> Point<T>::operator+ (const Point<T>& a) {
    this->coord = { this->coord.first + a.coord.first,
this->coord.second + a.coord.second };
    return *this;
}

```

```

template<class T>
Point<T> Point<T>::operator- (const Point<T>& a) {
    this->coord = { this->coord.first - a.coord.first,
this->coord.second - a.coord.second };
    return *this;
}

```

```

template<class T>
Point<T> Point<T>::operator* (T&& a) {
    this->coord = { this->coord.first * a, this->coord.second * a };
    return *this;
}

```

```

template<class T>
Point<T> Point<T>::operator/ (T&& a) {
    this->coord = { this->coord.first / a, this->coord.second / a };
    return *this;
}

```

```

template<class T>
void Point<T>::Print(const Point<T>& a) {
    std::cout << "point ( " << this->coord.first << ", " <<
this->coord.second << " )" << std::endl;
}

```

```

template<class T>
std::istream& operator>> (std::istream& in, Point<T>& a) {
    in >> a.coord.first >> a.coord.second;
}

```

```
        return in;
    }
```

rectangle.h

```
#pragma once
```

```
#include "point.h"
```

```
template<class T>
class Rectangle {
public:
    Point<T> P[4];
};
```

triangle.h

```
#pragma once
```

```
#include "point.h"
```

```
template<class T>
class Triangle {
public:
    Point<T> P[3];
};
```

quadrate.h

```
#pragma once
```

```
#include "point.h"
```

```
template<class T>
class Quadrate {
public:
```

```
    Point<T> P[4];  
};
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)  
  
project(lab4)  
  
add_executable(lab4 main.cpp)  
  
set_property(TARGET lab4 PROPERTY CXX_STANDARD 11)  
  
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -g")
```

6. Ссылка на репозиторий

https://github.com/nikit34/oop_exercise_04

7. Объяснение результатов работы программы

- 1) Пользователю предоставляется 4 опции: задать фигуру (квадрат, треугольник или треугольник), вывести информацию о фигурах в структуре данных tuple (координаты точек), вывести общую площадь всех фигур.
- 2) Перед занесением фигур в кортах каждая фигура проверяется. У квадрата проверяется перпендикулярность и равенство сторон, у треугольника — сумма двух сторон не может быть больше третьей стороны. После чего указатель на созданную фигуру заносится в кортеж tuple.
- 3) Вывод информации о всех фигурах производится с помощью цикла. Поочередно перебираются все элементы tuple, и с помощью метода Print() выводятся координаты каждой фигуры.
- 4) Общая площадь фигур находится посредством суммирования результата работы метода Square() для всех фигур вектора.

5) Если пользователь вводит «0», то считывание завершается, а все фигуры удаляются из памяти.

8. Вывод

В ходе работы были приобретены навыки работы с шаблонами и кортежами в C++. Написана программа, производящая операции с помощью шаблонов и работающая с кортежами. Создал базовый класс и 3 производных от него класса, которые посредством override методов переопределяли виртуальные методы базового класса.

9. Список литературы

1. Перегрузка операторов C++ [Электронный ресурс]. URL:
<https://metanit.com/cpp/tutorial/5.14.php>

(дата обращения: 29.09.2020).

2. Битовые операции C++ [Электронный ресурс]. URL:
<http://www.c-cpp.ru/books/bitovye-operator>

(дата обращения: 29.09.2020).