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: Основы метапрограммирования

Студент: Короткевич Л. В.
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. **Постановка задачи.** Вариант 12.

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:

1. Создает набор фигур согласно варианту задания (как минимум по одной фигуре каждого типа с координатами типа int и координатами типа double).
2. Сохраняет фигуры в std::tuple
3. Печатает на экран содержимое std::tuple с помощью шаблонной функции print.
4. Вычисляет суммарную площадь фигур в std::tuple и выводит значение на экран.

Вариант 12:

1. Ромб
2. Трапеция
3. Пятиугольник

2. **Описание программы**

main.cpp	
template <typename T> void readFigure(T &figure)	Считывает вершины фигуры
template <typename T, size_t index> typename std::enable_if<index == std::tuple_size<T>::value, void>::type printTuple(T &)	Печать переноса строки (или ничего) по мере достижения крайнего элемента кортежа
template <typename T, size_t index> typename std::enable_if <index < std::tuple_size<T>::value, void>::type printTuple(T &tp)	Печать каждого элемента кортежа
template <typename T> typename std::enable_if<(sizeof(T::points) / sizeof(T::points[0]) > 0), void>::type printVertices(T &figure)	Печать вершин фигуры
template <class T, size_t index> double totalTupleArea(T &tuple)	Суммарная площадь фигур кортежа
template <class T> auto figureArea(T &figure)	Площадь отдельной фигуры методом Гаусса
void printHelp()	Печать вспом. таблицы

Пользователь вводит ID команды и прозиводит соответствующее действие (1 – добавить ромб, 2 – трапецию, 3 – пятиугольник; 4 – вывести координаты всех фигур, 5 – вывести суммарную площадь фигур).

Вывод результатов производится в стандартный поток stdout.

3. **Набор тестов**

Тест №1	Пояснение	Тест №2	Пояснение	Тест №3	Пояснение
1	Добавление ромба	1	Добавление ромба	1	Добавление ромба
1 0		0 0		0 0	
2 2		1.5 0		1 1	
1 4		1.5 1.5		1 0.5	
0 2		0 1.5		1 0.5	
2	Добавление трапеции	5	Печать суммарной площади	5	Печать суммарной площади
0 0		2	Добавление трапеции	1	Добавление ромба
3 0		0.0 0.0		1 0	
2 1		2.0 0.0		2 1	
1 1		1.0 1.0		1 2	
3	Добавление пятиуг.	1.0 1.0		0 1	
3 0		5	Печать суммарной	4	Печать фигур

			площади		
5 0		3	Добавление пятиугольника	5	Печать суммарной площади
5 1		3 0		6	Выход
4 2		5 0			
3 1		5 1			
1	Добавление трапеции	4 2			
1 0		3 1			
2 2		5	Печать фигур		
1 3		4	Печать суммарной площади		
0 2		6	Выход		
4	Печать фигур				
5	Печать суммарной площади				
6	Выход				

4. Результаты выполнения тестов

[leo@pc LR4]$ ./main <test01.txt

1) Add Rhombus
2) Add Trapeze
3) Add Pentagon
4) Display all the vertices of tuple's figures
5) Display total area of tuple's figures
6) Exit
Rhombus successfully added
Trapezoid successfully added
Pentagon successfully added
Rhombus successfully added
Vertices of Rhombus: (1, 0) (2, 2) (1, 3) (0, 2)
Vertices of Trapezoid: (0, 0) (3, 0) (2, 1) (1, 1)
Vertices of Pentagon: (3, 0) (5, 0) (5, 1) (4, 2) (3, 1)
Total area of tuple's figures: 8
Exit

[leo@pc LR4]$ ./main <test02.txt

1) Add Rhombus
2) Add Trapeze
3) Add Pentagon
4) Display all the vertices of tuple's figures
5) Display total area of tuple's figures
6) Exit
Rhombus successfully added
Total area of tuple's figures: 2.25
Trapezoid successfully added
Total area of tuple's figures: 3.25
Pentagon successfully added
Total area of tuple's figures: 6.25
Vertices of Rhombus: (0, 0) (1.5, 0) (1.5, 1.5) (0, 1.5)
Vertices of Trapezoid: (0, 0) (2, 0) (1, 1) (1, 1)
Vertices of Pentagon: (3, 0) (5, 0) (5, 1) (4, 2) (3, 1)

```
Exit
[leo@pc LR4]$ ./main <test03.txt
1) Add Rhombus
2) Add Trapeze
3) Add Pentagon
4) Display all the vertices of tuple's figures
5) Display total area of tuple's figures
6) Exit
Rhombus successfully added
Total area of tuple's figures: 0.25
Rhombus successfully added
Vertices of Rhombus:     ( 1, 0 )  ( 2, 1 )  ( 1, 2 )  ( 0, 1 )
Vertices of Trapezoid:   ( 0, 0 )  ( 0, 0 )  ( 0, 0 )  ( 0, 0 )
Vertices of Pentagon:    ( 0, 0 )  ( 0, 0 )  ( 0, 0 )  ( 0, 0 )  ( 0, 0 )
Total area of tuple's figures: 2
Exit
```

5. Листинг программы

main.cpp

```
/*
Короткевич Л. В.
М8О-208Б-19
github.com/anxieuse/oop_exercise_04
Вариант 12:
   Ромб
   Трапеция
   Пятиугольник
*/

#include <iostream>
#include <algorithm>
#include <cmath>
#include <string>
#include <type_traits>
#include <tuple>

#include "Rhombus.hpp"
#include "Trapezoid.hpp"
#include "Pentagon.hpp"

// Figure Reading

template <typename T>
void readFigure(T &figure)
{
   int n = sizeof(figure.points) / sizeof(figure.points[0]);
   for (int i = 0; i < n; ++i) {
      std::cin >> figure.points[i].first >> figure.points[i].second;
   }
}

// Tuple Writing

template <typename T, size_t index>
```

```
typename std::enable_if<index == std::tuple_size<T>::value, void>::type printTuple(T &)
{
}

template <typename T, size_t index>
typename std::enable_if <index < std::tuple_size<T>::value, void>::type printTuple(T &tp)
{
   auto figure = std::get<index>(tp);
   std::cout << "Vertices of " << figure.name << ":\t";
   printVertices(figure);
   printTuple<T, index + 1>(tp);
}

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

// Area Calculations

template <class T, size_t index>
double totalTupleArea(T &tuple)
{
   auto figure = std::get<index>(tuple);
   double value = figureArea(figure);
   if constexpr ((index + 1) < std::tuple_size<T>::value)
   {
      return value + totalTupleArea<T, index + 1>(tuple);
   }
   return value;
}

template <class T>
auto figureArea(T &figure)
{
   double area = 0;
   int n = sizeof(T::points) / sizeof(T::points[0]);
   for (int i = 1; i < n; ++i)
      area += figure.points[i - 1].first * figure.points[i].second - figure.points[i].first * figure.points[i - 1].second;
   area += figure.points[n - 1].first * figure.points[0].second - figure.points[0].first * figure.points[n - 1].second;
   return std::fabs(area) / 2.0;
}

void printHelp()
{
   std::cout << "1) Add Rhombus\n";
   std::cout << "2) Add Trapeze\n";
   std::cout << "3) Add Pentagon\n";
   std::cout << "4) Display all the vertices of tuple's figures\n";
```

```
    std::cout << "5) Display total area of tuple's figures\n";
    std::cout << "6) Exit\n";
}

int main()
{
    printHelp();

    std::tuple<Rhombus<double>, Trapezoid<double>, Pentagon<int>> tp;

    int cmd;
    bool input = true;
    while (input)
    {
        std::cin >> cmd;
        switch (cmd)
        {
        case 1: // add rhombus
        {
            readFigure(std::get<0>(tp));
            std::cout << std::get<0>(tp).name << " successfully added" << '\n';
            break;
        }
        case 2: // add trapezoid
        {
            readFigure(std::get<1>(tp));
            std::cout << std::get<1>(tp).name << " successfully added" << '\n';
            break;
        }
        case 3: // add pentagon
        {
            readFigure(std::get<2>(tp));
            std::cout << std::get<2>(tp).name << " successfully added" << '\n';
            break;
        }
        case 4: // print all
        {
            printTuple<decltype(tp), 0>(tp);
            break;
        }
        case 5: // total area
        {
            std::cout << "Total area of tuple's figures: " << totalTupleArea<decltype(tp), 0>(tp) << '\n';
            break;
        }
        case 6:
        {
            input = false;
            std::cout << "Exit" << '\n';
            break;
        }
        }
```

```
    }
    return 0;
}
```
Rhombus.hpp:
```
#pragma once

template<typename T>
class Rhombus {
public:
    std::pair<T, T> points[4];
    std::string name = "Rhombus";
};
```
Trapezoid.hpp:
```
#pragma once

template<typename T>
class Trapezoid {
public:
    std::pair<T, T> points[4];
    std::string name = "Trapezoid";
};
```
Pentagon.hpp:
```
#pragma once

template<typename T>
class Pentagon {
public:
    std::pair<T, T> points[5];
    std::string name = "Pentagon";
};
```

6. Выводы

По мере выполнения данной ЛР я приобрел навыки работы с шаблонами функций, классов. Я также изучил полную и частичную специализацию шаблонов, приобрел навык реализации шаблонных функций для обхода, обработки кортежей. Получил навыки работы с идиомой SFINAE.

7. Литература

1. Уроки №173-180. Шаблоны [Электронный Ресурс]. URL: https://ravesli.com/
2. std::tuple [Электронный Ресурс]. URL: https://en.cppreference.com/w/cpp/utility/tuple
3. std::pair [Электронный Ресурс] URL: https://en.cppreference.com/w/cpp/utility/pair