**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: Ляшун Дмитрий Сергеевич

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Создать программу на C++, 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.

Использовать 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.

**Вариант №24:**

|  |  |  |
| --- | --- | --- |
| **Фигура №1** | **Фигура №2** | **Фигура №3** |
| 8-угольник | Треугольник | Квадрат |

Все фигуры выпуклые и равносторонние.

1. **Описание программы**

Для реализации заданных фигур опишем шаблонные структуры со следующими полями:

**Класс Square:**

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| a | pair<T, T> | координаты одной из вершин квадрата. |
| b | pair<T, T> | координаты противоположной для a вершины квадрата. |

**Класс Octagon:**

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| vertex | pair<T, T> | Массив координат вершин восьмиугольника. |

**Класс Triangle:**

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| v1 | pair<T, T> | Координата первой вершины треугольника. |
| v2 | pair<T, T> | Координата второй вершины треугольника. |
| v3 | pair<T, T> | Координата третьей вершины треугольника. |

Для более удобной работы с координатами вершин добавим для **std::pair** оператор вывода **<<**, а также опишем функцию **get\_length**, которая возвращает расстояние между двумя вершинами, что является соответственно стороной фигуры.

Печать фигур и вычисление их площадей будет реализовано с помощью шаблонных функций **print** и **square** с применением идиомы **SFINAE**: c помощью метода **enable\_if** будет проверяться наличие полей для заданных типов, и в зависимости от этого запускаться соответствующие реализации функции.

Для работы со структурой **std::tuple** опишем следующие шаблонные функции:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Возвращаемое значение** | **Назначение** |
| template<class T, size\_t index> print\_tuple(T& tuple) | void | Печатает содержимое tuple. |
| template <class T,size\_t index> square\_tuple(T& tuple) | double | Вычисляет и возвращает значение общей площади фигур в tuple. |

1. **Руководство по использованию программы**

**oop\_exercise\_04** - исполняемый файл программы.

**Входные данные**

Отсутствуют, поскольку нельзя интерактивно работать с std::tuple в процессе выполнения программы.

**Выходные данные**

Сперва выводится построчно набор добавленных фигур, затем значение их общей площади.

**Тестовые примеры**

1. Для набора фигур:

Square<int> s1{{0, 0}, {1, 1}};

Square<double> s2{{0, 0}, {4.5, 4.5}};

Triangle<double> t1{{1, 1}, {3, 1}, {2, 2.73205}};

Octagon<int> o1{{{1, 0}, {1, 3}, {2, 6}, {5, 6}, {6, 3}, {6, 0}, {5, -3}, {2, -3}}};

Octagon<double> o2{{{0.5, 0}, {1.5, 0}, {2, -1}, {2, -2}, {1.5, -3}, {0.5, -3}, {0, -2}, {0, -1}}};

Результат:

Введенные фигуры:

Квадрат: (0, 0), (1, 1), (0, 1), (1, 0).

Квадрат: (0, 0), (4.5, 4.5), (0, 4.5), (4.5, 0).

Треугольник: (1, 1), (3, 1), (2, 2.73205).

Восьмиугольник: (1, 0), (1, 3), (2, 6), (5, 6), (6, 3), (6, 0), (5, -3), (2, -3).

Восьмиугольник: (0.5, 0), (1.5, 0), (2, -1), (2, -2), (1.5, -3), (0.5, -3), (0, -2), (0, -1).

Общая площадь фигур: 92.0605

**4. Листинг программы**

Исходный код программы состоит из двух файлов: **figure.hpp** - содержит объявление структур **Square**, **Octagon**, **Triangle**, реализации функций работы с ними, а также функции работы с **std::tuple**; и **main.cpp** - основная программа. **CMakeLists.txt** - содержит код для утилиты cmake, которая проводит сборку программы из исходного кода.

**Содержимое файла figure.hpp**

/\*Ляшун Дмитрий Сергеевич, группа М8О-207Б-19 \*/

#include <iostream>

#include <cmath>

#include <utility>

#include <tuple>

using namespace std;

template<class T> ostream& operator<<(ostream& os, const pair<T, T> out) {

os << "(" << out.first << ", " << out.second << ")";

return os;

}

template<class T> double get\_length(const T& x1, const T& y1, const T& x2, const T& y2) {

return sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2));

}

template<class T> struct Octagon {

using type = T;

pair<T, T> vertex[8];

};

template<class T> struct Triangle {

using type = T;

pair<T, T> v1, v2, v3;

};

template<class T> struct Square {

using type = T;

pair<T, T> a, b;

};

template<class T> typename enable\_if<(sizeof(T::a)>0 && sizeof(T::b)>0), void>::type print(T& f) {

cout << "Квадрат: ";

pair<typename T::type, typename T::type> c{f.a.first, f.b.second}, d{f.b.first, f.a.second};

cout << f.a << ", " << f.b << ", " << c << ", " << d << "." << endl;

}

template<class T> typename enable\_if<(sizeof(T::v1)>0 && sizeof(T::v2)>0 && sizeof(T::v3)>0), void>::type print(T& f) {

cout << "Треугольник: ";

cout << f.v1 << ", " << f.v2 << ", " << f.v3 << "." << endl;

}

template<class T> typename enable\_if<(sizeof(T::vertex)>0), void>::type print(T& f) {

cout << "Восьмиугольник: ";

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

cout << f.vertex[i];

if (i != 7) {

cout << ", ";

}

else {

cout << "." << endl;

}

}

}

template<class T> typename enable\_if<(sizeof(T::a)>0 && sizeof(T::b)>0), typename T::type>::type square(T& f) {

auto side = get\_length(f.a.first, f.b.second, f.b.first, f.a.second);

return static\_cast<typename T::type>(side\*side);

}

template<class T> typename enable\_if<(sizeof(T::v1)>0 && sizeof(T::v2)>0 && sizeof(T::v3)>0), typename T::type>::type square(T& f) {

auto side = get\_length(f.v1.first, f.v1.second, f.v2.first, f.v2.second);

return static\_cast<typename T::type>(side\*side\*sqrt(3)/4);

}

template<class T> typename enable\_if<(sizeof(T::vertex)>0), typename T::type>::type square(T& f) {

auto side = get\_length(f.vertex[0].first, f.vertex[0].second, f.vertex[1].first, f.vertex[1].second);

return static\_cast<typename T::type>(8\*side\*side/(4\*tan(acos(-1)/8)));

}

template<class T, size\_t index> void print\_tuple(T& tuple) {

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

auto figure = get<index>(tuple);

print(figure);

print\_tuple<T, index+1>(tuple);

}

}

template <class T,size\_t index> double square\_tuple(T& tuple){

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

double value = square(item);

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

return value + square\_tuple<T,index+1>(tuple);

}

return value;

}

**Содержимое файла main.cpp**

/\* Ляшун Дмитрий Сергеевич, группа М8О-207Б-19 \*/

#include "figure.hpp"

int main() {

Square<int> s1{{0, 0}, {1, 1}};

Square<double> s2{{0, 0}, {4.5, 4.5}};

Triangle<double> t1{{1, 1}, {3, 1}, {2, 2.73205}};

Octagon<int> o1{{{1, 0}, {1, 3}, {2, 6}, {5, 6}, {6, 3}, {6, 0}, {5, -3}, {2, -3}}};

Octagon<double> o2{{{0.5, 0}, {1.5, 0}, {2, -1}, {2, -2}, {1.5, -3}, {0.5, -3}, {0, -2}, {0, -1}}};

tuple<decltype(s1), decltype(s2), decltype(t1), decltype(o1), decltype(o2)> tuple {s1, s2, t1, o1, o2};

cout << "Введенные фигуры: " << endl;

print\_tuple<decltype(tuple), 0>(tuple);

cout << "Общая площадь фигур: " << square\_tuple<decltype(tuple), 0>(tuple) << endl;

return 0;

}

**Содержимое файла CMakeLists.txt**

set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -std=c++17 -Wall")

cmake\_minimum\_required(VERSION 2.8) # Проверка версии CMake.

project(oop\_exercise\_04) # Название проекта

set(SOURCE\_EXE main.cpp figure.hpp) # Установка переменной со списком исходников для исполняемого файла

add\_executable(oop\_exercise\_04 ${SOURCE\_EXE}) # Создает исполняемый файл с именем oop\_exercise\_04

**5. Выводы**

В результате выполнения лабораторной работы я познакомился с основами метапрограммирования в C++, в частности научился создавать собственные шаблонные классы и функции, а также использовать уже встроенные шаблонные структуры **std::pair** и **std::tuple**. Также я получил навыки работы со специализацией шаблонов и идиомой **SFINAE**.

**Список литературы**

1. Справочник по языку C++ [Электронный ресурс]. URL: [http://www.cplusplus.com/](about:blank) (дата обращения: 01.11.2020)

2. Archived Stack Overflow Documentation [Электронный ресурс]. URL: <https://www.riptutorial.com/> (дата обращения 01.11.2020)