Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»
Дисциплина «Объектно-ориентированное программирование»

Лабораторная работа №4

Тема: Основы метапрограммирования

Студент: Инютин М. А.
Группа: М8О-207Б-19
Преподаватель: Чернышев Л. Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

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

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 шаблонные 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:

- 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.

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

Вариант 2: Квадрат, прямоугольник, трапеция.

2. Описание программы

Для каждой фигуры будем хранить координаты левого нижнего угла в виде std::pair<T, T>, для которого переопределим операцию вывода. Квадрат так же имеет поле длины стороны, прямоугольник ширину и высоту, трапеция длины оснований и высоты. Опишем шаблонные функции для вычисления площади и печати координат вершин каждой фигуры. Фигуры будем хранить в std::tuple, которая создаётся std::make_tuple. Печать фигур и вычисление площади реализуем с помощью выражение constexpr и std::get.

3. Вывод программы

Программа выводит на экран результат выполнения шаблонной функции PrintTuple и TotalSquare.

```
Square {(0, 0), (0, 1), (1, 1), (1, 0)}
Rectanle {(5, 5), (5, 8), (7, 8), (7, 5)}
Trapeze {(1, 1), (2, 5), (6, 5), (7, 1)}
Square {(-0.5, -0.5), (-0.5, 1.5), (1.5, 1.5), (1.5, -0.5)}
Rectanle {(-5, -2.5), (-5, 0), (0, 0), (0, -2.5)}
Trapeze {(-10, -10), (-8.5, -7), (-6, -7), (-4.5, -10)}
Total square is 55.5
```

4. Листинг программы

Программа разбита на файлы: square.hpp, rectangle.hpp, trapeze.hpp, main.cpp. В main.cpp описана работа с std::tuple, а в остальных файлах функции и классы для работы с фигурами.

square.hpp

```
#ifndef SQUARE_HPP
#define SQUARE_HPP

#include <iostream>
#include <tuple>
#include <vector>

template<class T>
struct Square {
	/* Cords of left bottom corner, side */
	std::pair<T, T> Cord;
	T Side;

	Square(const std::pair<T, T> & cord, T side) : Cord(cord),
Side(side) {}
};

template<class T>
T CalcSquare(const Square<T> & Sq) {
	return Sq.Side * Sq.Side;
}

template<class T>
std::ostream & operator << (std::ostream & out, const Square<T> &
sq) {
	out << "Square {";
	out << std::pair<T, T>(sq.Cord.first, sq.Cord.second) << ", ";
	out << std::pair<T, T>(sq.Cord.first, sq.Cord.second +
sq.Side) << ", ";
	out << std::pair<T, T>(sq.Cord.first + sq.Side, sq.Cord.second
+ sq.Side) << ", ";
	out << std::pair<T, T>(sq.Cord.first + sq.Side,
sq.Cord.second);
	out << "}";
	return out;
}

#endif /* SQUARE_HPP */
```

rectangle.hpp

```
#ifndef RECTANGLE_CPP
#define RECTANGLE_CPP

#include <iostream>
#include <tuple>
#include <vector>

template<class T>
struct Rectangle {
	/* Cords of left bottom corner, width and height */
	std::pair<T, T> Cord;
	T Width, Height;

	Rectangle(const std::pair<T, T> & cord, T width, T height) :
Cord(cord), Width(width), Height(height) {}
};

template<class T>
T CalcSquare(const Rectangle<T> & rectangle) {
	return rectangle.Width * rectangle.Height;
}

template<class T>
std::ostream & operator << (std::ostream & out, const Rectangle<T>
& rectangle) {
	out << "Rectanle {";
	out << std::pair<T, T>(rectangle.Cord.first,
rectangle.Cord.second) << ", ";
	out << std::pair<T, T>(rectangle.Cord.first,
rectangle.Cord.second + rectangle.Height) << ", ";
	out << std::pair<T, T>(rectangle.Cord.first + rectangle.Width,
rectangle.Cord.second + rectangle.Height) << ", ";
	out << std::pair<T, T>(rectangle.Cord.first + rectangle.Width,
rectangle.Cord.second);
	out << "}";
	return out;
}

#endif /* RECTANGLE_CPP */
```

trapeze.hpp

```
#ifndef TRAPEZE_HPP
#define TRAPEZE_HPP

#include <iostream>
#include <tuple>
#include <vector>

template<class T>
struct Trapeze {
     /* Cords of left bottom corner, greater and smaller base,
heigth */
     std::pair<T, T> Cord;
     T GreaterBase, SmallerBase, Height;

     Trapeze(const std::pair<T, T> & cord, T greaterBase, T
smallerBase, T height) : Cord(cord), GreaterBase(greaterBase),
SmallerBase(smallerBase), Height(height) {
          if (SmallerBase > GreaterBase) {
               std::swap(SmallerBase, GreaterBase);
          }
     }
};

template<class T>
T CalcSquare(const Trapeze<T> & trapeze) {
     return (trapeze.Height * (trapeze.GreaterBase +
trapeze.SmallerBase)) / 2.0;
}

template<class T>
std::ostream & operator << (std::ostream & out, const Trapeze<T> &
trapeze) {
     T d = (trapeze.GreaterBase - trapeze.SmallerBase) / 2.0;
     out << "Trapeze {";
     out << std::pair<T, T>(trapeze.Cord.first,
trapeze.Cord.second) << ", ";
     out << std::pair<T, T>(trapeze.Cord.first + d,
trapeze.Cord.second + trapeze.Height) << ", ";
     out << std::pair<T, T>(trapeze.Cord.first +
trapeze.SmallerBase + d, trapeze.Cord.second + trapeze.Height) <<
", ";
     out << std::pair<T, T>(trapeze.Cord.first +
trapeze.GreaterBase, trapeze.Cord.second);
     out << "}";
     return out;
}

#endif /* TRAPEZE_HPP */
```

main.hpp

```
#include "square.hpp"
#include "rectangle.hpp"
#include "trapeze.hpp"

/*
 * Инютин М А М8О-207Б-19
 * 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 шаблонные функции:
 * - 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).
 * - 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 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.
 * При реализации шаблонных функций допускается использование
 * вспомогательных шаблонов std::enable_if, std::tuple_size,
 * std::is_same.
 * Квадрат, прямоугольник, трапеция.
 */

using IntVertex = std::pair<int, int>;

using DoubleVertex = std::pair<long double, long double>;

template<class T1, class T2>
std::ostream & operator << (std::ostream & out, const
std::pair<T1, T2> & p) {
     out << "(" << p.first << ", " << p.second << ")";
     return out;
}
```

```
template<class T, size_t index = 0>
void PrintTuple(T & tup) {
     if constexpr(index < std::tuple_size<T>::value) {
          std::cout << std::get<index>(tup) << std::endl;
          PrintTuple<T, index + 1>(tup);
     }
}

template<class T, size_t index = 0>
long double TotalSquare(T & tup) {
     if constexpr(index < std::tuple_size<T>::value) {
          return (long double)CalcSquare(std::get<index>(tup)) +
TotalSquare<T, index + 1>(tup);
     } else {
          return 0;
     }
}

signed main() {
     Square<int> squareInt(IntVertex(0, 0), 1);
     Rectangle<int> rectangleInt(DoubleVertex(5, 5), 2, 3);
     Trapeze<int> trapezeInt(IntVertex(1, 1), 6, 4, 4);

     Square<long double> squareDouble = {DoubleVertex(-0.5, -0.5),
2};
     Rectangle<long double> rectangleDouble(DoubleVertex(-5.0, -
2.5), 5.0, 2.5);
     Trapeze<long double> trapezeDouble(DoubleVertex(-10.0, -10.0),
5.5, 2.5, 3.0);

     auto tup = std::make_tuple(squareInt, rectangleInt,
trapezeInt, squareDouble, rectangleDouble, trapezeDouble);
     PrintTuple(tup);
     std::cout << "Total square is " << TotalSquare(tup) <<
std::endl;
     return 0;
}
```

5. Выводы

Я научился работать с шаблонами, std::pair и std::tuple в C++. Во время выполнения работы я реализовал свой tuple, чтобы лучше понять работу std::tuple. Вызвало сложность использование функций std::make_tuple и std::get, потому что первая возвращает сложный тип, а вторая принимает не менее сложный тип (в первом случае помогло использование auto).

Список литературы

1. Основы шаблонов C++: шаблоны функций / Хабр — Habr
   URL: https://habr.com/ru/post/436880/ (дата обращения 27.10.2020).
2. Шаблоны в C++ — Википедия
   URL: https://ru.wikipedia.org/wiki/Шаблоны_C%2B%2B
   (дата обращения 20.10.2020).