**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: Савченко Илья Владимирович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Вариант 24.

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 23:

1. Квадрат
2. Треугольник
3. Восьмиугольник
4. **Описание программы**

|  |  |
| --- | --- |
| main.cpp | |
| using coord\_t = double; | Тип координат |
| template<typename T, class R>  typename std::enable\_if<..., bool>::type  readFigure(T figure) | Чтение координат вершин фигуры из стандартного потока |
| template<typename T, size\_t index>  typename std::enable\_if<index == std::tuple\_size<T>::value, void>::type  printTuple(T &) | Печать элементов кортежа (данная функция является условием окончания хвостовой рекурсии) |
| template<typename T, size\_t index>  typename std::enable\_if<index < std::tuple\_size<T>::value, void>::type  printTuple(T &t) | Печать элементов кортежа |
| template<typename T>  typename std::enable\_if<(sizeof(T::points) / sizeof(T::points[0]) > 0), void>::type  printVertices(T &figure) | Печать координат вершин фигур на экран |
| template<class T, size\_t index>  coord\_t totalTupleArea | Подсчет общей площади в кортеже |
| template<class R> double triangleArea(std::pair<R,R> const &first, &second, &third) | Подсчитывает площадь треугольника (вспомогательная) |
| template<class T>  auto figureArea(T &figure) | Подсчет площади фигуры |
| void help() | Вывод краткой справки на экран |

1. **Набор тестов**

Для ввода треугольника указываются 3 точки, для квадрата — 2 точки (стороны AB, остальные точки достраиваются), для восьмиугольника — 2 точки: центр и одна из вершин.  
  
Квадрат:  
A - (0, 0)  
B - (0, 3)

C - (3, 3)  
D - (3, 0)  
S = 9  
  
Треугольник:  
A - (0, 0)  
B - (2, 3)  
C - (5, 0)  
S = 7.5  
  
Восьмиугольник  
Центр - (0, 0)  
A - (1, 4)  
B - (3.53, 2.12)  
C - (4, -1)  
D - (2.12, -3.53)  
…  
S = 48.08

1. **Результаты выполнения тестов**  
   $ ./a.out

to add:

s - square

t - triangle

o - octagon

---

v - print vertices

a - total area

>s

Suare, 2 points

0 0

0 3

Square successfully added

>t

Triangle, 3 points

0 0

2 3

5 0

Triangle successfully added

>o

Octagon, 2 points

0 0

1 4

Octagon successfully added

>v

Vertices of Square: ( 0, 0 ) ( 0, 3 ) ( 3, 3 ) ( 3, 0 )

Vertices of Triangle: ( 0, 0 ) ( 2, 3 ) ( 5, 0 )

Vertices of Octagon: ( 1, 4 ) ( 3.53553, 2.12132 ) ( 4, -1 ) ( 2.12132, -3.53553 ) ( -1, -4 ) ( -3.53553, -2.12132 ) ( -4, 1 ) ( -2.12132, 3.53553 )

>a

Total area of tuple

S = 64.5833

>^D

1. **Листинг программы**

/\*\*

\* Савченко И.В.

\* М8О-208Б-19

\* https://github.com/ShyFly46/oop\_excercise\_04

\* Вариант 24:

\* \* Квадрат

\* \* Треугольник

\* \* Восьмиугольник

\*\*/

#include <iostream>

#include <type\_traits>

#include <cmath>

#include <tuple>

#include <utility>

#include <string>

template <typename T>

struct Triangle {

std::pair<T, T> points[3];

std::string name = "Triangle";

};

template <typename T>

struct Square {

std::pair<T, T> points[4];

const std::string name = "Square";

};

template <typename T>

struct Octagon {

std::pair<T, T> points[8];

std::string name = "Octagon";

};

using coord\_t = double;

template<typename T, class R>

typename std::enable\_if<std::is\_same<T, Square<R>&>::value, bool>::type

readFigure(T figure) {

std::cout << "Suare, 2 points\n";

std::cin >> figure.points[0].first >> figure.points[0].second

>> figure.points[1].first >> figure.points[1].second;

std::pair<R, R> vec(

figure.points[1].second - figure.points[0].second,

figure.points[0].first - figure.points[1].first

);

figure.points[3].first = figure.points[0].first + vec.first;

figure.points[3].second = figure.points[0].second + vec.second;

figure.points[2].first = figure.points[1].first + vec.first;

figure.points[2].second = figure.points[1].second + vec.second;

return true;

}

template<typename T, class R>

typename std::enable\_if<std::is\_same<T, Octagon<R>&>::value, bool>::type

readFigure(T figure) {

std::cout << "Octagon, 2 points\n";

std::pair<R, R> centre;

std::cin >> centre.first

>> centre.second;

std::cin >> figure.points[0].first

>> figure.points[0].second;

std::pair<R, R> vec;

vec.first = figure.points[0].first - centre.first;

vec.second = figure.points[0].second - centre.second;

figure.points[4].first = centre.first - vec.first;

figure.points[4].second = centre.second - vec.second;

R temp = vec.second;

vec.second = -vec.first;

vec.first = temp;

figure.points[2].first = centre.first + vec.first;

figure.points[2].second = centre.second + vec.second;

figure.points[6].first = centre.first - vec.first;

figure.points[6].second = centre.second - vec.second;

const double r2 = 1 / std::sqrt(2);

std::pair<R, R> dia(

(vec.first + vec.second) \* r2,

(vec.second - vec.first) \* r2

);

figure.points[3].first = centre.first + dia.first;

figure.points[3].second = centre.second + dia.second;

figure.points[7].first = centre.first - dia.first;

figure.points[7].second = centre.second - dia.second;

temp = dia.second;

dia.second = -dia.first;

dia.first = temp;

figure.points[5].first = centre.first + dia.first;

figure.points[5].second = centre.second + dia.second;

figure.points[1].first = centre.first - dia.first;

figure.points[1].second = centre.second - dia.second;

return true;

}

template<typename T, class R>

typename std::enable\_if<std::is\_same<T, Triangle<R>&>::value, bool>::type

readFigure(T figure) {

std::cout << "Triangle, 3 points\n";

for(int i = 0; i < 3; ++i){

std::cin >> figure.points[i].first

>> figure.points[i].second;

}

return true;

}

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 &t) {

auto figure = std::get<index>(t);

std::cout << "Vertices of " << figure.name << ":\t";

printVertices(figure);

printTuple<T, index + 1>(t);

}

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 << '\n';

}

template<class T, size\_t index>

double totalTupleArea(T &tuple) {

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

coord\_t value = figureArea(figure);

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

return value + totalTupleArea<T, index + 1>(tuple);

}

return value;

}

template<class R>

double triangleArea(

std::pair<R, R> const &first,

std::pair<R, R> const &second,

std::pair<R, R> const &third)

{

// a x

// b y

double a = second.first - first.first;

double b = second.second - first.second;

double x = third.first - first.first;

double y = third.second - first.second;

return std::fabs((a\*y - x\*b) \* 0.5);

}

template<class T>

double figureArea(T &figure) {

double area = 0;

int n = sizeof(T::points) / sizeof(T::points[0]);

for (size\_t i = 2; i < n; ++i)

area += triangleArea(

figure.points[0],

figure.points[i-1],

figure.points[i]

);

return area;

}

void help(){

std::cout << "to add:\ns - square\nt - triangle\no - octagon\n";

std::cout << "---\nv - print vertices\na - total area\n";

}

int main() {

std::tuple<Square<double>, Triangle<double>, Octagon<double>> tp;

help();

char cmd;

std::cout << ">";

while(std::cin >> cmd){

switch (cmd) {

case 's': // add square

{

auto &fig = std::get<0>(tp);

if(readFigure<decltype(fig), coord\_t>(fig))

std::cout << std::get<0>(tp).name << " successfully added\n";

break;

}

case 't': // add triangle

{

auto &fig = std::get<1>(tp);

if(readFigure<decltype(fig), coord\_t>(fig))

std::cout << std::get<1>(tp).name << " successfully added\n";

break;

}

case 'o': // add octagon

{

auto &fig = std::get<2>(tp);

if(readFigure<decltype(fig), coord\_t>(fig))

std::cout << std::get<2>(tp).name << " successfully added\n";

break;

}

case 'v': { // all vertices

printTuple<decltype(tp), 0>(tp);

break;

}

case 'a': { // total area

std::cout << "Total area of tuple\nS = " << totalTupleArea<decltype(tp), 0>(tp) << '\n';

break;

}

default:{

help();

break;

}

}

std::cout << ">";

}

std::cout << '\n';

return 0;

}

1. **Выводы**  
   По мере выполнения данной лабораторной работы я приобрел навыки работы с шаблонами функций и классов, методы специализации шаблонов, научился использовать конструкцию if consexpr, познакомился с идиомой SFINAE. А также познакомился с std::tuple и std::pair, а также работе с ними.

**Литаретура**

1. Основы шаблонов C++: шаблоны функций [Электронный ресурс]. URL: <https://habr.com/ru/post/436880/>  
   (дата обращения 01.02.2021)
2. std::tuple [Электронный Ресурс].  
   URL: <https://en.cppreference.com/w/cpp/utility/tuple>  
   (дата обращения 01.02.2021)
3. Templates - C++ Tutorials [Электронный ресурс].  
   URL: <https://www.cplusplus.com/doc/oldtutorial/templates/>  
   (дата обращения 01.02.2021)