Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Лабораторной работе № 04
по курсу «Объектно-ориентированное программирование»

Тема:
«Основы метапрограммирования»

Студент:	Пшеницын А. А.
Группа:	М80-208Б-18
Преподаватель:	Журавлев А. А.
Вариант:	17
Оценка:	
Дата:	

Москва
2019

Цель:

- Изучение основ работы с шаблонами (template) в C++;

Задание(Вариант 17)

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения.
Создать набор шаблонов, создающих функции, реализующие:
1. Вычисление геометрического центра фигуры;
2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;
3. Вычисление площади фигуры;
Параметром шаблона должен являться тип класса фигуры (например Square<int>). Помимо самого класса фигуры, шаблонная функция должна уметь работать с tuple. Например, std::tuple<std::pair<int,int>, std::pair<int,int>,std::pair<int,int>> должен интерпретироваться как треугольник. std::tuple<std::pair<int,int>, std::pair<int,int>,std::pair<int,int>, std::pair<int,int>> - как квадрат. Каждый std::pair<int,int> - соответствует координатам вершины
фигуры вращения.
Создать программу, которая позволяет:
• Вводить из стандартного ввода std::cin фигуры, согласно варианту задания (как в виде класса, так и в виде std::tuple).
• Вызывать для нее шаблонные функции (1-3).
При реализации шаблонных функций допускается использование вспомогательных шаблонов std::enable_if, std::tuple_size, std::is_same.

Код программы

vertex.h

```
#ifndef D_VERTEX_H_
#define D_VERTEX_H_ 1

#include <iostream>
#include<vector>
#include<cmath>
#include<string.h>

template<class T>
struct vertex {
     T x;
     T y;
};

template<class T>
std::istream& operator>> (std::istream& is, vertex<T>& p) {
   is >> p.x >> p.y;
   return is;
}

template<class T>
std::ostream& operator<< (std::ostream& os, const vertex<T>& p) {
   os << p.x << ' ' << p.y;
   return os;
}

template<class T>
double scalar_mult(vertex<T> top1_end, vertex<T> top_begin, vertex<T> top2_end){
     return (top_begin.x - top1_end.x)*(top_begin.x - top2_end.x) + (top_begin.y -
top1_end.y)*(top_begin.y - top2_end.y);
}

template<class T>
double segment_length(vertex<T> top1, vertex<T> top2){
     return sqrt(pow(top1.x - top2.x, 2) + pow(top1.y - top2.y, 2));
}

#endif // D_VERTEX_H_
```

triangle.h

```
#ifndef D_TRIANGLE_H_
#define D_TRIANGLE_H_ 1

#include <algorithm>
#include <iostream>

#include "vertex.h"

template<class T>
struct triangle {
private:
  vertex<T> verts[3];
public:
  triangle(std::istream& is);
  triangle(vertex<T>& p1, vertex<T>& p2, vertex<T>& p3);

  double area() const;
  vertex<double> center() const;
  void print(std::ostream& os) const;

};

template<class T>
triangle<T>::triangle(std::istream& is) {
     for(int i = 0; i < 3; ++i){
          is >> verts[i];
     }
     if((verts[0].x - verts[1].x) * (verts[2].y - verts[1].y) == (verts[0].y - verts[1].y) *
(verts[2].x - verts[1].x)){
          throw std::logic_error("It is not triangle");
     }
}

template<class T>
triangle<T>::triangle(vertex<T>& p1, vertex<T>& p2, vertex<T>& p3){
     verts[0].x = p1.x;
     verts[0].y = p1.y;
     verts[1].x = p2.x;
     verts[1].y = p2.y;
     verts[2].x = p3.x;
     verts[2].y = p3.y;
     if((verts[0].x - verts[1].x) * (verts[2].y - verts[1].y) == (verts[0].y - verts[1].y) *
(verts[2].x - verts[1].x)){
          throw std::logic_error("It is not triangle");
     }
}

template<class T>
double triangle<T>::area() const {
  const T dx1 = verts[1].x - verts[0].x;
  const T dy1 = verts[1].y - verts[0].y;
  const T dx2 = verts[2].x - verts[0].x;
  const T dy2 = verts[2].y - verts[0].y;
  return std::abs(dx1 * dy2 - dy1 * dx2) * 0.5;
}

template<class T>
void triangle<T>::print(std::ostream& os) const {
     for(int i = 0; i < 3; ++i){
          os << '[' << verts[i] << ']';
          if(i + 1 != 3){
               os << ' ';
          }
     }
     os << '\n';
     return;
}

template<class T>
vertex<double> triangle<T>::center() const{
     double mid1 = (verts[0].x + verts[1].x + verts[2].x) / 3;
     double mid2 = (verts[0].y + verts[1].y + verts[2].y) / 3;
     return vertex<double>{mid1, mid2};
}
```

```
#endif // D_TRIANGLE_H_
```

rectangle.h

```
#ifndef D_RECTANGLE_H_
#define D_RECTANGLE_H_ 1

#include <algorithm>
#include <iostream>

#include "vertex.h"

template<class T>
struct rectangle{
private:
     vertex<T> top1, top2, top3, top4;
public:
     rectangle(std::istream& is){
          is >> top1.x >> top1.y >> top2.x >> top2.y >> top3.x >> top3.y >> top4.x >> top4.y;
          double scalar1 = scalar_mult(top2, top1, top3);
          double scalar2 = scalar_mult(top2, top1, top4);
          double scalar3 = scalar_mult(top3, top1, top4);
          double scalar01, scalar02, scalar03;
          if(scalar1 == 0){
               scalar01 = scalar_mult(top4, top2, top1);
               scalar02 = scalar_mult(top2, top4, top3);
               scalar03 = scalar_mult(top1, top3, top4);
               if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0)){
                    throw std::logic_error("It is not rectangle");
               }
          } else if(scalar2 == 0){
               scalar01 = scalar_mult(top1, top2, top3);
               scalar02 = scalar_mult(top1, top4, top3);
               scalar03 = scalar_mult(top2, top3, top4);
               if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0)){
                    throw std::logic_error("It is not rectangle");
               }
          } else if(scalar3 == 0){
               scalar01 = scalar_mult(top3, top2, top4);
               scalar02 = scalar_mult(top1, top4, top2);
               scalar03 = scalar_mult(top2, top3, top1);
               if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0)){
                    throw std::logic_error("It is not rectangle");
               }
          } else {
               throw std::logic_error("It is not rectangle");
          }
     }

     rectangle(vertex<T>& p1, vertex<T>& p2, vertex<T>& p3, vertex<T>& p4);
     vertex<double> center() const;
     double area() const;
     void print(std::ostream& os) const;
};

template<class T>
rectangle<T>::rectangle(vertex<T>& p1, vertex<T>& p2, vertex<T>& p3, vertex<T>& p4){
     top1.x = p1.x;
     top1.y = p1.y;
     top2.x = p2.x;
     top2.y = p2.y;
     top3.x = p3.x;
     top3.y = p3.y;
     top4.x = p4.x;
     top4.y = p4.y;
     double scalar1 = scalar_mult(top2, top1, top3);
     double scalar2 = scalar_mult(top2, top1, top4);
     double scalar3 = scalar_mult(top3, top1, top4);
     double scalar01, scalar02, scalar03;
     if(scalar1 == 0){
          scalar01 = scalar_mult(top4, top2, top1);
          scalar02 = scalar_mult(top2, top4, top3);
          scalar03 = scalar_mult(top1, top3, top4);
          if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0)){
               throw std::logic_error("It is not quadrate");
          }
```

```
     } else if(scalar2 == 0){
          scalar01 = scalar_mult(top1, top2, top3);
          scalar02 = scalar_mult(top1, top4, top3);
          scalar03 = scalar_mult(top2, top3, top4);
          if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0)){
               throw std::logic_error("It is not quadrate");
          }
     } else if(scalar3 == 0){
          scalar01 = scalar_mult(top3, top2, top4);
          scalar02 = scalar_mult(top1, top4, top2);
          scalar03 = scalar_mult(top2, top3, top1);
          if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0)){
               throw std::logic_error("It is not quadrate");
          }
     } else {
          throw std::logic_error("It is not quadrate");
     }
}


template<class T>
void rectangle<T>::print(std::ostream& os) const{
     os << "[" << top1 << "]" << " ";
     os << "[" << top2 << "]" << " ";
     os << "[" << top3 << "]" << " ";
     os << "[" << top4 << "]" << " ";
     os << '\n';
}

template<class T>
double rectangle<T>::area() const{
     double scalar1 = scalar_mult(top2, top1, top3);
     double scalar2 = scalar_mult(top2, top1, top4);
     double scalar3 = scalar_mult(top3, top1, top4);
     double mid1, mid2;
     if(scalar1 == 0){
          mid1 = segment_length(top1, top2);
          mid2 = segment_length(top1, top3);
     } else if(scalar2 == 0){
          mid1 = segment_length(top1, top2);
          mid2 = segment_length(top1, top4);
     } else if(scalar3 == 0){
          mid1 = segment_length(top1, top3);
          mid2 = segment_length(top1, top4);
     }
     return mid1 * mid2;
}

template<class T>
vertex<double> rectangle<T>::center() const{
     double scalar1 = scalar_mult(top2, top1, top3);
     double scalar2 = scalar_mult(top2, top1, top4);
     double scalar3 = scalar_mult(top3, top1, top4);
     double midx, midy;
     if(scalar1 == 0){
          midx = (top3.x + top2.x) * 0.5;
          midy = (top3.y + top2.y) * 0.5;
     } else if(scalar2 == 0){
          midx = (top4.x + top2.x) * 0.5;
          midy = (top4.y + top2.y) * 0.5;
     } else if(scalar3 == 0){
          midx = (top3.x + top4.x) * 0.5;
          midy = (top3.y + top4.y) * 0.5;
     }
     return vertex<double>{midx, midy};
}


#endif
```

quadrate.h

```
#ifndef D_QUADRATE_H_
#define D_QUADRATE_H_ 1

#include <algorithm>
```

```
#include <iostream>

#include "vertex.h"

template<class T>
struct quadrate{
private:
     vertex<T> top1, top2, top3, top4;
public:
     quadrate(std::istream& is){
          is >> top1.x >> top1.y >> top2.x >> top2.y >> top3.x >> top3.y >> top4.x >> top4.y;
          double scalar1 = scalar_mult(top2, top1, top3);
          double scalar2 = scalar_mult(top2, top1, top4);
          double scalar3 = scalar_mult(top3, top1, top4);
          double scalar01, scalar02, scalar03, scalar_diag;
          if(scalar1 == 0){
                scalar01 = scalar_mult(top4, top2, top1);
                scalar02 = scalar_mult(top2, top4, top3);
                scalar03 = scalar_mult(top1, top3, top4);
                scalar_diag = (top3.x - top2.x)*(top4.x - top1.x) + (top3.y - top2.y)*(top4.y -
top1.y);
                if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0)){
                     throw std::logic_error("It is not quadrate");
                }
          } else if(scalar2 == 0){
               scalar01 = scalar_mult(top1, top2, top3);
               scalar02 = scalar_mult(top1, top4, top3);
               scalar03 = scalar_mult(top2, top3, top4);
               scalar_diag = (top4.x - top2.x)*(top3.x - top1.x) + (top4.y - top2.y)*(top3.y -
top1.y);
               if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0)){
                    throw std::logic_error("It is not quadrate");
               }
          } else if(scalar3 == 0){
               scalar01 = scalar_mult(top3, top2, top4);
               scalar02 = scalar_mult(top1, top4, top2);
               scalar03 = scalar_mult(top2, top3, top1);
               scalar_diag = (top4.x - top3.x)*(top2.x - top1.x) + (top4.y - top3.y)*(top2.y -
top1.y);
               if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0)){
                    throw std::logic_error("It is not quadrate");
               }
          } else {
               throw std::logic_error("It is not quadrate");
          }
     }

     quadrate(vertex<T>& p1, vertex<T>& p2, vertex<T>& p3, vertex<T>& p4);
     vertex<double> center() const;
     double area() const;
     void print(std::ostream& os) const;
};

template<class T>
quadrate<T>::quadrate(vertex<T>& p1, vertex<T>& p2, vertex<T>& p3, vertex<T>& p4){
     top1.x = p1.x;
     top1.y = p1.y;
     top2.x = p2.x;
     top2.y = p2.y;
     top3.x = p3.x;
     top3.y = p3.y;
     top4.x = p4.x;
     top4.y = p4.y;
     double scalar1 = scalar_mult(top2, top1, top3);
     double scalar2 = scalar_mult(top2, top1, top4);
     double scalar3 = scalar_mult(top3, top1, top4);
     double scalar01, scalar02, scalar03, scalar_diag;
     if(scalar1 == 0){
          scalar01 = scalar_mult(top4, top2, top1);
          scalar02 = scalar_mult(top2, top4, top3);
          scalar03 = scalar_mult(top1, top3, top4);
          scalar_diag = (top3.x - top2.x)*(top4.x - top1.x) + (top3.y - top2.y)*(top4.y -
top1.y);
          if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0)){
               throw std::logic_error("It is not quadrate");
```

```
        }
    } else if(scalar2 == 0){
        scalar01 = scalar_mult(top1, top2, top3);
        scalar02 = scalar_mult(top1, top4, top3);
        scalar03 = scalar_mult(top2, top3, top4);
        scalar_diag = (top4.x - top2.x)*(top3.x - top1.x) + (top4.y - top2.y)*(top3.y -
top1.y);
        if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0)){
            throw std::logic_error("It is not quadrate");
        }
    } else if(scalar3 == 0){
        scalar01 = scalar_mult(top3, top2, top4);
        scalar02 = scalar_mult(top1, top4, top2);
        scalar03 = scalar_mult(top2, top3, top1);
        scalar_diag = (top4.x - top3.x)*(top2.x - top1.x) + (top4.y - top3.y)*(top2.y -
top1.y);
        if(!(scalar01 == 0 && scalar02 == 0 && scalar03 == 0 && scalar_diag == 0)){
            throw std::logic_error("It is not quadrate");
        }
    } else {
        throw std::logic_error("It is not quadrate");
    }
}

template<class T>
void quadrate<T>::print(std::ostream& os) const{
    os << "[" << top1 << "]" << " ";
    os << "[" << top2 << "]" << " ";
    os << "[" << top3 << "]" << " ";
    os << "[" << top4 << "]" << " ";
    os << '\n';
}

template<class T>
double quadrate<T>::area() const{
    double scalar1 = scalar_mult(top2, top1, top3);
    double scalar2 = scalar_mult(top2, top1, top4);
    double scalar3 = scalar_mult(top3, top1, top4);
    double mid;
    if(scalar1 == 0){
        mid = segment_length(top1, top2);
    } else if(scalar2 == 0){
        mid = segment_length(top1, top4);
    } else if(scalar3 == 0){
        mid = segment_length(top1, top3);
    }
    return mid * mid;
}

template<class T>
vertex<double> quadrate<T>::center() const{
    double scalar1 = scalar_mult(top2, top1, top3);
    double scalar2 = scalar_mult(top2, top1, top4);
    double scalar3 = scalar_mult(top3, top1, top4);
    double midx, midy;
    if(scalar1 == 0){
        midx = (top3.x + top2.x) * 0.5;
        midy = (top3.y + top2.y) * 0.5;
    } else if(scalar2 == 0){
        midx = (top4.x + top2.x) * 0.5;
        midy = (top4.y + top2.y) * 0.5;
    } else if(scalar3 == 0){
        midx = (top3.x + top4.x) * 0.5;
        midy = (top3.y + top4.y) * 0.5;
    }
    return vertex<double>{midx, midy};
}

#endif
```

main.cpp

```
#include <array>
#include <iostream>
#include <tuple>
```

```
#include "triangle.h"
#include "quadrate.h"
#include "rectangle.h"
#include "templates.h"


int main(){
     while(1){
          char com1[40];
          std::cin >> com1;
          if(strcmp(com1, "triangle") == 0){
               vertex<double> p1, p2, p3;
               std::cin >> p1 >> p2 >> p3;
               triangle<double> r_tr(p1, p2, p3);
               std::cout << area(r_tr) << std::endl;
               print(r_tr, std::cout);
               std::cout << "{" << center(r_tr) << "}" << std::endl;
               std::tuple<vertex<double>, vertex<double>, vertex<double>> f_tr{p1, p2, p3};
               std::cout << area(f_tr) << std::endl;
               print(f_tr, std::cout);
               std::cout << "{" << center(f_tr) << "}" << std::endl;

               std::array<vertex<double>, 3> f_t{p1, p2, p3};
               std::cout << area(f_t) << std::endl;
               print(f_t, std::cout);
               std::cout << "{" << center(f_t) << "}" << std::endl;
          } else if(strcmp(com1, "quadrate") == 0){
               vertex<double> p1, p2, p3, p4;
               std::cin >> p1 >> p2 >> p3 >> p4;
               quadrate<double> r_tr(p1, p2, p3, p4);
               std::cout << area(r_tr) << std::endl;
               print(r_tr, std::cout);
               std::cout << "{" << center(r_tr) << "}" << std::endl;
               std::tuple<vertex<double>, vertex<double>, vertex<double>, vertex<double>>
f_tr{p1, p2, p3, p4};
               std::cout << area(f_tr) << std::endl;
               print(f_tr, std::cout);
               std::cout << "{" << center(f_tr) << "}" << std::endl;

               std::array<vertex<double>, 4> f_t{p1, p2, p3, p4};
               std::cout << area(f_t) << std::endl;
               print(f_t, std::cout);
               std::cout << "{" << center(f_t) << "}" << std::endl;
          } else if(strcmp(com1, "rectangle") == 0){
               vertex<double> p1, p2, p3, p4;
               std::cin >> p1 >> p2 >> p3 >> p4;
               rectangle<double> r_tr(p1, p2, p3, p4);
               std::cout << area(r_tr) << std::endl;
               print(r_tr, std::cout);
               std::cout << "{" << center(r_tr) << "}" << std::endl;
               std::tuple<vertex<double>, vertex<double>, vertex<double>, vertex<double>>
f_tr{p1, p2, p3, p4};
               std::cout << area(f_tr) << std::endl;
               print(f_tr, std::cout);
               std::cout << "{" << center(f_tr) << "}" << std::endl;

               std::array<vertex<double>, 4> f_t{p1, p2, p3, p4};
               std::cout << area(f_t) << std::endl;
               print(f_t, std::cout);
               std::cout << "{" << center(f_t) << "}" << std::endl;
          } else if(strcmp(com1, "exit") == 0){
               break;
          } else {
               std::cout << "Incorrect command\n";
          }
     }
}
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 3.5)

project(oop_exercise_04)

add_executable(oop_exercise_04
```

```
                main.cpp
)

set_property(TARGET oop_exercise_04 PROPERTY CXX_STANDARD 17)
```

Ссылка на репозиторий на GitHub

https://github.com/AlexPshen/oop_exercise_04.git

Набор тестов

test_01.txt

triangle
0 0 0 1 2 0
exit

test_02.txt

quadrate
0 0 0 1 1 0 1 1
exit

test_03.txt

rectangle
0 0 0 2 1 0 1 2
exit

Результаты

test_01.txt
1
[0 0] [0 1] [2 0]
{0.66667, 0.33333}
1
[0 0] [0 1] [2 0]
{0.66667, 0.33333}
1
[0 0] [0 1] [2 0]
{0.66667, 0.33333}
test_02.txt
1
[0 0] [0 1] [1 0] [1 1]
{0.5, 0.5}
1
[0 0] [0 1] [1 0] [1 1]
{0.5, 0.5}
1
[0 0] [0 1] [1 0] [1 1]
{0.5, 0.5}

test_03.txt
2
[0 0] [0 2] [1 0] [1 2]
{0.5, 1}

2
[0 0] [0 2] [1 0] [1 2]
{0.5, 1}
2
[0 0] [0 2] [1 0] [1 2]
{0.5, 1}

Объяснение работы программы

На ввод подаются команда com1. Если com1 является:

- quadrate, то считываем 4 вершины
- print, то вводим вторую команду com2, которая может быть tops, center, square. Данная команда печатает соответственно вершины, центр, площадь фигур.
- delete, тогда вводим целое число id и удаляем фигуру по данному индексу
- exit, тогда выходим из программы

point center() const — вывод центра фигуры
double square() const — вывод площади фигуры
void print(std::ostream& os) const — вывод вершин фигуры

Вывод

В данной лабораторной работе были рассмотрены механизмы работы с метапрограммированием в C++. Метапрограммирование — это «программирование программ», то есть написание некой промежуточной программы, результатом которой будет некая часть другой программы. Вместо написания N одинаковых функций для разных типов, мы пишем шаблон, и компилятор сам соберет нам эти N функций.