Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Объектно-ориентированное программирование»
III Семестр

Задание 4
Вариант 24
Основы метапрограммирования

Студент:	Синявский А.В
Группа:	М80-208Б-18
Преподаватель:	Журавлёв А.А
Оценка:	
Дата:	

Москва 2019

1. Код программы на языке C++
1.1 vertex.h

```
#ifndef OOP_EXERCISE_04_VERTEX_H
#define OOP_EXERCISE_04_VERTEX_H

#include <iostream>
#include <type_traits>
#include <cmath>


template<class T>
struct vertex {
    T x;
    T y;
    vertex<T>& operator=(vertex<T> A);
};

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, vertex<T> p) {
    os << '(' << p.x << ' ' << p.y << ')';
    return os;
}

template<class T>
vertex<T> operator+(const vertex<T>& A, const vertex<T>& B) {
    vertex<T> res;
    res.x = A.x + B.x;
    res.y = A.y + B.y;
    return res;
}

template<class T>
vertex<T>& vertex<T>::operator=(const vertex<T> A) {
    this->x = A.x;
    this->y = A.y;
    return *this;
}

template<class T>
vertex<T> operator+=(vertex<T> &A, const vertex<T> &B) {
    A.x += B.x;
    A.y += B.y;
    return A;
}
```

```
template<class T>
vertex<T> operator/=(vertex<T>& A, const double B) {
   A.x /= B;
   A.y /= B;
}

template<class T>
double vert_length(vertex<T>& A, vertex<T>& B) {
   double res = sqrt( pow(B.x - A.x, 2) + pow(B.y - A.y, 2) );
   return res;
}

template<class T>
struct is_vertex : std::false_type {};

template<class T>
struct is_vertex<vertex<T>> : std::true_type {};


#endif //OOP_EXERCISE_04_VERTEX_H
```

1.2 classes.h

```
#ifndef OOP_EXERCISE_04_CLASSES_H
#define OOP_EXERCISE_04_CLASSES_H

#include "vertex.h"
#include <type_traits>
#include <iostream>

template <class T>
class Triangle {
public:
   vertex<T> dots[3];
   int size = 3;
   explicit Triangle<T>(std::istream& is) {
      for (auto & dot : dots) {
         is >> dot;
      }
   }
};

template <class T>
class Square {
public:
   vertex<T> dots[4];
   int size = 4;
   explicit Square<T>(std::istream& is) {
      for (auto & dot : dots) {
         is >> dot;
      }
      if (!is_proper_square(dots)) {
```

```
            throw std::logic_error("square is not squarish enouth");
        }
    }
};

template<class T>
bool is_proper_square (vertex<T> dots[4]) {
    bool equasion1 = ((vert_length(dots[0], dots[1]) == vert_length(dots[1], dots[2])) &&
(vert_length(dots[2], dots[3]) == vert_length(dots[3], dots[0])) &&
                    (vert_length(dots[0], dots[1]) == vert_length(dots[3], dots[0])) );
    bool equasion2 =  (((dots[0].x - dots[1].x) * (dots[2].x - dots[1].x) + (dots[0].y - dots[1].y) *
(dots[2].y - dots[1].y)) == 0);
    return equasion1 && equasion2;
}

template <class T>
class Octagon {
public:
    vertex<T> dots[8];
    int size = 8;
    explicit Octagon<T>(std::istream& is) {
        for (auto & dot : dots) {
            is >> dot;
        }
    }
};

#endif //OOP_EXERCISE_04_CLASSES_H
```

1.3 templates.h

```
#ifndef OOP_LAB4_V2_FIGURES_H
#define OOP_LAB4_V2_FIGURES_H

#include <tuple>
#include <type_traits>
#include <cassert>

#include "vertex.h"
#include "classes.h"

//==============================ASSISTANCE
TEMPLATES=============================================//

template<class T, class = void>
struct has_dots : std::false_type {};

template<class T>
struct has_dots<T, std::void_t<decltype(std::declval<const T&>().dots)>> : std::true_type {};

template<class T>
struct is_figurelike_tuple : std::false_type {};
```

```
template<class Head, class... Tail>
struct is_figurelike_tuple<std::tuple<Head, Tail...>> :
        std::conjunction<is_vertex<Head>, std::is_same<Head, Tail>...> {};


//
=======================================PRINT============================
=======================//

template<size_t Id, class T>
void tuple_print(const T& object, std::ostream& os) {
   if constexpr (Id >= std::tuple_size<T>::value) {
   } else {
      os << std::get<Id>(object) << " ";
      tuple_print<Id + 1>(object, os);
   }
}

template <class T>
void printout(const T& object, std::ostream& os) {
    if constexpr (has_dots<T>::value) {
        for (auto dot : object.dots) {
            os << dot << " ";
        }
    } else if constexpr (is_figurelike_tuple<T>::value) {
        tuple_print<0>(object, os);
    } else {
        throw std::logic_error("ERROR! Perhaps tuple is incorrect");
    }
}

//
====================================CENTER============================
==================//

template<size_t Id, class T>
vertex<double> tuple_center(const T& object) {
    if constexpr (Id >= std::tuple_size<T>::value) {
        return vertex<double> {0, 0};
    } else {
        vertex<double> res = std::get<Id>(object);
        return res + tuple_center<Id+1>(object);
    }
}

template <class T>
vertex<double> center(const T& object) {
    vertex<double> res{0.0, 0.0};
    int i = 0;
    if constexpr (has_dots<T>::value) {
        for (auto dot : object.dots) {
```

```
            res += dot;
            ++i;
        }
        res /= i;
        return res;
    } else if constexpr (is_figurelike_tuple<T>::value) {
        res = tuple_center<0>(object);
        res /= std::tuple_size_v<T>;
        return res;
    } else {
        throw std::logic_error ("ERROR! Perhaps tuple is incorrect");
    }
}

//
====================================AREA==============================
================//

template<size_t Id, class T>
double tuple_area(const T& object) {
    if constexpr (Id >= std::tuple_size<T>::value - 1) {
        return 0.0;
    } else {
        double res = (std::get<Id>(object).x * std::get<Id+1>(object).y) -
                (std::get<Id+1>(object).x * std::get<Id>(object).y);
        return res + tuple_area<Id+1>(object);
    }
}

template <class T>
double area(const T& object) {
    double res = 0.0;
    if constexpr (has_dots<T>::value) {
        for (int i = 0; i < object.size-1; ++i) {
            res += (object.dots[i].x * object.dots[i+1].y) - (object.dots[i+1].x * object.dots[i].y);
        }
        res += (object.dots[object.size-1].x * object.dots[0].y) - (object.dots[0].x *
object.dots[object.size-1].y);
        return std::abs(res) / 2;
    } else if constexpr (is_figurelike_tuple<T>::value) {
        res = tuple_area<0>(object);
        res += (std::get<std::tuple_size<T>::value - 1>(object).x * std::get<0>(object).y) -
(std::get<0>(object).x * std::get<std::tuple_size<T>::value - 1>(object).y);
        return std::abs(res) / 2;
    } else {
        throw std::logic_error ("ERROR! Perhaps tuple is incorrect");
    }
}

//================================================================//

#endif //OOP_LAB4_V2_FIGURES_H
```

1.4 process.h

```
#ifndef OOP_EXERCISE_04_PROCESS_H
#define OOP_EXERCISE_04_PROCESS_H

template<class T>
void process(std::istream& is, std::ostream& os) {
    if constexpr (has_dots<T>::value) {
        T object(is);
        printout(object, os);
        os << std::endl;
        os << area(object) << std::endl;
        os << center(object) << std::endl;
    } else if constexpr (is_figurelike_tuple<T>::value) {
        size_t s;
        os << "enter number of angles: ";
        is >> s;
        switch(s) {
            case 3: {
                vertex<double> fig[3];
                for (auto & i : fig) {
                    is >> i;
                }
                auto[a, b, c] = fig;
                auto object = std::make_tuple(a, b, c);
                printout(object, os);
                os << std::endl;
                os << area(object) << std::endl;
                os << center(object) << std::endl;
                break;
            }
            case 4: {
                vertex<double> fig[4];
                for (auto & i : fig) {
                    is >> i;
                }
                if (!is_proper_square(fig)) {
                    throw std::logic_error ("square is not squarish enough");
                }
                auto[a, b, c, d] = fig;
                auto object = std::make_tuple(a, b, c, d);
                printout(object, os);
                os << std::endl;
                os << area(object) << std::endl;
                os << center(object) << std::endl;
                break;
            }
            case 8: {
                vertex<double> fig[8];
                for (auto & i : fig) {
```

```
            is >> i;
        }
        auto[a, b, c, d, e, f, g, h] = fig;
        auto object = std::make_tuple(a, b, c, d, e, f, g, h);
        printout(object, os);
        os << std::endl;
        os << area(object) << std::endl;
        os << center(object) << std::endl;
        break;
      }
      default:
        throw std::logic_error("incorrect number of angles, try 3, 4 or 8");

    }


  } else {
    throw std::logic_error ("trying to process something stupid. Tuple must be like a figure");
  }
}

#endif //OOP_EXERCISE_04_PROCESS_H
```

1.5 main.cpp

```
#include <iostream>
#include "templates.h"
#include "vertex.h"
#include "process.h"


int main() {
  char option = '0';
  while (option != 'q') {
    std::cout << "choose option (m for man, q to quit): ";
    std::cin >> option;
    switch (option) {
      default:
        std::cout << "no such option, try m for man" << std::endl;
        break;
      case 'q':
        return 0;
      case 'm': {
        std::cout << "1) process triangle" << '\n'
        << "2) process square" << '\n'
        << "3) process octagon" << '\n'
        << "4) process tuple" <<  '\n'
        << "5) place for your tests" << std::endl;
        break;
      }
      case '1': {
```

```
            process<Triangle<double>>(std::cin, std::cout);
            break;
        }
        case '2': {
            process<Square<double>>(std::cin, std::cout);
            break;
        }
        case '3': {
            process<Octagon<double>>(std::cin, std::cout);
            break;
        }
        case '4': {
            process<std::tuple<vertex<double>>>(std::cin, std::cout);
            break;
        }
        case '5' : {
//====================PLACE-FOR-CUSTOM-
TESTS============================//
            auto tup = std::make_tuple(vertex<double>{0.0, 0.0},
                vertex<int>{0, 2}, vertex<float>{2.0, 0.0});
            printout(tup, std::cout);

//
=========================================================================
//
            break;
        }
    }
  }

  return 0;
}
```

2. Ссылка на репозиторий на GitHub

https://github.com/Siegmeyer1/oop_exercise_04

3. Набор тестов

```
1)
m
1
0 0     2 0     0 2
1
2 2     2 0     0 2
2
2 0     1 3     -2 2    -1 -1
3
2 0    4 0    6 1    6 3    4 4    2 4    0 3    0 1
q
```

2)
5

4. Результаты тестов

1)
anri@andrew-HP-250-G6:~/Documents/Github_repositories/OOP_lab4/build$ ./oop_exercise_04
choose option (m for man, q to quit): m
1) process triangle
2) process square
3) process octagon
4) process tuple
5) place for your tests
choose option (m for man, q to quit): 1
0 0 2 0 0 2
(0 0) (2 0) (0 2)
2
(0.666667 0.666667)
choose option (m for man, q to quit): 1
2 2 2 0 0 2
(2 2) (2 0) (0 2)
2
(1.33333 1.33333)
choose option (m for man, q to quit): 2
2 0 1 3 -2 2 -1 -1
(2 0) (1 3) (-2 2) (-1 -1)
10
(0 1)
choose option (m for man, q to quit): 3
2 0 4 0 6 1 6 3 4 4 2 4 0 3 0 1
(2 0) (4 0) (6 1) (6 3) (4 4) (2 4) (0 3) (0 1)
20
(3 2)
choose option (m for man, q to quit): q
anri@andrew-HP-250-G6:~/Documents/Github_repositories/OOP_lab4/build$

2)
anri@andrew-HP-250-G6:~/Documents/Github_repositories/OOP_lab4/build$ ./oop_exercise_04
choose option (m for man, q to quit): 5
terminate called after throwing an instance of 'std::logic_error'
  what(): ERROR! Perhaps tuple is incorrect
Aborted (core dumped)
anri@andrew-HP-250-G6:~/Documents/Github_repositories/OOP_lab4/build$

5. Объяснение работы программы

В файле classes.h описываются фигуры заданного варианта.

Файл templates.h можно логически разделить на блоки:

Вспомогательные шаблоны

Шаблоны с использованием type_traits, определяющие, является ли объект классом фигуры или тьюплом, похожим на фигуру по структуре.

Вывод

Содержит шаблоны функций вывода. Функция printout компилируется по-разному в зависимости от типа входной переменной. tuple_print – функция для вывода тьюпла, вызываемая printout-ом.

Центр

Аналогично вводу содержит функции подсчёта геометрического центра. В обоих случаях (фигура и тьюпл) считается просто среднее значение координат всех точек.

Площадь

Площадь считается по формуле Гаусса (метод шнуровки) для вычисления площади произвольного многоугольника без пересечений.

Файл process.h содержит шаблон функции, которая в качестве шаблона принимает тип необходимой фигуры, осуществляет ввод указанного типа и применения к нему всех необходимых функций. Если тип – тьюпл, то функция запрашивает информацию о количестве углов фигуры для её корректной обработки.

main.cpp представляет собой интерфейс для обработки запрашиваемых типов данных. По сути, функция спрашивает у пользователя, какую фигуру он хочет обработать, и вызывает функцию process для фигуры запрашиваемого типа.

Вывод

Проделав работу, я изучил некоторые возможности, которые предоставляет библиотека type_traits, расширил свои знания о процессе компиляции программы и как можно им управлять, научился писать шаблонные функции. Научился кидать ошибки через throw.