Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Отчёт по Лабораторной работе №4
“Наследование, Полиморфизм“
по курсу “Объектно-Объективное Программирование“
III Семестр

Студент:	Катермин В.С.
Группа:	М8О-208Б-18
Преподаватель:	Журавлёв А.А.
Оценка:	
Дата:	**11.11.19**

1. **Тема**: Основы метапрограммирования в C++.
2. **Код программы:**

vertex.h

```
#ifndef D_VERTEX_H_
#define D_VERTEX_H_ 1

#include <iostream>

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;
}

#endif // D_VERTEX_H_
```

templates.h

```
#ifndef D_TEMPLATES_H_
#define D_TEMPLATES_H_ 1

#include <tuple>
#include <type_traits>

#include "vertex.h"

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

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

template<class Type, size_t SIZE>
struct is_figurelike_tuple<std::array<Type, SIZE>> :
   is_vertex<Type> {};

template<class T>
inline constexpr bool is_figurelike_tuple_v =
   is_figurelike_tuple<T>::value;

//center
```

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

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

template<class T>
inline constexpr bool has_center_method_v =
       has_center_method<T>::value;

template<class T>
std::enable_if_t<has_center_method_v<T>, vertex<double>>
center(const T& figure) {
   return figure.center();
}

template<class T>
inline constexpr const int tuple_size_v = std::tuple_size<T>::value;

template<size_t ID, class T>
vertex<double> sngl_center(const T& t) {
   vertex<double> v;
   v.x = std::get<ID>(t).x;
   v.y = std::get<ID>(t).y;
   v.x = v.x / std::tuple_size_v<T>;
   v.y = v.y / std::tuple_size_v<T>;
   return v;
}

template<size_t ID, class T>
vertex<double> rcrsv_center(const T& t) {
   if constexpr (ID < std::tuple_size_v<T>){
      return  sngl_center<ID>(t) + rcrsv_center<ID+1>(t);
   } else {
      vertex<double> v;
      v.x = 0;
      v.y = 0;
      return v;
   }
}

template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, vertex<double>>
center(const T& fake) {
   return rcrsv_center<0>(fake);
}

//area
template<class T, class = void>
struct has_area_method : std::false_type {};

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

template<class T>
inline constexpr bool has_area_method_v =
       has_area_method<T>::value;

template<class T>
std::enable_if_t<has_area_method_v<T>, double>
```

```
area(const T& figure) {
   return figure.area();
}

template<size_t ID, class T>
double sngl_center(const T& t) {
   const auto& a = std::get<0>(t);
   const auto& b = std::get<ID - 1>(t);
   const auto& c = std::get<ID>(t);
   const double dx1 = b.x - a.x;
   const double dy1 = b.y - a.y;
   const double dx2 = c.x - a.x;
   const double dy2 = c.y - a.y;
   return std::abs(dx1 * dy2 - dy1 * dx2) * 0.5;
}

template<size_t ID, class T>
double rcrsv_center(const T& t) {
   if constexpr (ID < std::tuple_size_v<T>){
      return sngl_center<ID>(t) + rcrsv_center<ID + 1>(t);
   }
   return 0;
}

//print
template<class T, class = void>
struct has_print_method : std::false_type {};

template<class T>
struct has_print_method<T,
   std::void_t<decltype(std::declval<const T>().print(std::cout))>> :
      std::true_type {};

template<class T>
inline constexpr bool has_print_method_v =
  has_print_method<T>::value;

template<class T>
std::enable_if_t<has_print_method_v<T>, void>
   print (const T& figure,std::ostream& os) {
      return figure.print(os);
}

template<size_t ID, class T>
void sngl_print(const T& t) {
   std::cout << std::get<ID>(t);
   return ;
}

template<size_t ID, class T>
void rcrsv_print(const T& t) {
   if constexpr (ID < std::tuple_size_v<T>){
      sngl_print<ID>(t);
      rcrsv_print<ID+1>(t);
      return ;
   }
   return;
}

#endif // D_TEMPLATES_H_
```

square.h

```
#ifndef D_SQUARE_H
#define D_SQUARE_H

#include "figure.h"

struct square : figure{
  square(const point& p1, const point& p2, const point& p3, const point& p4);
    square(std::istream& is);

    point center() override;
    double area() override;
    void print(std::ostream& os) override;

private:
    point p1_, p2_, p3_, p4_;
};

#endif //D_SQUARE_H
```

rectangle.h

```
#ifndef D_RECTANGLE_H
#define D_RECTANGLE_H

#include "figure.h"

struct rectangle : figure{
  rectangle(const point& p1, const point& p2, const point& p3, const point& p4);
    rectangle(std::istream& is);

    point center() override;
    double area() override;
    void print(std::ostream& os) override;

private:
    point p1_, p2_, p3_, p4_;
};

#endif //D_RECTANGLE_H
```

trapeze.h

```
#ifndef D_TRAPEZE_H
#define D_TRAPEZE_H

#include "figure.h"
#include <iostream>

struct trapeze : figure{
  trapeze(const point& p1, const point& p2, const point& p3, const point& p4);
    trapeze(std::istream& is);

    point center() override;
    double area() override;
    void print(std::ostream& os) override;

private:
    point p1_, p2_, p3_, p4_;
};

#endif //D_TRAPEZE_H
```

lab4.cpp

```
#include <iostream>
#include <vector>

#include "square.h"
#include "rectangle.h"
#include "trapeze.h"

int main(){
    std::vector<figure*> figures;
    for (;;){
        int command;
        std::cin >> command;
        if (command == 0){
            break;
        }else if (command == 1){
            int figure_type;
            std::cin >> figure_type;
            figure* ptr;
            if (figure_type == 0){
                ptr = new square(std::cin);
            }else if (figure_type == 1){
                ptr = new rectangle(std::cin);
            } else {
             ptr = new trapeze(std::cin);
            }
            figures.push_back(ptr);
        }else if (command == 2){
            int id;
            std::cin >> id;
            delete figures[id];
            figures.erase(figures.begin() + id);
        }else if (command == 3){
            std::cout << "Centers:\n";
            for (figure* ptr: figures){
                std::cout << ptr->center() << std::endl;
            }
        }else if (command == 4){
            std::cout << "Areas:\n";
            for (figure* ptr: figures){
                std::cout << ptr->area() << std::endl;
            }
        }else if (command == 5){
            std::cout << "Figures:\n";
            for (figure* ptr: figures){
                ptr->print(std::cout);
                std::cout << std::endl;
            }
        }
    }
    for (figure* ptr: figures){
        delete ptr;
    }
}
```

CMakeLists.txt

```
project(lab4)

set(CMAKE_CXX_STANDARD 17)

add_executable(lab4
  ./lab4.cpp)

set(CMAKE_CXX_FLAGS
```

"${CMAKE_CXX_FLAGS} -Wall -Wextra")
"${CMAKE_CXX_FLAGS} -Wall -Wextra")

3. **Ссылка на репозиторий:**
   https://github.com/GitGood2000/oop_exercise_04

4. **Набор testcases:**

test_00.test
1
0 2 2 2 2 0 0 0
2
1 7 4 7 4 2 1 2
3
2 4 6 4 7 1 1 1
0

test_00.result
Square [0 2] [2 2] [2 0] [0 0]
Center: [1 1]
Area: 4
Rectangle [1 7] [4 7] [4 2] [1 2]
Center: [2.5 4.5]
Area: 15
Trapeze [2 4] [6 4] [7 1] [1 1]
Center: [4 2.5]
Area: 15

test_01.test
1
0 2 2 3 3 1 1 0
1
1 3 4 6 7 3 4 0
2
0 2 4 4 5 2 1 0
2
1 3 7 5 8 2 2 0
3
0 3 3 5 6 5 0 1
3
1 5 7 3 5 1 2 2
0

test_01.result
Square [0 2] [2 3] [3 1] [1 0]
Center: [1.5 1.5]
Area: 5
Square [1 3] [4 6] [7 3] [4 0]
Center: [4 3]
Area: 18
Rectangle [0 2] [4 4] [5 2] [1 0]
Center: [2.5 2]
Area: 10
Rectangle [1 3] [7 5] [8 2] [2 0]
Center: [4.5 2.5]
Area: 20
Trapeze [0 3] [3 5] [6 5] [0 1]
Center: [2.25 3.5]
Area: 7.5
Trapeze [1 5] [7 3] [5 1] [2 2]
Center: [3.75 2.75]
Area: 12

test_02.test
1
-1 1 0 2 1 1 0 0
2
-1 1 1 3 2 2 0 0
3
-1 1 0 2 2 2 0 0
0

test_02.result
Square [-1 1] [0 2] [1 1] [0 0]
Center: [0 1]
Area: 2
Rectangle [-1 1] [1 3] [2 2] [0 0]
Center: [0.5 1.5]
Area: 4
Trapeze [-1 1] [0 2] [2 2] [0 0]
Center: [0.25 1.25]
Area: 3

5. **Результаты выполнения тестов:**

user@PSB133S01ZFH:~/3sem_projects/oop_exercise_04/tests$ bash test.sh ../build/lab4
Test test_00.test: SUCCESS
Test test_01.test: SUCCESS
Test test_02.test: SUCCESS

6. **Объяснение результатов работы программы:**
   1) Программа выполняет определённые действия по введённым командам:
      A) 0 — Выход из программы;
      B) 1,2,3 — Создание фигуры(Квадрат, Прямоугольник, Трапеция соответственно), получение вершин через ввод, проверка, вывод данных вершин, вычисление центра и площади;
   2) Шаблонная функция print() печатает координаты всех точек данной фигуры или кортежа. Она определена для моих фигур и tuple. Во втором случае все дело вычисляется рекурсивно.
   3) Функция center() возвращает точку с x –деление суммы иксов всех точек данной фигуры на их количество, y – аналогично x. Она определена для моих фигур и tuple. Во втором случае все дело вычисляется рекурсивно;
   4) Функция area() вычисляет площадь данной фигуры или совокупности точек в кортеже в зависимости от типа фигуры () и возвращает это значение.

7. **Вывод:** 1) Ознакомились с шаблонами в C++ и усвоили навык работы с ними; 2) Написана программа, производящая операции с помощью шаблонов.