Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

I I I семестр

Задание 4: «Основы метапрограммирования»

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №19 |
| Студент: | Овечкин Виталий Андреевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 09.01.2020 |

Москва, 2019

1. **Тема**: Основы метапрограммирования
2. **Цель работы**: Изучение основ работы с шаблонами в C++
3. **Задание** (*вариант № 19* ):

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.

Созздать набор шаблонов, реализующий функции:

* Вычисление геометрического центра фигуры
* Вывод в стандартный поток std::cout координат вершин фигуры
* Вычисление площади фигуры

Параметром шаблона должен являться тип класса фигуры. Помимо самого класса фигуры, шаблонная функция должна уметь работать с tuple.

Фигуры (Вариант 3):

Прямоугольник,трапеция, ромб.

1. **Адрес репозитория на GitHub** <https://github.com/vitalouivi/oop_exercise_0>4
2. **Код программы на С++**

main.cpp

|  |
| --- |
| #include <iostream> |
|  | #include <tuple> |
|  | #include "point.h" |
|  | #include "trapezoid.h" |
|  | #include "rectangle.h" |
|  | #include "rhombus.h" |
|  | #include "templates.h" |
|  |  |
|  | template<class T> |
|  | void running(std::istream& is, std::ostream& os) { |
|  | if constexpr (is\_figurelike\_tuple<T>::value) { |
|  | int ang; |
|  | std::cout << "Input number of angles" << std::endl; |
|  | std::cin >> ang; |
|  | if (ang == 4) { |
|  | point<double> A, B, C, D; |
|  | is >> A >> B >> C >> D; |
|  | auto object = std::make\_tuple(A, B, C, D); |
|  | print(os, object); |
|  | os << area(object) << std::endl; |
|  | os << center(object) << std::endl; |
|  | }else if(ang == 3){ |
|  | point<double> A, B, C; |
|  | is >> A >> B >> C; |
|  | auto object = std::make\_tuple(A, B, C); |
|  | print(os, object); |
|  | os << area(object) << std::endl; |
|  | os << center(object) << std::endl; |
|  | } |
|  | }else { |
|  | T object(is); |
|  | print(os, object); |
|  | os << '\n' << area(object) << std::endl; |
|  | os << center(object) << std::endl; |
|  | } |
|  | }  //additional classes  class Names {  public:  std::string first;  std::string last;  };  class WrongNames {  public:  int first;  int last;  }; |
|  |  |
|  | int main() { |
|  | char obj\_type; |
|  | std::cout << "Input figure type: 1 - trapezoid, 2 - rhombus, 3 - rectangle, 4 - tuple, 5 - to check additional task, or 'q' to quit" << std::endl; |
|  | while (std::cin >> obj\_type){ |
|  | if(obj\_type == '4') { |
|  | running<std::tuple<point<double>>> (std::cin, std::cout); |
|  | }else if(obj\_type == '1'){ |
|  | running<trapezoid<double>>(std::cin, std::cout); |
|  | }else if(obj\_type == '2'){ |
|  | running<rhombus<double>>(std::cin, std::cout); |
|  | }else if(obj\_type == '3'){ |
|  | running<rectangle<double>>(std::cin, std::cout); |
|  | } |
|  | else if (obj\_type == '5') { |
|  | //additional task проверить являются ли поля класса first и last типа string |
|  | std::cout << HasFL<Names>::value << std::endl; |
|  | std::cout << HasFL<WrongNames>::value << std::endl; |
|  | return 0; |
|  | } |

point.h

|  |
| --- |
| #ifndef POINT\_H\_ |
|  | #define POINT\_H\_ |
|  |  |
|  | #include <iostream> |
|  |  |
|  | template<class T> |
|  | struct point { |
|  | T x; |
|  | T y; |
|  | }; |
|  |  |
|  | template<class T> |
|  | point<T> operator+(const point<T>& A, const point<T>& B) { |
|  | point<T> res; |
|  | res.x = A.x + B.x; |
|  | res.y = A.y + B.y; |
|  | return res; |
|  | } |
|  |  |
|  | template<class T> |
|  | point<T> operator/=(point<T>& A, const double B) { |
|  | A.x /= B; |
|  | A.y /= B; |
|  | return A; |
|  | } |
|  |  |
|  | template<class T> |
|  | std::istream& operator>> (std::istream& is, point<T>& p) { |
|  | is >> p.x >> p.y; |
|  | return is; |
|  | } |
|  |  |
|  | template<class T> |
|  | std::ostream& operator<< (std::ostream& os, const point<T>& p) { |
|  | os << '[' << p.x << ' ' << p.y << ']'; |
|  | return os; |
|  | } |
|  |  |
|  | #endif |

rhombus.h

|  |
| --- |
| #ifndef RHOMBUS\_H\_ |
|  | #define RHOMBUS\_H\_ |
|  | #include <iostream> |
|  | #include <cmath> |
|  | #include "point.h" |
|  |  |
|  |  |
|  | template<class T> |
|  | struct rhombus { |
|  | point<T> points[4]; |
|  | rhombus(std::istream& is); |
|  | double area() const; |
|  | point<T> center() const; |
|  | void print(std::ostream& os) const; |
|  | }; |
|  |  |
|  | template<class T> |
|  | rhombus<T>::rhombus(std::istream& is) { |
|  | for(int i = 0; i < 4; ++i){ |
|  | is >> points[i]; |
|  | } |
|  | double a, b, c, d; |
|  | a = sqrt((points[1].x - points[0].x) \* (points[1].x - points[0].x) + (points[1].y - points[0].y) \* (points[1].y - points[0].y)); |
|  | b = sqrt((points[2].x - points[1].x) \* (points[2].x - points[1].x) + (points[2].y - points[1].y) \* (points[2].y - points[1].y)); |
|  | c = sqrt((points[2].x - points[3].x) \* (points[2].x - points[3].x) + (points[2].y - points[3].y) \* (points[2].y - points[3].y)); |
|  | d = sqrt((points[3].x - points[0].x) \* (points[3].x - points[0].x) + (points[3].y - points[0].y) \* (points[3].y - points[0].y)); |
|  | if(a != b || a != c || a != d) |
|  | throw std::logic\_error("It`s not a rhombus"); |
|  | } |
|  |  |
|  | template<class T> |
|  | double rhombus<T>::area() const { |
|  | const T d1 = sqrt((points[0].x - points[2].x) \* (points[0].x - points[2].x) + (points[0].y - points[2].y) \* (points[0].y - points[2].y)); |
|  | const T d2 = sqrt((points[1].x - points[3].x) \* (points[1].x - points[3].x) + (points[1].y - points[3].y) \* (points[1].y - points[3].y)); |
|  | return d1 \* d2 / 2; |
|  | } |
|  |  |
|  | template<class T> |
|  | point<T> rhombus<T>::center() const { |
|  | point<T> res; |
|  | res.x = (points[0].x + points[1].x + points[2].x + points[3].x) / 4; |
|  | res.y = (points[0].y + points[1].y + points[2].y + points[3].y) / 4; |
|  | return res; |
|  | } |
|  |  |
|  | template<class T> |
|  | void rhombus<T>::print(std::ostream& os) const { |
|  | for(int i = 0; i < 4; ++i){ |
|  | os << points[i]; |
|  | if(i + 1 != 4){ |
|  | os << ' '; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | #endif |

rectangle.h

|  |
| --- |
| #ifndef RECTANGLE\_H\_ |
|  | #define RECTANGLE\_H\_ |
|  | #include <iostream> |
|  | #include "point.h" |
|  | #include <cmath> |
|  |  |
|  | template<class T> |
|  | struct rectangle { |
|  | point<T> points[4]; |
|  | rectangle(std::istream& is); |
|  | double area() const; |
|  | point<T> center() const; |
|  | void print(std::ostream& os) const; |
|  | }; |
|  |  |
|  | template<class T> |
|  | rectangle<T>::rectangle(std::istream& is) { |
|  | for(int i = 0; i < 4; ++i){ |
|  | is >> points[i]; |
|  | } |
|  | double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB; |
|  | a = sqrt((points[1].x - points[0].x) \* (points[1].x - points[0].x) + (points[1].y - points[0].y) \* (points[1].y - points[0].y)); |
|  | b = sqrt((points[2].x - points[1].x) \* (points[2].x - points[1].x) + (points[2].y - points[1].y) \* (points[2].y - points[1].y)); |
|  | c = sqrt((points[2].x - points[3].x) \* (points[2].x - points[3].x) + (points[2].y - points[3].y) \* (points[2].y - points[3].y)); |
|  | d = sqrt((points[3].x - points[0].x) \* (points[3].x - points[0].x) + (points[3].y - points[0].y) \* (points[3].y - points[0].y)); |
|  | d1 = sqrt((points[1].x - points[3].x) \* (points[1].x - points[3].x) + (points[1].y - points[3].y) \* (points[1].y - points[3].y)); |
|  | d2 = sqrt((points[2].x - points[0].x) \* (points[2].x - points[0].x) + (points[2].y - points[0].y) \* (points[2].y - points[0].y)); |
|  | ABC = (a \* a + b \* b - d2 \* d2) / 2 \* a \* b; |
|  | BCD = (b \* b + c \* c - d1 \* d1) / 2 \* b \* c; |
|  | CDA = (d \* d + c \* c - d2 \* d2) / 2 \* d \* c; |
|  | DAB = (a \* a + d \* d - d1 \* d1) / 2 \* a \* d; |
|  | if(ABC != BCD || ABC != CDA || ABC != DAB) |
|  | throw std::logic\_error("It`s not a rectangle"); |
|  | } |
|  |  |
|  | template<class T> |
|  | double rectangle<T>::area() const { |
|  | const T a = sqrt((points[1].x - points[0].x) \* (points[1].x - points[0].x) + (points[1].y - points[0].y) \* (points[1].y - points[0].y)); |
|  | const T b = sqrt((points[2].x - points[1].x) \* (points[2].x - points[1].x) + (points[2].y - points[1].y) \* (points[2].y - points[1].y)); |
|  | return a \* b; |
|  | } |
|  |  |
|  | template<class T> |
|  | point<T> rectangle<T>::center() const { |
|  | point<T> res; |
|  | res.x = (points[0].x + points[1].x + points[2].x + points[3].x) / 4; |
|  | res.y = (points[0].y + points[1].y + points[2].y + points[3].y) / 4; |
|  | return res; |
|  | } |
|  |  |
|  | template<class T> |
|  | void rectangle<T>::print(std::ostream& os) const { |
|  | for(int i = 0; i < 4; ++i){ |
|  | os << points[i]; |
|  | if(i + 1 != 4){ |
|  | os << ' '; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | #endif |

trapezoid.h

|  |
| --- |
| #ifndef TRAPEZOID\_H\_ |
|  | #define TRAPEZOID\_H\_ |
|  | #include <iostream> |
|  | #include <cmath> |
|  | #include "point.h" |
|  |  |
|  |  |
|  | template<class T> |
|  | struct trapezoid { |
|  | point<T> points[4]; |
|  | trapezoid(std::istream& is); |
|  | double area() const; |
|  | point<T> center() const; |
|  | void print(std::ostream& os) const; |
|  | }; |
|  |  |
|  | template<class T> |
|  | trapezoid<T>::trapezoid(std::istream& is) { |
|  | for(int i = 0; i < 4; ++i){ |
|  | is >> points[i]; |
|  | } |
|  | if((points[2].y - points[1].y) / (points[2].x - points[1].x) != (points[3].y - points[0].y) / (points[3].x - points[0].x)) |
|  | throw std::logic\_error("It`s not a trapezoid"); |
|  | } |
|  |  |
|  | template<class T> |
|  | double trapezoid<T>::area() const { |
|  |  |
|  | return 0.5 \* std::abs( points[0].x \* points[1].y + points[1].x \* points[2].y + points[2].x \* points[3].y + points[3].x \* points[0].y - points[1].x \* points[0].y - points[2].x \* points[1].y - points[3].x \* points[2].y - points[0].x \* points[3].y); |
|  | } |
|  |  |
|  | template<class T> |
|  | point<T> trapezoid<T>::center() const { |
|  | point<T> res; |
|  | res.x = (points[0].x + points[1].x + points[2].x + points[3].x) / 4; |
|  | res.y = (points[0].y + points[1].y + points[2].y + points[3].y) / 4; |
|  | return res; |
|  |  |
|  | } |
|  |  |
|  | template<class T> |
|  | void trapezoid<T>::print(std::ostream& os) const { |
|  | for(int i = 0; i < 4; ++i){ |
|  | os << points[i]; |
|  | if(i + 1 != 4){ |
|  | os << ' '; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | #endif |

templates.h

|  |
| --- |
| #ifndef TEMPLATES\_H\_ |
|  | #define TEMPLATES\_H\_ |
|  |  |
|  | #include <tuple> |
|  | #include <type\_traits> |
|  | #include "point.h" |
|  |  |
|  |  |
|  | template<class T> |
|  | struct is\_point : std::false\_type {}; |
|  |  |
|  | template<class T> |
|  | struct is\_point<point<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\_point<Head>, std::is\_same<Head, Tail>...> {}; |
|  |  |
|  | template<class T> |
|  | inline constexpr bool is\_figurelike\_tuple\_v = is\_figurelike\_tuple<T>::value; |
|  |  |
|  |  |
|  |  |
|  | template<class T, class = void> |
|  | struct has\_method\_area : std::false\_type {}; |
|  |  |
|  | template<class T> |
|  | struct has\_method\_area<T, std::void\_t<decltype(std::declval<const T&>().area())>> : std::true\_type {}; |
|  |  |
|  | template<class T> |
|  | inline constexpr bool has\_method\_area\_v = has\_method\_area<T>::value; |
|  |  |
|  | template<class T> |
|  | std::enable\_if\_t<has\_method\_area\_v<T>, double> area(const T& object) { |
|  | return object.area(); |
|  | } |
|  |  |
|  | template<class T, class = void> |
|  | struct has\_method\_center : std::false\_type {}; |
|  |  |
|  | template<class T> |
|  | struct has\_method\_center<T, std::void\_t<decltype(std::declval<const T&>().center())>> : std::true\_type {}; |
|  |  |
|  | template<class T> |
|  | inline constexpr bool has\_method\_center\_v = has\_method\_center<T>::value; |
|  |  |
|  | template<class T> |
|  | std::enable\_if\_t<has\_method\_center\_v<T>, point<double>> center(const T& object) { |
|  | return object.center(); |
|  | } |
|  |  |
|  | template<class T, class = void> |
|  | struct has\_method\_print : std::false\_type {}; |
|  |  |
|  | template<class T> |
|  | struct has\_method\_print<T, std::void\_t<decltype(std::declval<const T&>().print(std::cout))>> : std::true\_type {}; |
|  |  |
|  | template<class T> |
|  | inline constexpr bool has\_method\_print\_v = has\_method\_print<T>::value; |
|  |  |
|  | template<class T> |
|  | std::enable\_if\_t<has\_method\_print\_v<T>, void> print(std::ostream& os, const T& object) { |
|  | object.print(os); |
|  | } |
|  |  |
|  | template<size\_t Id, class T> |
|  | double compute\_area(const T& tuple) { |
|  | if constexpr (Id >= std::tuple\_size\_v<T>){ |
|  | return 0; |
|  | }else{ |
|  | const auto x1 = std::get<Id - 0>(tuple).x - std::get<0>(tuple).x; |
|  | const auto y1 = std::get<Id - 0>(tuple).y - std::get<0>(tuple).y; |
|  | const auto x2 = std::get<Id - 1>(tuple).x - std::get<0>(tuple).x; |
|  | const auto y2 = std::get<Id - 1>(tuple).y - std::get<0>(tuple).y; |
|  | const double local\_area = std::abs(x1 \* y2 - y1 \* x2) \* 0.5; |
|  | return local\_area + compute\_area<Id + 1>(tuple); |
|  | } |
|  | } |
|  |  |
|  | template<class T> |
|  | std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, double> |
|  | area(const T& object) { |
|  | if constexpr (std::tuple\_size\_v<T> < 3){ |
|  | throw std::logic\_error("It`s not a figure"); |
|  | }else{ |
|  | return compute\_area<2>(object); |
|  | } |
|  | } |
|  |  |
|  | template<size\_t Id, class T> |
|  | point<double> tuple\_center(const T& object) { |
|  | if constexpr (Id >= std::tuple\_size<T>::value) { |
|  | return point<double> {0, 0}; |
|  | } else { |
|  | point<double> res = std::get<Id>(object); |
|  | return res + tuple\_center<Id+1>(object); |
|  | } |
|  | } |
|  |  |
|  | template<class T> |
|  | point<double> compute\_center(const T &tuple) { |
|  | point<double> res{0, 0}; |
|  | res = tuple\_center<0>(tuple); |
|  | res /= std::tuple\_size\_v<T>; |
|  | return res; |
|  | } |
|  |  |
|  | template<class T> |
|  | std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, point<double>> |
|  | center(const T& object) { |
|  | if constexpr (std::tuple\_size\_v<T> < 3){ |
|  | throw std::logic\_error("It`s not a figure"); |
|  | }else{ |
|  | return compute\_center(object); |
|  | } |
|  | } |
|  |  |
|  | template<size\_t Id, class T> |
|  | void step\_print(const T& object, std::ostream& os) { |
|  | if constexpr (Id >= std::tuple\_size<T>::value) { |
|  | std::cout << "\n"; |
|  | } else { |
|  | os << std::get<Id>(object) << " "; |
|  | step\_print<Id + 1>(object, os); |
|  | } |
|  | } |
|  |  |
|  | template<class T> |
|  | std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, void> |
|  | print(std::ostream& os, const T& object) { |
|  | if constexpr (std::tuple\_size\_v<T> < 3){ |
|  | throw std::logic\_error("It`s not a figure"); |
|  | }else{ |
|  | step\_print<0>(object, os); |
|  | } |
|  | }  template<class U>  struct HasFL<U> :  std::conjunction<  std::is\_same<decltype(std::declval<const U>().first), std::string>,  std::is\_same<decltype(std::declval<const U>().last), std::string>  > {}; |
|  |  |
|  | #endif |

CMakeLists.txt

|  |
| --- |
| cmake\_minimum\_required (VERSION 3.5) |
|  |  |
|  | project(lab4) |
|  |  |
|  | add\_executable(oop\_exercise\_04 |
|  | main.cpp |
|  | ) |
|  |  |
|  | set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -Wall -Wextra –std:c++17 -g3") |
|  |  |
|  | set\_target\_properties(oop\_exercise\_04 PROPERTIES CXX\_STANDART 17 CXX\_STANDART\_REQUIRED ON) |

1. **Набор testcases**

test\_01.txt

|  |
| --- |
| 1 |
|  | 0 0 1 1 2 1 3 0 |
|  | 2 |
|  | 0 0 -1 2 0 4 1 2 |
|  | 3 |
|  | 0 0 0 1 4 1 4 0 |
|  | q |

test\_02.txt

|  |
| --- |
| 4 |
|  | 3 |
|  | 0 0 1 1 2 0 |
|  | 4 |
|  | 4 |
|  | 0 0 0 2 2 2 2 0 |

1. **Результаты выполнения тестов**

**Input figure type: 1 - trapezoid, 2 - rhombus, 3 - rectangle, 4 - tuple, 5 – additional task or 'q' t**

**o quit**

**1**

**0 0 1 1 2 1 3 0**

**[0 0] [1 1] [2 1] [3 0]**

**2**

**[1.5 0.5]**

**2**

**0 0 -1 2 0 4 1 2**

**[0 0] [-1 2] [0 4] [1 2]**

**4**

**[0 2]**

**3**

**0 0 0 1 4 1 4 0**

**[0 0] [0 1] [4 1] [4 0]**

**4**

**[2 0.5]**

**q**

**C:\lab4\lab4\Debug\lab4.exe (процесс 17196) завершил работу с кодом 0.**

**Input figure type: 1 - trapezoid, 2 - rhombus, 3 - rectangle, 4 - tuple, 5 – additional task or 'q' t**

**o quit**

**4**

**Input number of angles**

**3**

**0 0 1 1 2 0**

**[0 0] [1 1] [2 0]**

**1**

**[1 0.333333]**

**4**

**Input number of angles**

**4**

**0 0 0 2 2 2 2 0**

**[0 0] [0 2] [2 2] [2 0]**

**4**

**[1 1]**

**q**

**C:\lab4\lab4\Debug\lab4.exe (процесс 13468) завершил работу с кодом 0.**

**Input figure type: 1 - trapezoid, 2 - rhombus, 3 - rectangle, 4 – tuple, 5 – additional task or 'q' t**

**o quit**

**5**

**1**

**0**

**C:\lab4\lab4\Debug\lab4.exe (процесс 18464) завершил работу с кодом 0.**

1. **Объяснение результатов работы программы - вывод**

В файлах rectangle.h, trapezoid.h и rhombus.h описаны фигуры. В templates.h описаны шаблоны для работы с этими фигурами и tuple. Также мне было дано дополнительное задание – являются ли поля first и last класса одновременно типа std::string. Для этого необходимо было применить проверку сравнения типов std::is\_same.

В ходе выполнения данной лабораторной работы были получены навыки работы с шаблонами, а также хэдером <type\_traits>, создания шаблонных классов.