**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 4**

Тема: Основы метапрограммирования

Студент: Блистунова Валерия

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

Вариант №12: Трапеция, Ромб, 5-угольник

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.

Для хранения координат фигур необходимо использовать шаблон std::pair.

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;

Создать программу, которая позволяет:

Вводить из стандартного ввода std::cin фигуры, согласно варианту задания. Вызывать для неё шаблонные функции функции (1-3 см. выше)

1. Ссылка на github

<https://github.com/val-lera/oop_exercise_04>

1. Описание программы

Программа предназначена для работы с геометрическими фигурами. Печатается меню, где пользователь может выбрать нужное действие. При необходимости есть возможность распечатать меню снова. В меню необходимо выбрать нужную фигуру: трапецию, ромб или пятиугольник. После выбора фигуры пользователь выбирает необходимый тип данных: int, double. Далее необходимо ввести координаты вершин. Для трапеции необходимо задать три вершины, для ромба четыре, для пятиугольника координаты центра и радиус описанной окружности. Программа выведет на экран площадь, координаты вершин, координаты центра для введенной фигуры. Чтобы завершить работу программы необходимо выбрать пункт меню- 0.

1. Набор тестов

Тест 1.

|  |  |  |
| --- | --- | --- |
| Пункт меню | Действие | Предполагаемый результат |
| 2 | Создать трапецию | Откроется дополнительное меню, в котором необходимо выбрать нужный тип данных |
| 1 | Выбрать тип int  Ввести координаты трапеции  0 0 2 6 4 6 | Выберется тип int, который будет использоваться в дальнейших вычислениях  На экран будут выведены координаты вершин трапеции, её площадь и координаты центра |
| 3 | Создать ромб | Откроется дополнительное меню, в котором необходимо выбрать нужный тип данных |
| 1 | Выбрать тип int  Ввести координаты ромба  -2 0 0 4 -4 2 -6 -2 | Выберется тип int, который будет использоваться в дальнейших вычислениях  На экран будут выведены координаты вершин ромба, его площадь и координаты центра |
| 1 | Распечатать меню | На экран выведется меню |
| 4 | Создать пятиугольник | Откроется дополнительное меню, в котором необходимо выбрать нужный тип данных |
| 2 | Выбрать тип double  Ввести радиус описанной окружности и координаты центра пятиугольника  2 5 5 | Выберется тип double, который будет использоваться в дальнейших вычислениях  На экран будут выведены координаты вершин пятиугольника, его площадь и координаты центра |
| 2 | Создать трапецию | Откроется дополнительное меню, в котором необходимо выбрать нужный тип данных |
| 2 | Выбрать тип double  Ввести координаты трапеции  1 2 3 5 6 5 | Выберется тип double, который будет использоваться в дальнейших вычислениях  На экран будут выведены координаты вершин трапеции, её площадь и координаты центра |
| 0 | Выйти из программы | Завершение программы |

1. Результаты выполнения тестов

Фактические результаты выполнения тестов совпали с предполагаемыми.

1. Листинг программы

/\*Лабораторная работа №4

Выполнила: Блистунова Валерия

Группа М8О-201Б

Вариант №12: Трапеция, Ромб, 5-угольник

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.

Для хранения координат фигур необходимо использовать шаблон std::pair.

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;

Создать программу, которая позволяет:

Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.

Вызывать для неё шаблонные функции функции (1-3 см. выше).\*/

#include<iostream>

#include<cmath>

#include <type\_traits>

#include <string>

#include <tuple>

#define pi 3.14159265358979323846

using namespace std;

template <class T>

double length(T a, T b, T c, T d){

double l;

using type=T;

l= sqrt(pow((c-a),2)+pow((d-b),2));

return l;

};

template <class T>

class Trap{

public:

using type=T;

using vertex=std::pair<T,T>;

vertex a,b,c,d;

Trap(){}

Trap(vertex &v1, vertex &v2, vertex &v3){

a=v1;

b=v2;

c=v3;

vertex v4;

v4.first= length(b.first, b.second, c.first, c.second)+2\*length(a.first, a.second, b.first, a.second)+a.first;

v4.second=a.second;

d=v4;

}

};

template <class T>

struct is\_Trap{

static const bool value = false;

};

template <class T>

struct is\_Trap<Trap<T>>{

static const bool value = true;

};

template <class T>

using Trap\_coord = typename std::enable\_if<is\_Trap<T>::value, void >::type;

template <class T>

using Trap\_area = typename std::enable\_if<is\_Trap<T>::value, typename T::type >::type;

template <class T>

using Trap\_center = typename std::enable\_if<is\_Trap<T>::value, std::pair<typename T::type, typename T::type> >::type;

template <class T>

Trap\_coord<T> print\_coord(const T &t){

cout << "Координаты вершин трапеции: ";

cout << "(" << t.a.first << "; " << t.a.second << ") ";

cout << "(" << t.b.first << "; " << t.b.second << ") ";

cout << "(" << t.c.first << "; " << t.c.second << ") ";

cout << "(" << t.d.first << "; " << t.d.second << ") " << endl;;

}

template <class T>

std::pair<T, T> print\_coord(std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &f){

std::pair<T,T> v1, v2, v3;

v1 = std::get<0>(f);

v2 = std::get<1>(f);

v3 = std::get<2>(f);

Trap<T> t(v1, v2, v3);

return print\_coord(t);

}

template <class T>

Trap\_area<T> calc\_area(const T &t){

typename T::type x1, x2, x3, x4, y1, y2, y3, y4;

x1=t.a.first;

x2=t.b.first;

x3=t.c.first;

x4=t.d.first;

y1=t.a.second;

y2=t.b.second;

y3=t.c.second;

y4=t.d.second;

auto h=length(x2,y1,x2,y2);

auto ar=((length(x1,x1,x4,y4)+length(x3,y3,x2,y2))/2)\*h;

return ar;

}

template <class T>

std::pair<T, T> calc\_area(std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &f){

std::pair<T,T> v1, v2, v3;

v2 = std::get<1>(f);

v3 = std::get<2>(f);

Trap<T> t(v1, v2, v3);

return calc\_area(t);

}

template <class T>

Trap\_center<T> calc\_center(const T &t){

typename T::type x1, x2, x3, x4, y1, y2, y3, y4;

double xc,yc;

x1=t.a.first;

x2=t.b.first;

x3=t.c.first;

x4=t.d.first;

y1=t.a.second;

y2=t.b.second;

y3=t.c.second;

y4=t.d.second;

xc=(x1+x2+x3+x4)/4;

yc=(y1+y2+y3+y4)/4;

cout << "Координаты центра трапеции: " << "(" << xc << "; " << yc << ")" << endl;

}

template <class T>

std::pair<T, T> calc\_center( std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &f){

std::pair<T,T> v1, v2, v3;

v1 = std::get<0>(f);

v2 = std::get<1>(f);

v3 = std::get<2>(f);

Trap<T> t(v1, v2, v3);

return calc\_center(t);

}

template <class T>

class Romb{

public:

using type=T;

using vertex=std::pair<T,T>;

vertex a,b,c,d;

Romb(){}

Romb(vertex &v1, vertex &v2, vertex &v3, vertex &v4){

a=v1;

b=v2;

c=v3;

d=v4;

}

};

template <class T>

struct is\_Romb{

static const bool value = false;

};

template <class T>

struct is\_Romb<Romb<T>>{

static const bool value = true;

};

template <class T>

using Romb\_coord = typename std::enable\_if<is\_Romb<T>::value, void >::type;

template <class T>

using Romb\_area = typename std::enable\_if<is\_Romb<T>::value, typename T::type >::type;

template <class T>

using Romb\_center = typename std::enable\_if<is\_Romb<T>::value, std::pair<typename T::type, typename T::type> >::type;

template <class T>

Romb\_area<T> calc\_area(const T &r){

typename T::type x1, x2, x3, x4, y1, y2, y3, y4;

x1=r.a.first;

x2=r.b.first;

x3=r.c.first;

x4=r.d.first;

y1=r.a.second;

y2=r.b.second;

y3=r.c.second;

y4=r.d.second;

auto ar=(length(x1, y1, x3, y3)\*length(x2, y2, x4, y4))/2;

return ar;

}

template <class T>

std::pair<T, T> calc\_area(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T> > &f){

std::pair<T,T> v1, v2, v3, v4;

v1 = std::get<0>(f);

v2 = std::get<1>(f);

v3 = std::get<2>(f);

v4 = std::get<3>(f);

Romb<T> r(v1, v2, v3, v4);

return calc\_area(r);

}

template <class T>

Romb\_center<T> calc\_center(const T &r){

typename T::type x1, x2, x3, x4, xc, y1, y2, y3, y4, yc;

x1=r.a.first;

x2=r.b.first;

x3=r.c.first;

x4=r.d.first;

y1=r.a.second;

y2=r.b.second;

y3=r.c.second;

y4=r.d.second;

xc=(x1+x2+x3+x4)/4;

yc=(y1+y2+y3+y4)/4;

cout << "Координаты центра ромба: " << "(" << xc << "; " << yc << ")" << endl;

}

template <class T>

std::pair<T, T> calc\_center(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T> > &f){

std::pair<T,T> v1, v2, v3, v4;

v1 = std::get<0>(f);

v2 = std::get<1>(f);

v3 = std::get<2>(f);

v4 = std::get<3>(f);

Romb<T> r(v1, v2, v3, v4);

return calc\_center(r);

}

template <class T>

Romb\_coord<T> print\_coord(const T &r){

cout << "Координаты вершин ромба: ";

cout << "(" << r.a.first << "; " << r.a.second << ") ";

cout << "(" << r.b.first << "; " << r.b.second << ") ";

cout << "(" << r.c.first << "; " << r.c.second << ") ";

cout << "(" << r.d.first << "; " << r.d.second << ")" << endl;

}

template <class T>

std::pair<T, T> print\_coord(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &f){

std::pair<T,T> v1, v2, v3, v4;

v1 = std::get<0>(f);

v2 = std::get<1>(f);

v3 = std::get<2>(f);

v4 = std::get<3>(f);

Romb<T> r(v1, v2, v3, v4);

return print\_coord(r);

}

template <class T>

class Pent{

public:

using type=T;

using vertex=std::pair<T,T>;

vertex cen, a, b, c, d, e;

T r;

Pent(){}

Pent(vertex &v1, T &r){

r;

cen=v1;

a.first=cen.first+r\*cos(0);

b.first=cen.first+r\*cos(2\*pi\*1/5);

c.first=cen.first+r\*cos(2\*pi\*2/5);

d.first=cen.first+r\*cos(2\*pi\*3/5);

e.first=cen.first+r\*cos(2\*pi\*4/5);

a.second=cen.second+r\*sin(0);

b.second=cen.second+r\*sin(2\*pi\*1/5);

c.second=cen.second+r\*sin(2\*pi\*2/5);

d.second=cen.second+r\*sin(2\*pi\*3/5);

e.second=cen.second+r\*sin(2\*pi\*4/5);

}

};

template <class T>

struct is\_Pent{

static const bool value = false;

};

template <class T>

struct is\_Pent<Pent<T>>{

static const bool value = true;

};

template <class T>

using Pent\_coord = typename std::enable\_if<is\_Pent<T>::value, void >::type;

template <class T>

using Pent\_area = typename std::enable\_if<is\_Pent<T>::value, typename T::type >::type;

template <class T>

using Pent\_center = typename std::enable\_if<is\_Pent<T>::value, std::pair<typename T::type, typename T::type> >::type;

template <class T>

Pent\_area<T> calc\_area(const T &p){

typename T::type x1, x2, y1, y2;

x1=p.a.first;

x2=p.b.first;

auto l=length(x1,y1,x2,y2);

auto ar=(5/2)\*l\*sqrt((pow((l/(2\*sin(pi/5))),2))-(pow(l,2)/4));

return ar;

}

template <class T>

Pent\_center<T> calc\_center(const T &p){

cout << "Координаты центра пятиугольника: " << "(" << p.cen.first << "; " << p.cen.second << ")" << endl;

}

template <class T>

Pent\_coord<T> print\_coord(const T &p){

cout << "Координаты вершин пятиугольника: ";

cout << "(" << p.a.first << "; " << p.a.second << ") ";

cout << "(" << p.b.first << "; " << p.b.second << ") ";

cout << "(" << p.c.first << "; " << p.c.second << ") ";

cout << "(" << p.d.first << "; " << p.d.second << ") ";

cout << "(" << p.e.first << "; " << p.e.second << ") " << endl;

}

void Menu(){

cout << "Введите число от 1 до 4 или 0 для действий: " << endl <<

"1. Распечатать меню" << endl <<

"2. Трапеция" << endl <<

"3. Ромб" << endl <<

"4. Пятиугольник" << endl <<

"0. Выход из программы" << endl;

};

void Type\_menu(){

cout << "Введите число, соответствующее необходимому типу: " << endl <<

"1. int" << endl <<

"2. double" << endl;

};

int main(){

cout << "Работа с геометрическими фигурами" << endl;

int user\_command=1;

int k, j;

using pair\_int = std::pair<int, int>;

using pair\_double = std::pair<double, double>;

std::tuple<pair\_int, pair\_int, pair\_int> tuple\_intTrap;

std::tuple<pair\_double, pair\_double, pair\_double> tuple\_doubleTrap;

std::tuple<pair\_int, pair\_int, pair\_int, pair\_int> tuple\_intRomb;

std::tuple<pair\_double, pair\_double, pair\_double, pair\_double> tuple\_doubleRomb;

for(;;){

switch (user\_command){

case 1:

Menu();

break;

case 2: //trap

{Type\_menu();

cin >>k;

switch (k){

case 1: //int

{cout << "Введите координаты трёх вершин трапеции" << endl;

std::pair<int,int> v1, v2, v3;

int a, b, c, d, e, f;

cin >> a >> b >> c >> d >> e >> f;

v1.first=a;

v1.second=b;

v2.first=c;

v2.second=d;

v3.first=e;

v3.second=f;

Trap<int> t\_int(v1, v2, v3);

print\_coord(t\_int);

cout << "Площадь трапеции: " << calc\_area(t\_int) << endl;

calc\_center(t\_int);

std::make\_tuple(v1, v2, v3);}

break;

case 2: //double

{cout << "Введите координаты трёх вершин трапеции" << endl;

std::pair<double,double> v1, v2, v3;

double a, b, c, d, e, f;

cin >> a >> b >> c >> d >> e >> f;

v1.first=a;

v1.second=b;

v2.first=c;

v2.second=d;

v3.first=e;

v3.second=f;

Trap<double> t\_double(v1, v2, v3);

print\_coord(t\_double);

cout << "Площадь трапеции: " << calc\_area(t\_double) << endl;

calc\_center(t\_double);

std::make\_tuple(v1, v2, v3);}

break;

default:

break;}

}

break;

case 3: //romb

{Type\_menu();

cin >> k;

switch (k){

case 1: //int

{cout << "Введите координаты вершин ромба" << endl;

std::pair<int,int> v1, v2, v3, v4;

int a, b, c, d, e, f, g, h;

cin >> a >> b >> c >> d >> e >> f >> g >> h;

v1.first=a;

v1.second=b;

v2.first=c;

v2.second=d;

v3.first=e;

v3.second=f;

v4.first=g;

v4.second=h;

Romb<int> r\_int(v1, v2, v3, v4);

print\_coord(r\_int);

cout << "Площадь ромба: " << calc\_area(r\_int) << endl;

calc\_center(r\_int);

std::make\_tuple(v1, v2, v3, v4);}

break;

case 2: //double

{cout << "Введите координаты вершин ромба" << endl;

std::pair<double,double> v1, v2, v3, v4;

double a, b, c, d, e, f, g, h;

cin >> a >> b >> c >> d >> e >> f >> g >> h;

v1.first=a;

v1.second=b;

v2.first=c;

v2.second=d;

v3.first=e;

v3.second=f;

v4.first=g;

v4.second=h;

Romb<double> r\_double(v1, v2, v3, v4);

print\_coord(r\_double);

cout << "Площадь ромба: " << calc\_area(r\_double) << endl;

calc\_center(r\_double);

std::make\_tuple(v1, v2, v3, v4);}

break;

default:

break;}

}

break;

case 4: //pent

{Type\_menu();

cin >> k;

switch (k){

case 1: //int

{{cout << "Введите координаты центра пятиугольника и радиус описанной окружности " << endl;

std::pair<int,int> v1;

int a, b, r;

cin >> a >> b >> r;

v1.first=a;

v1.second=b;

Pent<int> p\_int(v1, r);

print\_coord(p\_int);

cout << "Площадь пятиугольника: " << calc\_area(p\_int) << endl;

calc\_center(p\_int);}

break;

case 2: //double

{cout << "Введите координаты центра пятиугольника и радиус описанной окружности " << endl;

std::pair<double,double> v1;

double a, b, r;

cin >> a >> b >> r;

v1.first=a;

v1.second=b;

Pent<double> p\_double(v1, r);

print\_coord(p\_double);

cout << "Площадь пятиугольника: " << calc\_area(p\_double) << endl;

calc\_center(p\_double);}

break;

default:

break;}

}

break;

case 0:

exit(0);

default:

cerr << "Нет такого пункта" << endl;

break;

}

cout << endl << "Введите пункт меню: ";

cin >> user\_command;

}

return 0;

}

1. Вывод

Создана программа на языке С++, позволяющая работать с геометрическими фигурами: находить координаты центра фигур, считать их площадь, выводить на экран координаты вершин.