Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа**

**по курсу «ООП»**

**Тема:**

**Основы метапрограммирования.**

|  |  |
| --- | --- |
| Студент: | Симонов С.Я. |
| Группа: | М8О-206Б-18 |
| Преподаватель: | Журавлев А.А. |
| Вариант: | 21 |
| Оценка: |  |
| Дата: |  |

Москва

2019

**1. Код на C++:**

**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;

}

template <class T>

vertex<T> operator+ (vertex<T> p1, vertex<T> p2) {

vertex<T> p;

p.x = p1.x + p2.x;

p.y = p1.y + p2.y;

return p;

}

template <class T>

vertex<T>& operator/ (vertex<T>& p, int num) {

p.x = p.x / num;

p.y = p.y / num;

return p;

}

#endif // D\_VERTEX\_H\_

**templates.h:**

#ifndef D\_TEMPLATES\_H\_

#define D\_TEMPLATES\_H\_ 1

#include <tuple>

#include <type\_traits>

#include <cmath>

#include "rhombus.h"

#include "pentagon.h"

#include "hexagon.h"

#include "vertex.h"

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;

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::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) {

figure.print();

}

template<size\_t ID, class T>

void single\_print(const T& t) {

std::cout << std::get<ID>(t);

return ;

}

template<size\_t ID, class T>

void Recursiveprint(const T& t) {

if constexpr (ID < std::tuple\_size\_v<T>){

single\_print<ID>(t);

Recursiveprint<ID+1>(t);

return ;

}

return;

}

template<class T>

std::enable\_if\_t <is\_figurelike\_tuple\_v<T>, void>

print(const T& fake) {

return Recursiveprint<0>(fake);

}

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> single\_center(const T& t) {

vertex<double> v;

v.x = std::get<ID>(t).x;

v.y = std::get<ID>(t).y;

v = v / std::tuple\_size\_v<T>;

return v;

}

template<size\_t ID, class T>

vertex<double> Recursivecenter(const T& t) {

if constexpr (ID < std::tuple\_size\_v<T>){

return single\_center<ID>(t) + Recursivecenter<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 Recursivecenter<0>(fake);

}

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 single\_area(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 Recursivearea(const T& t) {

if constexpr (ID < std::tuple\_size\_v<T>){

return single\_area<ID>(t) + Recursivearea<ID + 1>(t);

}

return 0;

}

template<class T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, double>

area(const T& fake) {

return Recursivearea<2>(fake);

}

template<class T, class = void>

struct has\_perimeter\_method : std::false\_type {};

template<class T>

struct has\_perimeter\_method<T,

std::void\_t<decltype(std::declval<const T>().perimeter())>> :

std::true\_type {};

template<class T>

inline constexpr bool has\_perimeter\_method\_v =

has\_perimeter\_method<T>::value;

template<class T>

std::enable\_if\_t<has\_perimeter\_method\_v<T>, double>

perimeter(const T& figure) {

return figure.perimeter();

}

template<size\_t ID, class T>

double single\_perimeter(const T& t) {

const auto& c = std::get<0>(t);

const auto& a = std::get<ID - 1>(t);

const auto& b = std::get<ID>(t);

const double dx1 = b.x - a.x;

const double dy1 = b.y - a.y;

const double dx2 = c.x - b.x;

const double dy2 = c.y - b.y;

if (ID == std::tuple\_size\_v<T> - 1) {

return std::sqrt((dx1 \* dx1) + (dy1 \* dy1)) + std::sqrt((dx2 \* dx2) + (dy2 \* dy2));

}

return std::sqrt((dx1 \* dx1) + (dy1 \* dy1));

}

template<size\_t ID, class T>

double Recursiveperimeter(const T& t) {

if constexpr (ID < std::tuple\_size\_v<T>) {

double s = single\_perimeter<ID>(t) + Recursiveperimeter<ID + 1>(t);

return s;

}

return 0;

}

template<class T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, double>

perimeter(const T& fake) {

return Recursiveperimeter<1>(fake);

}

#endif

**rhombus.h**

#ifndef D\_RHOMBUS\_H\_

#define D\_RHOMBUS\_H\_ 1

#include <algorithm>

#include <iostream>

#include <assert.h>

#include <cmath>

#include "vertex.h"

template<class T>

struct rhombus {

public:

rhombus (std::istream& is);

bool correct() const;

vertex<double> center() const;

double area() const;

double perimeter() const;

void print() const;

private:

vertex<T> a1, a2, a3, a4;

};

template <class T>

rhombus<T>::rhombus(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4;

assert(correct());

}

template <class T>

bool rhombus<T>::correct() const {

T str1, str2, str3, str4;

str1 = sqrt((a2.x - a1.x) \* (a2.x - a1.x) + (a2.y - a1.y) \* (a2.y - a1.y));

str2 = sqrt((a3.x - a2.x) \* (a3.x - a2.x) + (a3.y - a2.y) \* (a3.y - a2.y));

str3 = sqrt((a4.x - a3.x) \* (a4.x - a3.x) + (a4.y - a3.y) \* (a4.y - a3.y));

str4 = sqrt((a1.x - a4.x) \* (a1.x - a4.x) + (a1.y - a4.y) \* (a1.y - a4.y));

if (str1 == str2 && str2 == str3 && str3 == str4) {

return true;

}

return false;

}

template <class T>

vertex<double> rhombus<T>::center() const {

vertex<double> p;

p.x = (a1.x + a2.x + a3.x + a4.x) / 4;

p.y = (a1.y + a2.y + a3.y + a4.y) / 4;

return p;

}

template <class T>

double rhombus<T>::area() const {

const T s1 = 0.5 \* abs((a2.x - a1.x) \* (a3.y - a1.y) - (a3.x - a1.x) \* (a2.y - a1.y));

const T s2 = 0.5 \* abs((a3.x - a1.x) \* (a4.y - a1.y) - (a4.x - a1.x) \* (a3.y - a1.y));

return s1 + s2;

}

template <class T>

double rhombus<T>::perimeter() const {

const T str1 = sqrt((a2.x - a1.x) \* (a2.x - a1.x) + (a2.y - a1.y) \* (a2.y - a1.y));

const T str2 = sqrt((a3.x - a2.x) \* (a3.x - a2.x) + (a3.y - a2.y) \* (a3.y - a2.y));

const T str3 = sqrt((a4.x - a3.x) \* (a4.x - a3.x) + (a4.y - a3.y) \* (a4.y - a3.y));

const T str4 = sqrt((a1.x - a4.x) \* (a1.x - a4.x) + (a1.y - a4.y) \* (a1.y - a4.y));

return str1 + str2 + str3 + str4;

}

template <class T>

void rhombus<T>::print() const {

std::cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << '\n';

}

#endif

**pentagon.h**

#ifndef D\_PENTAGON\_H\_

#define D\_PENTAGON\_H\_ 1

#include <algorithm>

#include <iostream>

#include "vertex.h"

template<class T>

struct pentagon {

public:

pentagon(std::istream& is);

vertex<double> center() const;

double area() const;

double perimeter() const;

void print() const;

private:

vertex<T> a1, a2, a3, a4, a5;

};

template <class T>

pentagon<T>::pentagon(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5;

}

template <class T>

vertex<double> pentagon<T>::center() const {

vertex<double> p;

p.x = (a1.x + a2.x + a3.x + a4.x + a5.x) / 5;

p.y = (a1.y + a2.y + a3.y + a4.y + a5.y) / 5;

return p;

}

template <class T>

double pentagon<T>::area() const {

const T s1 = 0.5 \* abs((a2.x - a1.x) \* (a3.y - a1.y) - (a3.x - a1.x) \* (a2.y - a1.y));

const T s2 = 0.5 \* abs((a3.x - a1.x) \* (a4.y - a1.y) - (a4.x - a1.x) \* (a3.y - a1.y));

const T s3 = 0.5 \* abs((a4.x - a1.x) \* (a5.y - a1.y) - (a5.x - a1.x) \* (a4.y - a1.y));

return s1 + s2 + s3;

}

template <class T>

double pentagon<T>::perimeter() const {

const T str1 = sqrt((a2.x - a1.x) \* (a2.x - a1.x) + (a2.y - a1.y) \* (a2.y - a1.y));

const T str2 = sqrt((a3.x - a2.x) \* (a3.x - a2.x) + (a3.y - a2.y) \* (a3.y - a2.y));

const T str3 = sqrt((a4.x - a3.x) \* (a4.x - a3.x) + (a4.y - a3.y) \* (a4.y - a3.y));

const T str4 = sqrt((a5.x - a4.x) \* (a5.x - a4.x) + (a5.y - a4.y) \* (a5.y - a4.y));

const T str5 = sqrt((a1.x - a5.x) \* (a1.x - a5.x) + (a1.y - a5.y) \* (a1.y - a5.y));

return str1 + str2 + str3 + str4 + str5;

}

template <class T>

void pentagon<T>::print() const {

std::cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << ' ' << a5 << '\n';

}

#endif

**hexagon.h**

#ifndef D\_HEXAGON\_H\_

#define D\_HEXAGON\_H\_ 1

#include <algorithm>

#include <iostream>

#include "vertex.h"

template<class T>

struct hexagon {

public:

hexagon (std::istream& is);

vertex<double> center() const;

double area() const;

double perimeter() const;

void print() const;

private:

vertex<T> a1, a2, a3, a4, a5, a6;

};

template <class T>

hexagon<T>::hexagon(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;

}

template <class T>

vertex<double> hexagon<T>::center() const {

vertex<T> p;

p.x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x) / 6;

p.y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y) / 6;

return p;

}

template <class T>

double hexagon<T>::area() const {

const T s1 = 0.5 \* abs((a2.x - a1.x) \* (a3.y - a1.y) - (a3.x - a1.x) \* (a2.y - a1.y));

const T s2 = 0.5 \* abs((a3.x - a1.x) \* (a4.y - a1.y) - (a4.x - a1.x) \* (a3.y - a1.y));

const T s3 = 0.5 \* abs((a4.x - a1.x) \* (a5.y - a1.y) - (a5.x - a1.x) \* (a4.y - a1.y));

const T s4 = 0.5 \* abs((a5.x - a1.x) \* (a6.y - a1.y) - (a6.x - a1.x) \* (a5.y - a1.y));

return s1 + s2 + s3 + s4;

}

template <class T>

double hexagon<T>::perimeter() const {

const T str1 = sqrt((a2.x - a1.x) \* (a2.x - a1.x) + (a2.y - a1.y) \* (a2.y - a1.y));

const T str2 = sqrt((a3.x - a2.x) \* (a3.x - a2.x) + (a3.y - a2.y) \* (a3.y - a2.y));

const T str3 = sqrt((a4.x - a3.x) \* (a4.x - a3.x) + (a4.y - a3.y) \* (a4.y - a3.y));

const T str4 = sqrt((a5.x - a4.x) \* (a5.x - a4.x) + (a5.y - a4.y) \* (a5.y - a4.y));

const T str5 = sqrt((a6.x - a5.x) \* (a6.x - a5.x) + (a6.y - a5.y) \* (a6.y - a5.y));

const T str6 = sqrt((a1.x - a6.x) \* (a1.x - a6.x) + (a1.y - a6.y) \* (a1.y - a6.y));

return str1 + str2 + str3 + str4 + str5 + str6;

}

template <class T>

void hexagon<T>::print() const {

std::cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << ' ' << a5 << ' ' << a6 << '\n';

}

#endif

**main.cpp**

#include "rhombus.h"

#include "pentagon.h"

#include "hexagon.h"

#include "templates.h"

int main() {

int input;

while (true) {

std::cout << "0: Exit" << std::endl;

std::cout << "1: Fake figure demonstration" << std::endl;

std::cout << "2: Array figure demonstration" << std::endl;

std::cout << "3: Real figure demonstration" << std::endl;

std::cin >> input;

if (input == 0) {

break;

}

if (input > 3) {

std::cout << "Invalid input" << std::endl;

}

switch (input) {

case 1: {

std::cout << "Fake rhombus (float):" << std::endl;

std::tuple<vertex<float>, vertex<float>, vertex<float>, vertex<float>>

fakerhombus{{0, 0}, {0, 1}, {1, 1}, {1, 0}};

std::cout << "Coordinates: ";

print(fakerhombus);

std::cout << std::endl;

std::cout << "center: " << center(fakerhombus) << std::endl;

std::cout << "area: " << area(fakerhombus) << std::endl;

std::cout << "perimeter:" << perimeter(fakerhombus) << std::endl << std::endl;

std::cout << "Fake pentagon (int):" << std::endl;

std::tuple<vertex<int>, vertex<int>, vertex<int>, vertex<int>, vertex<int>>

fakepentagon{{0, 2}, {2, 4}, {4, 4}, {4, 2}, {2, 0}};

std::cout << "Coordinates: ";

print(fakepentagon);

std::cout << std::endl;

std::cout << "center: " << center(fakepentagon) << std::endl;

std::cout << "area: " << area(fakepentagon) << std::endl;

std::cout << "perimeter:" << perimeter(fakepentagon) << std::endl << std::endl;

std::cout << "Fake hexagon (double):" << std::endl;

std::tuple<vertex<double>, vertex<double>, vertex<double>, vertex<double>, vertex<double>, vertex<double>>

fakehexagon{{0, 5}, {1, 5}, {2, 5}, {2, 0}, {1, 0}, {0, 0}};

std::cout << "Coordinates: ";

print(fakehexagon);

std::cout << std::endl;

std::cout << "center: " << center(fakehexagon) << std::endl;

std::cout << "area: " << area(fakehexagon) << std::endl;

std::cout << "perimeter:" << perimeter(fakehexagon) << std::endl << std::endl;

break;

}

case 2: {

std::cout << "Array rhombus (double):" << std::endl;

std::array<vertex<double>, 4>

array\_rhombus{{{0, 0}, {0, 1}, {1, 1}, {1, 0}}};

std::cout << "Coordinates: ";

print(array\_rhombus);

std::cout << std::endl;

std::cout << "center: " << center(array\_rhombus) << std::endl;

std::cout << "area: " << area(array\_rhombus) << std::endl;

std::cout << "perimeter: " << perimeter(array\_rhombus) << std::endl << std::endl;

std::cout << "Array hexagon (float):" << std::endl;

std::array<vertex<float>, 6>

array\_hexagon{{{-1, 1}, {1, 2}, {3, 2}, {3, -1}, {1, -2}, {-1, -1}}};

std::cout << "Coordinates: ";

print(array\_hexagon);

std::cout << std::endl;

std::cout << "center: " << center(array\_hexagon) << std::endl;

std::cout << "area: " << area(array\_hexagon) << std::endl;

std::cout << "perimeter: " << perimeter(array\_hexagon) << std::endl << std::endl;

break;

}

case 3: {

int realID;

std::cout << "Input real figure id:" << std::endl;

std::cout << "1: rhombus" << std::endl;

std::cout << "2: pentagon" << std::endl;

std::cout << "3: hexagon" << std::endl;

std::cin >> realID;

switch (realID) {

case 1: {

std::cout << "Input 4 coordinates in a sequence" << std::endl;

rhombus<float> realrhombus(std::cin);

std::cout << "Coordinates: ";

print(realrhombus);

std::cout << std::endl;

std::cout << "center: " << center(realrhombus) << std::endl;

std::cout << "area: " << area(realrhombus) << std::endl;

std::cout << "perimeter: " << perimeter(realrhombus) << std::endl << std::endl;

break;

}

case 2: {

std::cout << "Input 5 coordinates in a sequence" << std::endl;

pentagon<double> realpentagon(std::cin);

std::cout << "Coordinates: ";

print(realpentagon);

std::cout << std::endl;

std::cout << "center: " << center(realpentagon) << std::endl;

std::cout << "area: " << area(realpentagon) << std::endl;

std::cout << "perimeter: " << perimeter(realpentagon) << std::endl << std::endl;

break;

}

case 3: {

std::cout << "Input 6 coordinates in a sequence" << std::endl;

hexagon<double> realhexagon(std::cin);

std::cout << "Coordinates: ";

print(realhexagon);

std::cout << std::endl;

std::cout << "center: " << center(realhexagon) << std::endl;

std::cout << "area: " << area(realhexagon) << std::endl;

std::cout << "area: " << perimeter(realhexagon) << std::endl << std::endl;

break;

}

}

break;

}

}

}

return 0;

}

**2. Ссылка на репозиторий в GitHub:**

https://github.com/keoni02032/oop\_exercise\_04

**3. Набор testcases:**

**test\_01.test:**

1

2

3

1

0 0

1 0

1 1

0 1

3

2

0 0

10 0

10 20

0 20

3

3

0 0

1 2

2 2

3 0

0

**test\_02.test:**

3

1

1234134 131

1312 321

321 2343

13 321

**test\_03.test:**

3

1

0 100

100 100

100 0

0 0

3

2

0 0

0.4 0

0.4 1000

0 1000

3

3

0 0

50 50

150 50

100 0

0

**4.Результаты выполнения программы:**

**test\_01.result**

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Fake rhombus (float):

Coordinates: [ 0 0 ][ 0 0.5 ][ 0.5 0.5 ][ 0.5 0 ]

center: [ 0.25 0.25 ]

area: 0.25

Fake pentagon (int):

Coordinates: [ 0 2 ][ 2 4 ][ 4 4 ][ 4 2 ][ 2 0 ]

center: [ 2.4 2.4 ]

area: 10

Fake hexagon (double):

Coordinates: [ 0 1 ][ 1 2 ][ 3 0 ][ 2 -2 ][ -1 -1 ][ -2 1 ]

center: [ 0.5 0.166667 ]

area: 11

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Array rhombus (double):

Coordinates: [ 1 1 ][ 2 3 ][ 3 1 ][ 2 -1 ]

center: [ 2 1 ]

area: 4

Array hexagon (float):

Coordinates: [ -1 1 ][ 1 2 ][ 3 2 ][ 3 -1 ][ 1 -2 ][ -1 -1 ]

center: [ 1 0.166667 ]

area: 13

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Input real figure id:

1: rhombus

2: pentagon

3: hexagon

Input 4 coordinates in a sequence

Coordinates: [ -1 0 ] [ 0 1 ] [ 1 0 ] [ 0 -1 ]

center: [ 0 0 ]

area: 2

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Input real figure id:

1: rhombus

2: pentagon

3: hexagon

Input 5 coordinates in a sequence

Coordinates: [ -1 0 ] [ 0 1 ] [ 1 1 ] [ 2 0 ] [ 1 -1 ]

center: [ 0.6 0.2 ]

area: 3.5

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

**test\_02.result**

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Fake rhombus (float):

Coordinates: [ 0 0 ][ 0 0.5 ][ 0.5 0.5 ][ 0.5 0 ]

center: [ 0.25 0.25 ]

area: 0.25

Fake pentagon (int):

Coordinates: [ 0 2 ][ 2 4 ][ 4 4 ][ 4 2 ][ 2 0 ]

center: [ 2.4 2.4 ]

area: 10

Fake hexagon (double):

Coordinates: [ 0 1 ][ 1 2 ][ 3 0 ][ 2 -2 ][ -1 -1 ][ -2 1 ]

center: [ 0.5 0.166667 ]

area: 11

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Array rhombus (double):

Coordinates: [ 1 1 ][ 2 3 ][ 3 1 ][ 2 -1 ]

center: [ 2 1 ]

area: 4

Array hexagon (float):

Coordinates: [ -1 1 ][ 1 2 ][ 3 2 ][ 3 -1 ][ 1 -2 ][ -1 -1 ]

center: [ 1 0.166667 ]

area: 13

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Input real figure id:

1: rhombus

2: pentagon

3: hexagon

Input 6 coordinates in a sequence

Coordinates: [ -1 0 ] [ 0 0.5 ] [ 1 1 ] [ 2 0 ] [ 1 -1 ] [ 0 -1 ]

center: [ 0.5 -0.0833333 ]

area: 3.5

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Fake rhombus (float):

Coordinates: [ 0 0 ][ 0 0.5 ][ 0.5 0.5 ][ 0.5 0 ]

center: [ 0.25 0.25 ]

area: 0.25

Fake pentagon (int):

Coordinates: [ 0 2 ][ 2 4 ][ 4 4 ][ 4 2 ][ 2 0 ]

center: [ 2.4 2.4 ]

area: 10

Fake hexagon (double):

Coordinates: [ 0 1 ][ 1 2 ][ 3 0 ][ 2 -2 ][ -1 -1 ][ -2 1 ]

center: [ 0.5 0.166667 ]

area: 11

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

**5. Объяснение результатов работы программы:**

Пользователю предоставляестся три варианта выбора работы прогрммы: указанное задание выполняется при помощи tuple (выполняется при поочередном задании точек), с использованием массива содержащем заранее заданные точки и заданием точек с клавиатуры. При вводе команды "0" происходит выход из программы. При ввооде команды "1" выводятся геометрический центр, координаты вершин фигуры, площадь фигур; фигуры выводятся в следедующем порядке: ромб типа float, пятиугольник типа int,  
шестиугольник типа double. При вводе команды "2" выводятся еометрический центр, координаты вершин фигуры, площадь фигур; фигуры выводятся в следедующем порядке: ромб с типом double, шестиугольник с типом float. При вооде команды "3" предоставляется выбор ввода одной из трех фигур: "1" ромб, "2" пятиугольник, "3" шестиугольник. После выбора фигуры необходимо ввести точки вершин, далее программа обрабатывает введеные результаты и выводит введеные координаты вершин, геометрическийй центр и площадь.

**6. Вывод:**

В данной лабораторной работе я освоил основы метапрограммирования, применил шаблоны класса для реализации классов фигур с переменным типом данных.