**1.**

//Пищик Е.С. М8О-206Б-19 Лабораторная работа №4 Вариант 16

//8-угольник, треугольник, квадрат

#include <iostream>

#include <string>

#include <utility>

#include <cmath>

#include <algorithm>

//Константа "pi"

#define PI 3.14159265

//Шаблонные классы фигур

//Класс квадрата

template <typename T, size\_t SIZE=4>

struct Square

{

    using type = T;

    std::pair<T,T> v\_arr[SIZE];

    std::pair<T,T>\* begin() { return &v\_arr[0]; }

    std::pair<T,T>\* end() { return &v\_arr[SIZE]; }

    constexpr double get\_square()

    {

        double side\_0 = sqrt(pow((v\_arr[1].first - v\_arr[0].first), 2) + pow((v\_arr[1].second - v\_arr[0].second), 2));

        double side\_1 = sqrt(pow((v\_arr[2].first - v\_arr[0].first), 2) + pow((v\_arr[2].second - v\_arr[0].second), 2));

        double side\_2 = sqrt(pow((v\_arr[3].first - v\_arr[0].first), 2) + pow((v\_arr[3].second - v\_arr[0].second), 2));

        double side = std::min({side\_0, side\_1, side\_2});

        return pow(side,2);

    }

};

//Класс треугольника

template <typename T, size\_t SIZE=3>

struct Triangle

{

    using type = T;

    std::pair<T,T> v\_arr[SIZE];

    std::pair<T,T>\* begin() { return &v\_arr[0]; }

    std::pair<T,T>\* end() { return &v\_arr[SIZE]; }

    constexpr double get\_square()

    {

        double side = sqrt(pow((v\_arr[1].first - v\_arr[0].first), 2) + pow((v\_arr[1].second - v\_arr[0].second), 2));

        return sqrt(3.0)/4.0\*pow(side,2);

    }

};

//Класс восьмиугольника

template <typename T, size\_t SIZE=8>

struct Octagon

{

    using type = T;

    std::pair<T,T> v\_arr[SIZE];

    std::pair<T,T>\* begin() { return &v\_arr[0]; }

    std::pair<T,T>\* end() { return &v\_arr[SIZE]; }

    constexpr double get\_square()

    {

        double side\_0 = sqrt(pow((v\_arr[1].first - v\_arr[0].first), 2) + pow((v\_arr[1].second - v\_arr[0].second), 2));

        double side\_1 = sqrt(pow((v\_arr[2].first - v\_arr[0].first), 2) + pow((v\_arr[2].second - v\_arr[0].second), 2));

        double side\_2 = sqrt(pow((v\_arr[3].first - v\_arr[0].first), 2) + pow((v\_arr[3].second - v\_arr[0].second), 2));

        double side\_3 = sqrt(pow((v\_arr[4].first - v\_arr[0].first), 2) + pow((v\_arr[4].second - v\_arr[0].second), 2));

        double side\_4 = sqrt(pow((v\_arr[5].first - v\_arr[0].first), 2) + pow((v\_arr[5].second - v\_arr[0].second), 2));

        double side\_5 = sqrt(pow((v\_arr[6].first - v\_arr[0].first), 2) + pow((v\_arr[6].second - v\_arr[0].second), 2));

        double side\_6 = sqrt(pow((v\_arr[7].first - v\_arr[0].first), 2) + pow((v\_arr[7].second - v\_arr[0].second), 2));

        double side = std::min({side\_0, side\_1, side\_2, side\_3, side\_4, side\_5, side\_6});

        return 2\*pow(side,2)/tan(PI/8);

    }

};

// Печать любой фигуры

template <typename T>

typename std::enable\_if<(sizeof(T::v\_arr)>0),void>::type print(T& fig)

{

    for(auto v: fig)

    {

        std::cout << "(" << v.first << "," << v.second << ") ";

    }

    std::cout << std::endl;

}

// Печать tuple фигур

template <typename T,size\_t index>

typename std::enable\_if<index>=std::tuple\_size<T>::value,void>::type print\_tuple(T&)

{

    std::cout << std::endl;

}

template <typename T,size\_t index>

typename std::enable\_if<index<std::tuple\_size<T>::value,void>::type print\_tuple(T& tuple)

{

    auto v = std::get<index>(tuple);

    std::cout << "Figure " << index << ": ";

    print(v);

    print\_tuple<T,index+1>(tuple);

}

//Нахождение суммарной площади в tuple

template <typename T,size\_t index>

double square\_tuple(T& tuple)

{

    auto item = std::get<index>(tuple);

    double value = item.get\_square();

    if constexpr ((index+1) < std::tuple\_size<T>::value)

    {

        return value + square\_tuple<T,index+1>(tuple);

    }

    return value;

}

int main()

{

    //Создаем три экземпляра шаблонных классов

    Square<int> sq;

    Triangle<double> tr;

    Octagon<double> ogon;

    //Инициализация

    sq.v\_arr[0] = {0,0};

    sq.v\_arr[1] = {1,1};

    sq.v\_arr[2] = {0,1};

    sq.v\_arr[3] = {1,0};

    tr.v\_arr[0] = {1.0,0.0};

    tr.v\_arr[1] = {0.0,0.0};

    tr.v\_arr[2] = {0.5,sqrt(0.75)};

    ogon.v\_arr[0] = {sqrt(2.0),sqrt(2.0)+2.0};

    ogon.v\_arr[1] = {sqrt(2.0)+1.0,1.0};

    ogon.v\_arr[2] = {-1.0,1.0};

    ogon.v\_arr[3] = {0.0,sqrt(2.0)+2.0};

    ogon.v\_arr[4] = {0.0,0.0};

    ogon.v\_arr[5] = {sqrt(2.0),0.0};

    ogon.v\_arr[6] = {-1.0,1.0+sqrt(2.0)};

    ogon.v\_arr[7] = {sqrt(2.0)+1,sqrt(2.0)+1.0};

    //Создаем tuple из фигур

    auto tuple = std::make\_tuple(tr, sq, ogon);

    //Печать tuple фигур

    print\_tuple<decltype(tuple),0>(tuple);

    //Печать суммарной площади фигур в tuple

    std::cout << "Square: " << square\_tuple<decltype(tuple),0>(tuple) << std::endl;

    return 0;

}

**2.** <https://github.com/Pe4enIks/OOP_Pishchik_206>

**3.**

Сначала инициализируем квадрат, потом треугольник, потом восьмиугольник, инициализация идет сразу в программе, т.к. объекты шаблонных классов используются в метафункциях.

sq – объект класса Square, tr – объект класса Triangle, ogon – объект класса Octagon.

**Тесты:**

sq.v\_arr[0] = {0,0};

sq.v\_arr[1] = {1,1};

sq.v\_arr[2] = {0,1};

sq.v\_arr[3] = {1,0};

tr.v\_arr[0] = {1.0,0.0};

tr.v\_arr[1] = {0.0,0.0};

tr.v\_arr[2] = {0.5,sqrt(0.75)};

ogon.v\_arr[0] = {sqrt(2.0),sqrt(2.0)+2.0};

ogon.v\_arr[1] = {sqrt(2.0)+1.0,1.0};

ogon.v\_arr[2] = {-1.0,1.0};

ogon.v\_arr[3] = {0.0,sqrt(2.0)+2.0};

ogon.v\_arr[4] = {0.0,0.0};

ogon.v\_arr[5] = {sqrt(2.0),0.0};

ogon.v\_arr[6] = {-1.0,1.0+sqrt(2.0)};

ogon.v\_arr[7] = {sqrt(2.0)+1,sqrt(2.0)+1.0};

**4.**

**Результат теста:**

Figure 0: (1,0) (0,0) (0.5,0.866025)

Figure 1: (0,0) (1,1) (0,1) (1,0)

Figure 2: (1.41421,3.41421) (2.41421,1) (-1,1) (0,3.41421) (0,0) (1.41421,0)

(-1,2.41421) (2.41421,2.41421)

Square: 11.0899

**5.** В программе создается 3 шаблонных класса Square, Triangle, Octagon, также создается метафункция print для печати объекта каждого класса, внутри каждого класса написан constexpr метод вычисляющий площадь равностороннего объекта, создана метафункция print\_tuple, принимающая tuple фигур и печатающая каждую фигуру, написана метафункция square\_tuple, возвращающая сумму площадей фигур из tuple.