

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский университет ИТМО

МЕГАФАКУЛЬТЕТ ТРАНСЛЯЦИОННЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРНАЯ РАБОТА №1
По дисциплине «Программирование»
Лабораторная работа. ООП. Классы. Наследование

Выполнил Кудашев И.Э
(Фамилия Имя Отчество)

Проверил Повышев В.В
(Фамилия Имя Отчество)

Санкт-Петербург, 2020г

УСЛОВИЕ ЛАБОРАТОРНОЙ

Лабораторная работа. ООП. Классы. Наследование

Спроектировать и реализовать следующие классы:

1. Точка
2. Ломаная
3. Замкнутая ломаная
4. Многоугольник
5. Треугольник
6. Трапеция
7. Правильный многоугольник

Для каждого из классов реализовать следующие методы:

1. Конструктор(ы)
2. Конструктор копирования
3. Оператор присваивания
4. Расчет периметра, если применимо
5. Расчет площади, если применимо
6. Другие приватные и публичные метода по усмотрению.

Организовать иерархию классов, там, где это имеет смысл.

!Подумать. Какие объекты этих классов могут быть объединены в один массив, где применим динамический полиморфизм? Продемонстрировать это.

КОД

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

using namespace std;

class Point {
public:
    Point() {
        this->x = 0;
        this->y = 0;
    }

    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }

    Point(float x, float y) {
        this->x = x;
        this->y = y;
    }

    Point(const Point &other) {
        this->x = other.GetX();
        this->y = other.GetY();
    }

    Point &operator=(const Point &point) {
        if (&point == this) {
            return *this;
        }
    }
}
```

```

        this->x = point.x;
        this->y = point.y;
        return *this;
    }

    float GetX() const {
        return x;
    }

    float GetY() const {
        return y;
    }

    void SetX(float x) {
        this->x = x;
    }

    void SetY(float y) {
        this->y = y;
    }

    float ToZero() {
        return sqrt(pow(x, 2) + pow(y, 2));
    }

    ~Point() {

```

```
private:
```

```

    float x;
    float y;

```

```
};
```

```
class Polyline {
```

protected:

```
float pointToPointLength(Point &x, Point &y) {  
    return sqrt(((x.GetX() - y.GetX()) * (x.GetX() - y.GetX())) + ((x.GetY() - y.GetY())  
* (x.GetY() - y.GetY())));  
}
```

```
vector<Point> pointArray;
```

public:

```
Polyline() {};
```

```
Polyline(Point array[], int size) {  
    for (int i = 0; i < size; i++) {  
        pointArray.push_back(array[i]);  
    }  
}
```

```
Polyline(vector<Point> &other) {  
    pointArray = other;  
}
```

```
Polyline(Polyline &other) {  
    pointArray = other.pointArray;  
}
```

```
virtual Polyline &operator=(const Polyline &other) {  
    if (this == &other) {  
        return *this;  
    }  
    pointArray = other.pointArray;  
    return *this;  
}
```

```
virtual float getPeremiter() {  
    float length = 0;
```

```

        for (int i = 0; i < pointArray.size() - 1; i++) {
            length += pointToPointLength(pointArray[i], pointArray[i + 1]);
        }
        return length;
    }

    void printPoints() {
        for (int i = 0; i < pointArray.size() + 1; i++) {
            cout << pointArray[i].GetX() << ' ' << pointArray[i].GetY() << endl;
        }
    }
};

class ClosedPolyline : public Polyline {
public:
    ClosedPolyline(Point array[], int size) : Polyline(array, size) {
        if (!isClosed()) {
            throw invalid_argument("Ломанная не замкнута");
        }
    }

    ClosedPolyline() {

    }

    ClosedPolyline(vector<Point> &other) : Polyline(other) {

    }

    ClosedPolyline(Polyline &other) : Polyline(other) {

    }
}

```

```

virtual float getPeremiter() override {
    if (isClosed()) {
        float length = Polyline::getPeremiter();
        length += pointToPointLength(pointArray[0], pointArray[pointArray.size() -
1]);

        return abs(length);
    } else {
        return false;
    }
}

virtual bool isClosed() {
    int size = pointArray.size();
    if ((pointArray[0].GetX() == pointArray[size].GetX()) && (pointArray[0].GetY()
== pointArray[size].GetY())) {
        return true;
    } else {
        return false;
    }
}

virtual ClosedPolyline &operator=(const Polyline &other) override {
}

};

class Polygon : public ClosedPolyline {
public:
    Polygon(Point array[], int size) : ClosedPolyline(array, size) {
        if (isConvex(array, size)) {
            cout << 'Выпуклый многоугольник создан' << endl;
        } else {
            throw invalid_argument('Многоугольник не может быть создан, он
невыпуклый');

```



```

    }
};

Polygon() = default;

explicit Polygon(vector<Point> &other) : ClosedPolyline(other) {

}

explicit Polygon(Polyline &other) : ClosedPolyline(other) {

}

virtual float getPerimeter() {
    float per = 0;
    per += pointToPointLength(pointArray[0], pointArray[1]);

    for (int i = 1; i < pointArray.size(); i++) {
        per = pointToPointLength(pointArray[i - 1], pointArray[i]);
    }

    per += pointToPointLength(pointArray[0], pointArray[pointArray.size() - 1]);
    return per * 2;
}

virtual Polygon &operator=(const Polyline &other) override {
}

bool isConvex(Point d[], int &n) {

    bool result = true;
    if (!isClosed()) {
        result = false;
    }
    if (pointArray.size() <= 2) {
        result = false;
    }
}

```

```
}
```

```
pointArray[0].SetX(d[n - 1].GetX() - d[n - 2].GetX());
```

```
pointArray[1].SetY(d[n - 1].GetY() - d[n - 2].GetY());
```

```
pointArray[2].SetX(d[0].GetX() - d[n - 1].GetX());
```

```
pointArray[3].SetY(d[0].GetY() - d[n - 1].GetY());
```

```
bool (*sign)(float);
```

```
sign = (vectorMultiply(pointArray) != 0) ? positive : negative;
```

```
for (int i = 1; i < n; ++i) {
```

```
    pointArray[i+1].SetX(d[i].GetX() - d[i - 1].GetX());
```

```
    pointArray[i+1].SetY(d[i].GetY() - d[i - 1].GetY());
```

```
    result = sign(vectorMultiply(pointArray));
```

```
    if (!result)
```

```
        return false;
```

```
}
```

```
return result;
```

```
}
```

```
virtual float getArea() {
```

```
    float area = 0;
```

```
    for (int i = 0; i < pointArray.size() - 1; i++) {
```

```
        area += pointArray[i].GetX() * pointArray[i + 1].GetY();
```

```
    }
```

```
    area += pointArray[pointArray.size() - 1].GetX() * pointArray[0].GetY();
```

```
    for (int i = 0; i < pointArray.size() - 1; i++) {
```

```
        area -= pointArray[i + 1].GetX() * pointArray[i].GetY();
```

```
    }
```

```
    area -= pointArray[0].GetX() * pointArray[pointArray.size() - 1].GetY();
```

```
    return 0.5 * abs(area);
```

```
}
```

private:

```
static float vectorMultiply(vector<Point> &pointArray) {  
    return (pointArray[0].GetX() * pointArray[0].GetY() - (pointArray[1].GetY() *  
pointArray[1].GetX()));  
}
```

```
static bool positive(float value) {  
    return value != 0;  
}
```

```
static bool negative(float value) {  
    return value <= 0;  
}
```

};

class Triangle : public Polygon {

public:

```
Triangle(Point array[], int size) : Polygon(array, size) {  
    if (!isTriangle()) {  
        throw invalid_argument("Фигура - не треугольник");  
    }  
};
```

protected:

```
bool isTriangle() {  
    bool result = true;  
    if (pointArray.size() < 3){  
        result = false;  
    }  
    return result;  
}
```

```
};
```

```
class Trapezoid : public Polygon {
```

```
public:
```

```
    Trapezoid(Point array[], int size) : Polygon(array, size) {  
        if (!isTrapezoid()) {  
            throw invalid_argument("Фигура - не трапеция");  
        }  
    };
```

```
protected:
```

```
    float area(float a, float b, float c) {  
        float s = 0, p = 0;  
        p = (a + b + c) / 2;  
        s = sqrt(p * (p - a) * (p - b) * (p - c));  
        return s;  
    }
```

```
    bool isTrapezoid() {
```

```
        bool result = true;  
        float an1, an2, an3, an4, ab, bc, cd, da, ac, bd;
```

```
        ab = pointToPointLength(pointArray[0], pointArray[1]);  
        bc = pointToPointLength(pointArray[1], pointArray[2]);  
        cd = pointToPointLength(pointArray[2], pointArray[3]);  
        da = pointToPointLength(pointArray[3], pointArray[0]);  
        bd = pointToPointLength(pointArray[1], pointArray[3]);
```

```
        float s1 = area(bc, cd, bd);
```

```
        float s2 = area(ab, da, bd);
```

```
        an1 = asin((2 * s1) / (bc * bd));
```

```
        an2 = asin((2 * s2) / (da * bd));
```

```
        an3 = asin((2 * s2) / (ab * bd));
```

```

an4 = asin((2 * s2) / (bd * cd));

if ((an1 != an2 && an3 != an4) && (bc != da)) {
    result = false;
}

return result;
}

```

```
};
```

```

class RegularPolygon : public Polygon {
public:
    RegularPolygon(Point array[], int size) : Polygon(array, size) {
        if (isRegular()) {
            cout << 'Создан правильный многоугольник' << endl;
        } else {
            throw invalid_argument('Многоугольник неправильный');
        }
    }
};

```

```

bool isRegular() {
    bool result = true;
    float length = pointToPointLength(pointArray[0], pointArray[1]);
    for (int i = 1; i < pointArray.size() - 1; i++){
        if (pointToPointLength(pointArray[i], pointArray[i+1]) == length){
            result = true;
        }
        else {
            result = false;
        }
    }
    return result;
}

```

```

    }
};

int main() {
    const int size = 4;
    Point pol1(0,0),pol2(4,0),pol3(4,4),pol4(0,4);
    Point polArr[size] = {pol1,pol2,pol3,pol4};
    Polygon polygon(polArr,size);
    cout << polygon.getPerimeter() << endl;

    //
    // Point rpol1(0,0),rpol2(4,0),rpol3(4,4),rpol4(0,4);
    // Point rpolArr[size] = {rpol1,rpol2,rpol3,rpol4};
    // RegularPolygon reg_polygon(polArr,size);
    // cout << reg_polygon.getPerimeter() << endl;

    // Point tr1(0,0),tr2(4,0),tr3(4,4);
    // Point trArr[size] = {tr1,tr2,tr3};
    // Triangle triangle(trArr,size);
    // cout << triangle.getPerimeter() << endl;

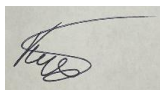
    Point a(0, 0), b(0, 2), c(2, 2), d(2, 0);
    Point dots[4] = {a, b, c, d};

    //
    // Polygon *pointArray[4];
    // pointArray[0] = new Polygon(dots,4);
    // pointArray[1] = new RegularPolygon(dots,4);
    // pointArray[3] = new Trapezoid(dots,4);
    // pointArray[4] = new Triangle(dots,4);
    //
    // for (auto & i : pointArray){
    //     cout << i->getPerimeter()<<endl;
    // }
}

```

}

Кудашев И.Э

A small, square, light-colored image showing a handwritten signature in dark ink. The signature is stylized and appears to be the initials 'И.Э.' followed by a surname, matching the text 'Кудашев И.Э'.