Ponto 2D / Vetor

Representação de pontos e vetores como pontos.

Estrutura

```
const double EPS = 1e-9;
const double PI = acos(-1);

struct point{
    double x, y;

    point(double _x, double _y) : x(_x), y(_y) {} ;
    point(){ x=y=0.0; };

    // Operadores
    bool operator <( point other ) const {
        if(fabs( x-other.x )>EPS) return x<other.x;
        else return y < other.y;
    }
    bool operator ==(point other) const {
        return fabs(x-other.x) < EPS && fabs(y-other.y)<EPS;
    }
    point operator +(point other) const {
        return point( x+other.x, y+other.y);
    }
    point operator -(point other) const {
        return point( x-other.x, y-other.y );
    }
    point operator *(double k) const {
        return point(x*k, y*k);
    }

    // Norma do vetor em relação a origem
    double norm() { return hypot(x, y); }

    // Vetor normalizado
    point normalized(){ return point(x, y)*(1.0/norm()); }

    // Angulo do vetor com a origem
    double angle() { return atan2(y, x); }

    // Angulo polar em relação a origem
    double polarAngle(){
        double a = atan2(y, x);
        return a < 0 ? a + 2*PI : a;
    }
};
```

Funções

```
// Distância entre 2 pontos
double dist(point p1, point p2){
    return hypot(p1.x-p2.x, p1.y-p2.y);
}

// Produto interno
double inner(point p1, point p2){
    return p1.x*p2.x+p1.y*p2.y;
}

// Produto vetorial entre 2 pontos (apenas a componente z)
double cross(point p1, point p2){
    return p1.x*p2.y-p1.y*p2.x;
}

// Teste se os pontos estão no sentido antihorário (DISCRIMINANTE)
bool ccw(point p, point q, point r){
    return cross(q-p, r-p)>0;
}

// Teste de colinearidade entre os pontos
bool collinear(point p, point q, point r){
    return fabs(cross(q-p, r-p))<EPS;
}

// Rotação do ponto em relação a origem
point rotate(point p, double rad){
    return point(p.x*cos(rad)-p.y*sin(rad),
                 p.x*sin(rad)+p.y*cos(rad));
}

// Angulo formado entre vetores a e b com o de origem
double angle(point a, point o, point b){
    return acos(inner(a-o, b-o)/(dist(o, a)*dist(o,b)));
}

// Ponto q está dentro no segmento p r
bool between(point p, point q, point r){
    return collinear(p, q, r) && inner(p-q, r-q)<=0;
}

// Ponto formado pela intersecção das retas a b e A B
point lineIntersectSeg(point p, point q, point A, point B){
    double c = cross(A-B, p-q);
    double a = cross(A, B);
    double b = cross(p, q);
    return ( (p-q)*(a/c) ) - ( (A-B)*(b/c) );
}
```

```
// Teste de paralelidade
bool parallel(point a, point b){
    return fabs(cross(a, b))<EPS;
}

// Verifica se segmentos a b e p q se interceptam
bool segIntersects(point a, point b, point p, point q){
    if(parallel(a-b, p-q)){
        return between(a, p, q) || between(a, q, b) ||
                between(p, a, q) || between(p, b, q);
    }
    point i = lineIntersectSeg(a, b, p, q);
    return between(a, i, b) && between(p, i, q);
}

// Ponto mais próximo entre p e o segmento de reta a b
point closetToSegment(point p, point a, point b){
    double u = inner(p-a, b-a)/inner(b-a, b-a);
    if(u < 0.0) return a;
    if(u > 1.0) return b;
    return a+((b-a)*u);
}
```