```cpp
const double pi = acos(-1);

int cmp(double a, double b = 0){
  if (fabs(a-b)<1e-8) return 0;
  if (a<b) return -1;
  return 1;
}

struct pt {
  double x,y;
  explicit pt(double x = 0, double y = 0): x(x), y(y) {}

  pt operator +(pt q){ return pt(x + q.x, y + q.y); }
  pt operator -(pt q){ return pt(x - q.x, y - q.y); }
  pt operator *(double t){ return pt(x * t, y * t); }
  pt operator /(double t){ return pt(x / t, y / t); }
  double operator *(pt q){ return x * q.x + y * q.y; }
  double operator %(pt q){ return x * q.y - y * q.x; }

  int cmp(pt q) const {
  if (int t = ::cmp(x, q.x)) return t;
  return ::cmp(y, q.y);
  }
  bool operator ==(pt q) const { return cmp(q) == 0; }
  bool operator !=(pt q) const { return cmp(q) != 0; }
  bool operator < (pt q) const { return cmp(q) < 0; }
};

struct reta {
  pt ini,fim;
  reta(){}
  reta(pt ini, pt fim): ini(ini), fim(fim) {}
};

struct eq_reta {
  double A,B,C;  /* Ax + By + C = 0 */

  void init(reta p){
  pt aux = p.ini - p.fim;
  A = aux.y;
  B = -aux.x;
  C = -A*p.ini.x - B*p.ini.y;
  }
  eq reta(reta p){ init(p); }
};

typedef vector<pt> poly;
typedef pair<pt,double> circ;

pt normal(pt v){ return pt(-v.y,v.x); }
double norma(pt v){ return hypot(v.x, v.y); }
pt versor(pt v){ return v/norma(v); }
double anglex(pt v){ return atan2(v.y, v.x); }
double angle(pt v1, pt v2){ /* angulo orientado ccw */
  return atan2(v1%v2 , v1*v2);
}
double triarea(pt a, pt b, pt c){ /* area c/ sinal */
  return ((b-a)%(c-a))/2.0; /* area>0 = ccw ; area<0 = cw */
}
int ccw(pt a, pt b, pt c){ /* b-a em relacao a c-a */
  return cmp((b-a)%(c-a)); /* ccw=1 ; cw=-1 ; colinear=0 */
  /* equivalente a cmp(triarea(a,b,c)), mas evita divisao */
}
pt projecao(pt v, pt w){ /* proj de v em w */
  double alfa = (v*w)/(w*w);
  return w*alfa;
}

pt pivo;
/* ordena em sentido horario */
```

```
 71  bool cmp radial(pt a, pt b){
 72    int aux = ccw(pivo, a,b);
 73    return ((aux<0) || (aux==0 && norma(a-pivo)<norma(b-pivo)));
 74  }
 75  bool cmp pivo(pt p, pt q){ /* pega o de menor x e y */
 76    int aux = cmp( p.x, q.x );
 77    return ((aux<0) || (aux==0 && cmp( p.y, q.y )<0));
 78  }
 79  /* usar poly& p reduz tempo, mas desordena o conj de pontos */
 80  poly graham(poly p){
 81    int i,j,n = p.size();
 82    poly g;
 83
 84    /* ordena e torna o conj de pontos um poligono estrelado */
 85    pivo = *min element(p.begin(), p.end(), cmp pivo);
 86    sort(p.begin(), p.end(), cmp radial);
 87
 88     // para pegar colineares do final do poligono
 89
 90    for(i=n-2 ; i>=0 && ccw(p[0], p[i], p[n-1])==0 ; i--);
 91    reverse(p.begin()+i+1, p.end());
 92
 93
 94    for(i=j=0 ; i<n ; i++) {
 95     // trocar ccw>=0 por ccw>0 para pegar colineares
 96    while( j>=2 && ccw(g[j-2], g[j-1], p[i]) >= 0 ){
 97      g.pop back(); j--;
 98    }
 99    g.push back(p[i]); j++;
100    }
101
102    return g;
103  }
104
105  /* ponto p entre segmento [qr] */
106  int between3(pt p, pt q, pt r){
107    if(cmp((q-p)%(r-p)) == 0) /* colinear */
108    if(cmp((q-p)*(r-p)) <= 0) /* < para nao contar extremos */
109      return 1;
110
111    return 0;
112  }
113
114
115  /* distância de um ponto a uma reta */
116  double distPR(pt p, reta r){
117    pt v = p - r.ini;
118    pt w = r.fim - r.ini;
119
120    pt proj = projecao(v,w);
121    /* (proj+r.ini) é o ponto mais proximo de p,
122     e que pertence à reta r */
123
124     // para segmentos de reta
125
126    if( !between3(proj+r.ini, r.ini, r.fim) )
127      return min( norma(p-r.ini), norma(p-r.fim) );
128
129
130    return norma(v - proj);
131  }
132
133
134  /* ponto p entre segmento [qr] */
135  bool iscolin(pt p, pt q, pt r){
136    if(cmp((q-p)%(r-p)) == 0) /* colinear */
137    return true;
138  return false;
139  }
140
```

```
141 int main() {
142
143   return 0;
144 }
145
```