Geometria

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <cmath>
#include <cstdio>

using namespace std;

#define EPS 1e-8
#define PI acos(-1)
#define Vector Point

struct Point
{
    double x, y;
    Point(){}
    Point(double a, double b) { x = a; y = b; }
    double mod2() { return x*x + y*y; }
    double mod()  { return sqrt(x*x + y*y); }
    double arg()  { return atan2(y, x); }
    Point ort()   { return Point(-y, x); }
    Point unit()  { double k = mod(); return Point(x/k, y/k); }
};

Point operator +(const Point &a, const Point &b) { return Point(a.x + b.x, a.y + b.y); }
Point operator -(const Point &a, const Point &b) { return Point(a.x - b.x, a.y - b.y); }
Point operator /(const Point &a, double k) { return Point(a.x/k, a.y/k); }
Point operator *(const Point &a, double k) { return Point(a.x*k, a.y*k); }

bool operator ==(const Point &a, const Point &b)
{
    return fabs(a.x - b.x) < EPS && fabs(a.y - b.y) < EPS;
}
bool operator !=(const Point &a, const Point &b)
{
    return !(a==b);
}
bool operator <(const Point &a, const Point &b)
{
    if(a.x != b.x) return a.x < b.x;
    return a.y < b.y;
}

```

```
45 //### FUNCIONES BASICAS ##############################################################
46
47 double dist(const Point &A, const Point &B)    { return hypot(A.x - B.x, A.y - B.y); }
48 double cross(const Vector &A, const Vector &B) { return A.x * B.y - A.y * B.x; }
49 double dot(const Vector &A, const Vector &B)   { return A.x * B.x + A.y * B.y; }
50 double area(const Point &A, const Point &B, const Point &C) { return cross(B - A, C - A); }
51
52 // Heron triangulo y cuadrilatero ciclico
53 // http://mathworld.wolfram.com/CyclicQuadrilateral.html
54 // http://www.spoj.pl/problems/QUADAREA/
55
56 double areaHeron(double a, double b, double c)
57 {
58     double s = (a + b + c) / 2;
59     return sqrt(s * (s-a) * (s-b) * (s-c));
60 }
61
62 double circumradius(double a, double b, double c) { return a * b * c / (4 * areaHeron(a, b, c)); }
63
64 double areaHeron(double a, double b, double c, double d)
65 {
66     double s = (a + b + c + d) / 2;
67     return sqrt((s-a) * (s-b) * (s-c) * (s-d));
68 }
69
70 double circumradius(double a, double b, double c, double d) { return sqrt((a*b + c*d) * (a*c + b*d) * (a*d + b*c))  / (4 * areaHeron(a, b, c, d)); }
71
72 //### DETERMINA SI P PERTENECE AL SEGMENTO AB ##########################################
73 bool onSegment(const Point &A, const Point &B, const Point &P)
74 {
75     return abs(area(A, B, P)) < EPS &&
76             P.x >= min(A.x, B.x) && P.x <= max(A.x, B.x) &&
77             P.y >= min(A.y, B.y) && P.y <= max(A.y, B.y);
78 }
79
80 //### DETERMINA SI EL SEGMENTO P1Q1 SE INTERSECTA CON EL SEGMENTO P2Q2 ####################
81 bool intersects(const Point &P1, const Point &P2, const Point &P3, const Point &P4)
82 {
83     double A1 = area(P3, P4, P1);
84     double A2 = area(P3, P4, P2);
85     double A3 = area(P1, P2, P3);
86     double A4 = area(P1, P2, P4);
87
88     if( ((A1 > 0 && A2 < 0) || (A1 < 0 && A2 > 0)) &&
89         ((A3 > 0 && A4 < 0) || (A3 < 0 && A4 > 0)))
90             return true;
91
92     else if(A1 == 0 && onSegment(P3, P4, P1)) return true;
93     else if(A2 == 0 && onSegment(P3, P4, P2)) return true;
94     else if(A3 == 0 && onSegment(P1, P2, P3)) return true;
```

```
95      else if(A4 == 0 && onSegment(P1, P2, P4)) return true;
96      else return false;
97  }
98
99  //### DETERMINA SI A, B, M, N PERTENECEN A LA MISMA RECTA
        ##############################
100 bool sameLine(Point P1, Point P2, Point P3, Point P4)
101 {
102     return area(P1, P2, P3) == 0 && area(P1, P2, P4) == 0;
103 }
104 //### SI DOS SEGMENTOS O RECTAS SON PARALELOS
        ##################################################
105 bool isParallel(const Point &P1, const Point &P2, const Point &P3, const Point &
        P4)
106 {
107     return cross(P2 - P1, P4 - P3) == 0;
108 }
109
110 //### PUNTO DE INTERSECCION DE DOS RECTAS NO PARALELAS
        ################################
111 Point lineIntersection(const Point &A, const Point &B, const Point &C, const
        Point &D)
112 {
113     return A + (B - A) * (cross(C - A, D - C) / cross(B - A, D - C));
114 }
115
116 //### FUNCIONES BASICAS DE POLIGONOS
        ################################################
117 bool isConvex(const vector <Point> &P)
118 {
119     int n = P.size(), pos = 0, neg = 0;
120     for(int i=0; i<n; i++)
121     {
122         double A = area(P[i], P[(i+1)%n], P[(i+2)%n]);
123         if(A < 0) neg++;
124         else if(A > 0) pos++;
125     }
126     return neg == 0 || pos == 0;
127 }
128
129 double area(const vector <Point> &P)
130 {
131     int n = P.size();
132     double A = 0;
133     for(int i=1; i<=n-2; i++)
134         A += area(P[0], P[i], P[i+1]);
135     return abs(A/2);
136 }
137
138 bool pointInPoly(const vector <Point> &P, const Point &A)
139 {
140     int n = P.size(), cnt = 0;
141     for(int i=0; i<n; i++)
142     {
143         int inf = i, sup = (i+1)%n;
144         if(P[inf].y > P[sup].y) swap(inf, sup);
145         if(P[inf].y <= A.y && A.y < P[sup].y)
146             if(area(A, P[inf], P[sup]) > 0)
```

```
147                 cnt++;
148     }
149     return (cnt % 2) == 1;
150 }
151
152 //### CONVEX HULL ##########################################################################
153
154 // O(n log n)
155 vector <Point> ConvexHull(vector <Point> P)
156 {
157     sort(P.begin(),P.end());
158     int n = P.size(),k = 0;
159     Point H[2*n];
160
161     for(int i=0;i<n;++i){
162         while(k>=2 && area(H[k-2],H[k-1],P[i]) <= 0) --k;
163         H[k++] = P[i];
164     }
165
166     for(int i=n-2,t=k;i>=0;--i){
167         while(k>t && area(H[k-2],H[k-1],P[i]) <= 0) --k;
168         H[k++] = P[i];
169     }
170
171     return vector <Point> (H,H+k-1);
172 }
173
174 //### DETERMINA SI P ESTA EN EL INTERIOR DEL POLIGONO CONVEXO A ########################
175
176 bool isInConvexSlow(const vector <Point> &P, const Point &A)
177 {
178     int n = P.size(), pos = 0, neg = 0;
179     for(int i=0; i<n; i++)
180     {
181         double AA = area(A, P[i], P[(i+1)%n]);
182         if(AA < 0) neg++;
183         else if(AA > 0) pos++;
184     }
185     return neg == 0 || pos == 0;
186 }
187
188 // O (log n)
189 bool isInConvex(const vector <Point> &A, const Point &P)
190 {
191     int n = A.size(), lo = 1, hi = A.size() - 1;
192
193     if(area(A[0], A[1], P) <= 0) return 0;
194     if(area(A[n-1], A[0], P) <= 0) return 0;
195
196     while(hi - lo > 1)
197     {
198         int mid = (lo + hi) / 2;
199
200         if(area(A[0], A[mid], P) > 0) lo = mid;
201         else hi = mid;
202     }
```

```
203
204     return area(A[lo], A[hi], P) > 0;
205 }
206
207 // O(n)
208 Point norm(const Point &A, const Point &O)
209 {
210     Vector V = A - O;
211     V = V * 10000000000.0 / V.mod();
212     return O + V;
213 }
214
215 bool isInConvex(vector <Point> &A, vector <Point> &B)
216 {
217     if(!isInConvex(A, B[0])) return 0;
218     else
219     {
220         int n = A.size(), p = 0;
221
222         for(int i=1; i<B.size(); i++)
223         {
224             while(!intersects(A[p], A[(p+1)%n], norm(B[i], B[0]), B[0])) p = (p
                +1)%n;
225
226             if(area(A[p], A[(p+1)%n], B[i]) <= 0) return 0;
227         }
228
229         return 1;
230     }
231 }
```