```
#include <vector>
    #include <algorithm>
 3
    #include <cstdlib>
    #include <complex>
    #include <iostream
    using namespace std;
    typedef complex<long double> point;
    #define sz(a) ((int)(a).size())
    #define all(n) (n).begin(),(n).end()
10
    #define EPS 1e-9
11
    #define 00 1e9
    #define X real()
#define Y imag()
13
14
    #define vec(a,b) ((b)-(a))
#define polar(r,t) ((r)*exp(point(0,(t))))
15
16
17
    #define angle(v) (atan2((v).Y,(v).X))
    #define length(v) ((long double)hypot((v).Y,(v).X))
18
    #define lengthSqr(v) (dot(v,v))
#define dot(a,b) ((conj(a)*(b)).real())
19
20
21
    #define cross(a,b) ((conj(a)*(b)).imag())
    #define rotate(v,t) (polar(v,t))
    #define rotateabout(v,t,a)
                                   (rotate(vec(a,v),t)+(a))
23
    #define reflect(p,m) ((conj((p)/(m)))*(m))
24
    #define normalize(p) ((p)/length(p))
25
26
    #define same(a,b) (lengthSqr(vec(a,b))<EPS)</pre>
27
    #define mid(a,b) (((a)+(b))/point(2,0))
    #define perp(a) (point(-(a).\dot{Y},(a).\dot{X}))
28
    #define colliner pointOnLine
29
30
31
    enum STATE {
32
             IN, OUT, BOUNDRY
33
    };
34
35
    bool intersect(const point &a, const point &b, const point &p, const point &q,
36
                    point &ret) {
37
38
         //handle degenerate cases
39
         double d1 = cross(p - a, b - a);
double d2 = cross(q - a, b - a);
ret = (d1 * q - d2 * p) / (d1 - d2);
40
41
42
         if(fabs(d1 - d2) > EPS) return 1;
43
44
         return 0;
45
    }
46
47
     bool pointOnLine(const point& a, const point& b, const point& p) {
48
         if(same(a,b)) return same(a,p);
         return fabs(cross(vec(a,b),vec(a,p))) < EPS;</pre>
49
50
51
52
     bool pointOnRay(const point& a, const point& b, const point& p) {
         if(same(a,b)) return same(a,p);
53
54
         point v1 = normalize(vec(a,b));
55
         point v2 = normalize(vec(a,p));
56
         return same(v1,v2);
57
    }
58
    bool pointOnSegment(const point& a, const point& b, const point& p) {
59
60
              if (same(a, b))
61
                      return same(a,p);
62
              return pointOnRay(a, b, p) && pointOnRay(b, a, p);
63
64
65
    long double pointLineDist(const point& a, const point& b, const point& p) {
66
             if (same(a,b))
67
                       return hypot(a.X - p.X, a.Y - p.Y);
68
69
```

```
70
              return fabs(cross(vec(a,b),vec(a,p)) / length(vec(a,b)));
 71
 72
 73
     long double pointSegmentDist(const point& a, const point& b, const point& p) {
 74
              if (dot(vec(a,b),vec(a,p)) < EPS)</pre>
 75
                      return length(vec(a,p));
 76
              if (dot(vec(b,a),vec(b,p)) < EPS)</pre>
 77
                      return length(vec(b,p));
 78
              return pointLineDist(a, b, p);
 79
     }
 80
 81
     int lineLatticePointsCount(int x1, int y1, int x2, int y2) {
 82
              return abs(gcd(x1 - x2, y1 - y2)) + 1;
 83
 84
     long double triangleAreaBH(long double b, long double h) {
 85
              return b * h / 2;
 86
 87
 88
     long double triangleArea2sidesAngle(long double a, long double b, long double
 89
 90
              return fabs(a * b * sin(t) / 2);
 91
     }
 92
     long double triangleArea2anglesSide(long double t1, long double t2,
 93
 94
                      long double s) {
 95
              return fabs(s * s * sin(t1) * sin(t2) / (2 * sin(t1 + t2)));
 96
     }
 97
     long double triangleArea3sides(long double a, long double b, long double c) {
 98
              long double s = ((a + b + c) / 2);
return sqrt(s * (s - a) * (s - b) * (s - c));
 99
100
101
     }
102
     long double triangleArea3points(const point& a, const point& b, const point&
103
104
              return fabs(cross(a,b) + cross(b,c) + cross(c,a)) / 2;
105
     }
106
107
      //count interior
108
     int picksTheorm(int a, int b) {
              return a - b / 2 + 1;
109
110
     }
111
112
      //get angle opposite to side a
113
     long double cosRule(long double a, long double b, long double c) {
              // Handle denom = 0
114
              long double res = (b * b + c * c - a * a) / (2 * b * c);
115
116
              if (res > 1)
117
                      res = 1;
118
              if (res < -1)
                      res = -1;
119
120
              return acos(res):
121
122
123
     long double sinRuleAngle(long double s1, long double s2, long double a1) {
              // Handle denom = 0
124
              long double res = s2 * sin(a1) / s1;
125
126
              if (res > 1)
127
                      res = 1;
              if (res < -1)
128
129
                      res = -1:
              return asin(res);
130
131
132
     long double sinRuleSide(long double s1, long double a2) {
133
              // Handle denom = 0
134
              long double res = s1 * sin(a2) / sin(a1);
135
136
              return fabs(res);
```

```
137
     }
138
139
      int circleLineIntersection(const point& p0, const point& p1, const point& cen,
                       long double rad, point& r1, point & r2) {
140
141
142
              if (same(p0,p1)){
143
                       if(fabs(lengthSqr(vec(p0,cen))-(rad*rad)) < EPS)</pre>
144
145
                           r1 = r2 = p0;
                           return 1;
146
                       }
147
148
                       return 0;
149
150
              long double a, b, c, t1, t2;
              a = dot(p1-p0,p1-p0);
151
              b = 2 * dot(p1-p0, p0-cen);
152
              c = dot(p0-cen,p0-cen) - rad * rad;
double det = b * b - 4 * a * c;
153
154
155
              int res:
              if (fabs(det) < EPS)</pre>
156
157
                       det = 0, res = 1;
158
              else if (det < 0)
159
                       res = 0;
160
              else
161
                       res = 2;
              det = sqrt(det);
162
              t1 = (-b + det) / (2 * a);
163
              t2 = (-b - det) / (2 * a);
164
              r1 = p0 + t1 * (p1 - p0);

r2 = p0 + t2 * (p1 - p0);
165
166
167
              return res;
168
169
170
     int circleCircleIntersection(const point &c1, const long double&r1,
                       const point &c2, const long double&r2, point &res1, point
171
     &res2) {
172
              if (same(c1,c2) && fabs(r1 - r2) < EPS) {
173
                       res1 = res2 = c1;
                       return fabs(r1) < EPS ? 1 : 00;
174
175
176
              long double len = length(vec(c1,c2));
177
              if (fabs(len - (r1 + r2)) < EPS || fabs(fabs(r1 - r2) - len) < EPS) {</pre>
                       point d, c;
178
179
                       long double r;
180
                       if (r1 > r2)
181
                               d = vec(c1,c2), c = c1, r = r1;
182
183
                               d = vec(c2,c1), c = c2, r = r2;
                       res1 = res2 = normalize(d) * r + c;
184
185
                       return 1;
186
              if (len > r1 + r2 || len < fabs(r1 - r2))
187
188
                       return 0;
              long double a = cosRule(r2, r1, len);
189
190
              point c1c2 = normalize(vec(c1,c2)) * r1;
191
              res1 = rotate(c1c2,a) + c1;
              res2 = rotate(c1c2, -a) + c1;
192
193
              return 2:
194
195
     void circle2(const point& p1, const point& p2, point& cen, long double& r) {
196
              cen = mid(p1,p2);
197
              r = length(vec(p1,p2)) / 2;
198
199
200
201
     bool circle3(const point& p1, const point& p2, const point& p3, point& cen,
                      long double& r) {
202
              point m1 = mid(p1,p2);
203
              point m2 = mid(p2,p3);
204
```

```
205
               point perp1 = perp(vec(p1,p2));
206
               point perp2 = perp(vec(p2,p3));
207
               bool res = intersect(m1, m1 + perp1, m2, m2 + perp2, cen);
              r = length(vec(cen,p1));
208
209
              return res;
210
      }
211
212
      STATE circlePoint(const point & cen, const long double & r, const point& p) {
              long double lensqr = lengthSqr(vec(cen,p));
213
              if (fabs(lensqr - r * r) < EPS)</pre>
214
                       return BOUNDRY;
215
216
               if (lensqr < r * r)
                       return IN;
217
218
               return OUT;
219
      }
220
221
      int tangentPoints(const point & cen, const long double & r, const point& p,
              point &r1, point &r2) {
STATE s = circlePoint(cen, r, p);
222
223
              if (s != OUT) {
224
225
                       r1 = r2 = p;
226
                       return s == BOUNDRY;
227
              point cp = vec(cen,p);
228
               long double h = length(cp);
229
230
               long double a = acos(r / h);
231
               cp = normalize(cp) * r;
232
               r1 = rotate(cp,a) + cen;
233
              r2 = rotate(cp,-a) + cen;
234
               return 2;
235
      }
236
      // minimum enclosing circle //init p array with the points and ps with the number of points % \left( 1\right) =\left( 1\right) ^{2}
237
238
      //cen and rad are result circle
239
240
      //you must call random_shuffle(p,p+ps); before you call mec
241
      #define MAXPOINTS 100000
      point p[MAXPOINTS], r[3], cen;
242
243
      int ps, rs;
244
      long double rad;
245
246
      void mec() {
247
               if (rs == 3) {
                       circle3(r[0], r[1], r[2], cen, rad);
248
249
                       return;
250
251
               if (rs == 2 && ps == 0) {
                       circle2(r[0], r[1], cen, rad);
252
253
                       return:
254
              if (!ps) {
255
                       cen = r[0];
256
                       rad = 0;
257
258
                       return:
259
               }
260
               ps--;
261
              mec();
              if (circlePoint(cen, rad, p[ps]) == OUT) {
262
263
                       r[rs++] = p[ps];
264
                       mec();
265
                       rs--;
266
              }
267
              ps++;
268
269
270
      //to check if the points are sorted anti-clockwise or clockwise
271
      //remove the fabs at the end and it will return -ve value if clockwise
272
273
      long double polygonArea(const vector<point>&p) {
```

```
274
              long double res = 0;
275
              for (int i = 0; i < sz(p); i++) {
276
                      int j = (i + 1) \% sz(p);
                      res += cross(p[i],p[j]);
277
278
279
              return fabs(res) / 2;
280
281
282
     // return the centroid point of the polygon
     // The centroid is also known as the "centre of gravity" or the "center of
283
     mass". The position of the centroid
      // assuming the polygon to be made of a material of uniform density.
284
285
     point polyginCentroid(vector<point> &polygon) {
              point res(0, 0);
286
287
              long double a = 0;
288
289
              for (int i = 0; i < (int) polygon.size(); i++) {
290
                      int j = (i + 1) % polygon.size();
291
                      res.X += (polygon[i].X + polygon[j].X) * (polygon[i].X *
292
     polygon[j].Y
293
                                       - polygon[j].X * polygon[i].Y);
294
295
                      res.Y += (polygon[i].Y + polygon[j].Y) * (polygon[i].X *
     polygon[j].Y
                                        - polygon[j].X * polygon[i].Y);
296
297
                      a += polygon[i].X * polygon[j].Y - polygon[i].Y * polygon
298
     [j].X;
              }
299
300
301
              a *= 0.5;
              res.X /= 6 * a;
res.Y /= 6 * a;
302
303
304
305
              return res;
306
307
308
     int picksTheorm(vector<point>& p) {
              long double area = 0;
309
310
              int bound = 0;
              for (int i = 0; i < sz(p); i++) {
311
312
                      int j = (i + 1) \% sz(p);
                      area += cross(p[i],p[j]);
313
314
                      point v = vec(p[i],p[j]);
315
                      bound += abs(\_gcd((int) v.X, (int) v.Y));
316
              area /= 2;
317
              area = fabs(area);
318
              return round(area - bound / 2 + 1);
319
320
321
     void polygonCut(const vector<point>& p, const point&a, const point&b, vector<</pre>
322
323
                      point>& res) {
324
              res.clear();
325
              for (int i = 0; i < sz(p); i++) {
                      int j = (i + 1) \% sz(p);
326
                      bool in1 = cross(vec(a,b),vec(a,p[i])) > EPS;
327
                      bool in2 = cross(vec(a,b),vec(a,p[j])) > EPS;
328
329
                      if (in1)
330
                               res.push_back(p[i]);
                      if (in1 ^ in2) {
331
                               point r;
332
                               intersect(a, b, p[i], p[j], r);
333
334
                               res.push_back(r);
335
                      }
              }
336
337
     }
338
```

```
//assume that both are anti-clockwise
339
340
      void convexPolygonIntersect(const vector<point>& p, const vector<point>& q,
341
                        vector<point>& res) {
342
               res = q;
               for (int i = 0; i < sz(p); i++) {
    int j = (i + 1) % sz(p);</pre>
343
344
345
                        vector<point> temp;
                        polygonCut(res, p[i], p[j], temp);
346
347
                        res = temp;
                        if (res.empty())
348
349
                                 return;
350
               }
351
      }
352
      void voronoi(const vector<point> &pnts, const vector<point>& rect, vector<</pre>
353
354
                        vector<point> > &res) {
               res.clear();
355
               for (int i = 0; i < sz(pnts); i++) {
356
                        res.push_back(rect);
357
                        for (int j = 0; j < sz(pnts); j++) {
    if (j == i)</pre>
358
359
360
                                          continue;
                                 point p = perp(vec(pnts[i],pnts[j]));
361
                                 point m = mid(pnts[i],pnts[j]);
362
                                 vector<point> temp;
363
364
                                 polygonCut(res.back(), m, m + p, temp);
365
                                  res.back() = temp;
366
                        }
               }
367
368
      }
369
370
      STATE pointInPolygon(const vector<point>& p, const point &pnt) {
               point p2 = pnt + point(1, 0);
371
               int cnt = 0;
372
               for (int i = 0; i < sz(p); i++) {
373
                        int j = (i + 1) \% sz(p);
374
375
                        if (pointOnSegment(p[i], p[j], pnt))
                                 return BOUNDRY;
376
377
                        point r;
378
                        intersect(pnt, p2, p[i], p[j], r);
379
                        if (!pointOnRay(pnt, p2, r))
380
                                 continue;
                        if (same(r,p[i]) || same(r,p[j]))
        if (fabs(r.Y - min(p[i].Y, p[j].Y)) < EPS)</pre>
381
382
383
                                          continue;
384
                        if (!pointOnSegment(p[i], p[j], r))
385
                                 continue;
386
                        cnt++:
387
               return cnt & 1 ? IN : OUT;
388
389
      }
390
391
      struct cmp {
               point about;
392
393
               cmp(point c) {
394
                        about = c;
395
396
               bool operator()(const point& p, const point& q) const {
397
                        double cr = cross(vec(about,p),vec(about,q));
398
                        if (fabs(cr) < EPS)</pre>
399
                                 return make_pair(p.Y, p.X) < make_pair(q.Y, q.X);</pre>
400
                        return cr > 0:
401
               }
402
403
      void sortAntiClockWise(vector<point>& pnts){
404
               point mn(1 / 0.0, 1 / 0.0);

for (int i = 0; i < sz(pnts); i++)
405
406
                    if (make_pair(pnts[i].Y, pnts[i].X) < make_pair(mn.Y, mn.X))</pre>
407
```

```
408
                         mn = pnts[i];
409
                sort(all(pnts),cmp(mn));
410
411
      }
412
      void convexHull(vector<point> pnts, vector<point> &convex) {
413
                sortAntiClockWise(pnts);
414
415
                convex.clear();
                convex.push_back(pnts[0]);
if (sz(pnts) == 1)
416
417
418
                         return;
                convex.push_back(pnts[1]);
419
                for (int i = 2; i <= sz(pnts); i++) {
    point c = pnts[i % sz(pnts)];
    while (sz(convex) > 1) {
420
421
422
423
                                   point b = convex.back();
                                   point a = convex[sz(convex) - 2];
424
425
                                   if (cross(vec(b,a),vec(b,c)) < -EPS)</pre>
426
                                            break:
427
                                   convex.pop_back();
428
429
                         if (i < sz(pnts))</pre>
430
                                   convex.push_back(pnts[i]);
                }
431
432 }
```