

```
1 bool feq(double x, double y) { return abs(x - y) <= eps; }
2 bool fge(double x, double y) { return x >= y - eps; }
3 double fsqrt(double x) { return feq(x, 0) ? 0 : sqrt(x); }
4
5 // polygon
6
7 struct pt_t {
8     double x, y;
9     pt_t operator+(const pt_t &p) { return { x + p.x, y + p.y }; }
10    pt_t operator-(const pt_t &p) { return { x - p.x, y - p.y }; }
11    pt_t operator*(const double &c) { return { x * c, y * c }; }
12    bool operator<(const pt_t &another) const {
13        return (x != another.x ? x < another.x : y < another.y);
14    }
15 };
16
17 // aX + bY + c = 0
18 struct line_t {
19     double a, b, c;
20 };
21
22 // (X - x)^2 + (Y - y)^2 = r^2
23 struct circle_t {
24     double x, y, r;
25 };
26
27 // normal vector = (a, b), passing p
28 line_t solve_line(double a, double b, pt_t p) {
29     return { a, b, -a * p.x - b * p.y };
30 }
31
32 // passing p, q
33 line_t solve_line(pt_t p, pt_t q) {
34     return solve_line(q.y - p.y, -q.x + p.x, p);
35 }
36
37 // t should be radius
38 pt_t rot(pt_t p, double r) {
39     return {
40         cos(r) * p.x - sin(r) * p.y,
41         sin(r) * p.x + cos(r) * p.y
42     };
43 }
44
45 double norm2(pt_t p) {
46     return p.x * p.x + p.y * p.y;
47 }
48
49 double norm(pt_t p) {
50     return sqrt(norm2(p));
51 }
52
53 // angle [0, 2PI) of vector p to vector q
54 double angle(pt_t p, pt_t q) {
55     p = p * (1.0 / norm(p));
56     q = q * (1.0 / norm(q));
57     double r0 = acos(max(min(p.x * q.x + p.y * q.y, 1.0), -1.0));
58     double r1 = asin(max(min(p.x * q.y - p.y * q.x, 1.0), -1.0));
59     if (r1 >= 0) return r0;
60     else return 2 * M_PI - r0;
61 }
62
63 double dist(line_t l, pt_t p) {
64     return abs(l.a * p.x + l.b * p.y + l.c)
65         / sqrt(l.a * l.a + l.b * l.b);
66 }
67
68 bool on_same_line(pt_t s, pt_t t, pt_t p) {
69     line_t l = solve_line(s, t);
70     if (feq(dist(l, p), 0)) return true;
71     else return false;
```

```

72 }
73
74 bool in_segment(pt_t s, pt_t t, pt_t p) {
75     line_t l = solve_line(s, t);
76     if (feq(dist(l, p), 0)
77         && fge(p.x, min(s.x, t.x))
78         && fge(max(s.x, t.x), p.x)
79         && fge(p.y, min(s.y, t.y))
80         && fge(max(s.y, t.y), p.y)) return true;
81     else return false;
82 }
83
84 // (NAN, NAN) if lines coincide with each other
85 // (INF, INF) if lines are parallel but not coincide
86 pt_t cross_point(line_t l, line_t m) {
87     double d = l.a * m.b - l.b * m.a;
88     if (feq(d, 0)) {
89         if (feq(l.a * m.c - l.c * m.a, 0)) return { INF, INF };
90         else return { NAN, NAN };
91     }
92     else {
93         double x = l.b * m.c - m.b * l.c;
94         double y = l.a * m.c - m.a * l.c;
95         return { x / d, y / -d };
96     }
97 }
98
99 // if size is 0, then not crossed
100 vector<pt_t> cross_point(circle_t f, line_t l) {
101     double d = dist(l, { f.x, f.y });
102     if (!fge(f.r, d)) return {};
103     line_t m = solve_line(l.b, -l.a, { f.x, f.y });
104     pt_t p = cross_point(l, m);
105     if (feq(d, f.r)) return { p };
106     else {
107         pt_t u = { l.b, -l.a };
108         pt_t v = u * (sqrt(pow(f.r, 2) - pow(d, 2)) / norm(u));
109         return { p + v, p - v };
110     }
111 }
112
113 // if size is 0, then not crossed
114 vector<pt_t> cross_point(circle_t f, circle_t g) {
115     line_t l = {
116         -2 * f.x + 2 * g.x,
117         -2 * f.y + 2 * g.y,
118         (f.x * f.x + f.y * f.y - f.r * f.r) - (g.x * g.x + g.y * g.y - g.r * g.r)
119     };
120     return cross_point(f, l);
121 }
122
123 // tangent points of f through p
124 // if size is 0, then p is strictly contained in f
125 // if size is 1, then p is on f
126 // otherwise size is 2
127 vector<pt_t> tangent_point(circle_t f, pt_t p) {
128     vector<pt_t> ret;
129     double d2 = norm2(pt_t({ f.x, f.y }) - p);
130     double r2 = d2 - f.r * f.r;
131     if (fge(r2, 0)) {
132         circle_t g = { p.x, p.y, fsqrt(r2) };
133         ret = cross_point(f, g);
134     }
135     return ret;
136 }
137
138 // tangent lines of f through p
139 // if size is 0, then p is strictly contained in f
140 // if size is 1, then p is on f
141 // otherwise size is 2
142 vector<line_t> tangent_line(circle_t f, pt_t p) {

```

```

143     vector<pt_t> qs = tangent_point(f, p);
144     vector<line_t> ret(qs.size());
145     Loop(i, ret.size()) {
146         ret[i] = solve_line(qs[i].x - f.x, qs[i].y - f.y, qs[i]);
147     }
148     return ret;
149 }
150
151 // tangent points on f through which there is a line tangent to g
152 // if size is 0, then one is strictly contained in the other
153 // if size is 1, then they are touched inside
154 // if size is 2, then they are crossed
155 // if size is 3, then they are touched outside
156 // otherwise size is 4
157 vector<pt_t> tangent_point(circle_t f, circle_t g) {
158     vector<pt_t> ret;
159     double d2 = norm2({ g.x - f.x, g.y - f.y });
160     vector<double> r2(2);
161     r2[0] = d2 - f.r * f.r + 2 * f.r * g.r;
162     r2[1] = d2 - f.r * f.r - 2 * f.r * g.r;
163     Loop(k, 2) {
164         if (fge(r2[k], 0)) {
165             circle_t g2 = { g.x, g.y, fsqrt(r2[k]) };
166             vector<pt_t> buf = cross_point(f, g2);
167             Loop(i, buf.size()) ret.push_back(buf[i]);
168         }
169     }
170     return ret;
171 }
172
173 // common tangent lines between two circles
174 // if size is 0, then one is strictly contained in the other
175 // if size is 1, then they are touched inside
176 // if size is 2, then they are crossed
177 // if size is 3, then they are touched outside
178 // otherwise size is 4
179 vector<line_t> tangent_line(circle_t f, circle_t g) {
180     vector<pt_t> qs = tangent_point(f, g);
181     vector<line_t> ret(qs.size());
182     Loop(i, ret.size()) {
183         ret[i] = tangent_line(f, qs[i]).front();
184     }
185     return ret;
186 }
187
188 // suppose a.size() >= 3
189 double polygon_area(vector<pt_t> a) {
190     double ret = 0;
191     Loop(i, a.size()) {
192         int j = (i + 1 < a.size() ? i + 1 : 0);
193         ret += a[i].x * a[j].y - a[j].x * a[i].y;
194     }
195     ret = abs(ret) / 2;
196     return ret;
197 }

```