```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef pair<int, int> PII;
#define rep(i, j) for (register int i = 0, i##_end_ = (j); i < i##_end_; ++ i)
#define For(i, j, k) for (register int i = (j), i##_end_ = (k); i <= i##_end_; ++
i)
#define Fordown(i, j, k) for (register int i = (j), i##_end_ = (k); i >=
i##_end_; -- i)
#define Set(a, b) memset(a, b, sizeof(a))
#define Cpy(a, b) memcpy(a, b, sizeof(a))
#define fir first
#define sec second
#define pb(a) push_back(a)
#define mp(a, b) make_pair(a, b)
#define ALL(a) (a).begin(), (a).end()
#define SZ(a) ((int)(a).size())
#define INF (0x3f3f3f3f)
#define INF1 (2139062143)
#define Mod (1000000007)
#define debug(...) fprintf(stderr, __VA_ARGS__)

template <typename T> inline bool chkmax(T &a, T b) { return a < b ? a = b, 1 :
0; }
template <typename T> inline bool chkmin(T &a, T b) { return b < a ? a = b, 1 :
0; }

inline int read()
{
    register int _, __; register char c_;
    for (_ = 0, __ = 1, c_ = getchar(); c_ < '0' || c_ > '9'; c_ = getchar()) if
(c_ == '-') __ = -1;
    for ( ; c_ >= '0' && c_ <= '9'; c_ = getchar()) _ = (_ << 1) + (_ << 3) +
(c_ ^ 48);
    return _ * __;
}

const double eps = 1e-10, Pi = acos(-1.0);

struct Point
{
    double x, y;
    Point(double x = 0, double y = 0): x(x), y(y) {}
};
typedef Point Vector;

struct Circle
{
    Point o; double r;
    Circle(Point o, double r): o(o), r(r) {}
    Point point(double a) { return Point(o.x + cos(a) * r, o.y + sin(a) * r); }
};

int dcmp(double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }
Vector operator + (Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
Vector operator - (Vector A, Vector B) { return Vector(A.x - B.x, A.y - B.y); }
Vector operator * (Vector A, double p) { return Vector(A.x * p, A.y * p); }
Vector operator / (Vector A, double p) { return Vector(A.x / p, A.y / p); }
bool operator < (const Point& a, const Point& b) { return a.x < b.x || (a.x ==
b.x && a.y < b.y); }
bool operator == (const Point& a, const Point& b) { return !dcmp(a.x - b.x) &&
!dcmp(a.y - b.y); }
double Dot(Vector A, Vector B) { return A.x * B.x + A.y * B.y; }
double Length(Vector A) { return sqrt(Dot(A, A)); }
double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) /
Length(B)); }
double Angle(Vector A) { return atan2(A.y, A.x); }
double Cross(Vector A, Vector B) { return A.x * B.y - A.y * B.x; } //B在A的左边为
正、右边为负
double Area2(Point A, Point B, Point C) { return Cross(B - A, C - A); }
Vector Rotate(Vector A, double rad) { return Vector(A.x * cos(rad) - A.y *
sin(rad), A.x * sin(rad) + A.y * cos(rad)); }
Vector Normal(Vector A) { double L = Length(A); return Vector(-A.y / L, A.x /
L); } //向左旋转90度且长度变成1
Point GetLineProjection(Point P, Point A, Point B) {
    Vector v = B - A;
    return A + v * (Dot(v, P - A) / Dot(v, v));
}
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);
    return P + v * t;
}
double DistanceToLine(Point P, Point A, Point B) {
    Vector v1 = B - A, v2 = P - A;
    return fabs(Cross(v1, v2)) / Length(v1);
}
double DistanceToSegment(Point P, Point A, Point B) {
    if (A == B) return Length(P - A);
    Vector v1 = B - A, v2 = P - A, v3 = P - B;
    if (dcmp(Dot(v1, v2)) < 0) return Length(v2);
    if (dcmp(Dot(v1, v3)) > 0) return Length(v3);
    return fabs(Cross(v1, v2)) / Length(v1);
}
int GetLineCircleIntersection(Point A, Point B, Circle C, Point& S1, Point& S2)
{
    double d = DistanceToLine(C.O, A, B);
    register int tmp = dcmp(d - C.r);
    if (tmp > 0) return 0;
    Point P = GetLineProjection(C.O, A, B);
    if (!tmp) { S1 = S2 = P; return 1; }
    double L = sqrt(C.r * C.r - d * d);
    Vector v = (B - A) / Length(B - A);
    S1 = P - v * L, S2 = P + v * L;
    return 2;
}
int GetCircleIntersection(Circle C1, Circle C2, Point& P1, Point& P2) {
    double d = Length(C1.O - C2.O);
    if (!dcmp(d)) { if (!dcmp(C1.r - C2.r)) return -1; return 0; }
    if (dcmp(C1.r + C2.r - d) < 0 || dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;
    double a = Angle(C2.O - C1.O), da = acos((C1.r * C1.r + d * d - C2.r * C2.r)
/ (2 * C1.r * d));
    P1 = C1.point(a - da), P2 = C1.point(a + da);
    if (P1 == P2) return 1;
    return 2;
}
int GetTangents(Point P, Circle C, Point& P1, Point& P2) {
    double d = Length(C.O - P);
    if (dcmp(d - C.r) < 0) return 0;
    double aPC = Angle(P - C.O), theta = acos(C.r / d);
    P1 = C.point(aPC + theta), P2 = C.point(aPC - theta);
    if (P1 == P2) return 1;
    return 2;
}
int GetTangents(Circle C1, Circle C2, Point* a, Point* b) {
    int cnt = 0;
    if (C1.r < C2.r) swap(C1, C2);
    double d = Length(C1.O - C2.O), dif = C1.r - C2.r, sum = C1.r + C2.r;
    if (d < dif) return 0;//内含
    double base = Angle(C2.O - C1.O);
    if (!dcmp(d) && !dcmp(C1.r - C2.r)) return -1;//重合
    if (!dcmp(d - dif)) {//内切
        a[++ cnt] = C1.point(base), b[cnt] = a[cnt];
        return 1;
    }
    //外公切线
    double ang = acos((C1.r - C2.r) / d);
    a[++ cnt] = C1.point(base + ang), b[cnt] = C2.point(base + ang);
    a[++ cnt] = C1.point(base - ang), b[cnt] = C2.point(base - ang);
    if (!dcmp(d - sum)) a[++ cnt] = C1.point(base), b[cnt] = a[cnt];//一条内公切线
    else if (dcmp(d - sum) > 0) {
        ang = acos((C1.r + C2.r) / d);
        a[++ cnt] = C1.point(base + ang), b[cnt] = C2.point(Pi + base + ang);
        a[++ cnt] = C1.point(base - ang), b[cnt] = C2.point(Pi + base - ang);
    }
    return cnt;
}
bool IsPointOnSegment(Point P, Point A, Point B) {
    if (A.x > B.x) swap(A, B);
    if (P.x < A.x || P.x > B.x || dcmp((B.y - A.y) / (B.x - A.x) - (B.y - P.y) /
(B.x - P.x))) return 0;
    return 1;
}
int IsPointInPolygon(Point P, Point* p, int n) {
    int wn = 0;
    Point A, B;
    For(i, 1, n) {
        A = p[i], B = p[i % n + 1];
        if (!IsPointOnSegment(P, A, B)) return -1;
        int k = dcmp(Cross(B - A, P - A)), d1 = dcmp(A.y - P.y), d2 = dcmp(B.y -
P.y);
        if (k > 0 && d1 <= 0 && d2 > 0) ++ wn;
        if (k < 0 && d2 <= 0 && d1 > 0) -- wn;
    }
    if (wn) return 1;
    return 0;
}

int main()
{
    return 0;
```

```
}
```