```cpp
#include <bits/stdc++.h>
using namespace std;

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

typedef Point Vector;

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.x*p);
}
Vector operator / (Vector A, double p) {
    return Vector(A.x/p, A.x/p);
}

bool operator < (const Point& a, const Point b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

const double EPS = 1e-10;

int dcmp(double x) {
    if(fabs(x) < EPS) return 0;
    else return x < 0 ? -1 : 1;
}

bool operator == (const Point& a, const Point& b) {
    return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y);
}

//向量a的极角
double Angle(const Vector& v) {
    return atan2(v.y, v.x);
}

//向量点积
double Dot(Vector A, Vector B) {
    return A.x*B.x + A.y*B.y;
}

//向量长度\share\CodeBlocks\templates\wizard\console\cpp
double Length(Vector A) {
    return sqrt(Dot(A, A));
}

//向量夹角
double Angle(Vector A, Vector B) {
    return acos(Dot(A, B) / Length(A) / Length(B));
}
```

```
//向量叉积
double Cross(Vector A, Vector B) {
    return A.x*B.y - A.y*B.x;
}

//三角形有向面积的二倍
double Area2(Point A, Point B, Point C) {
    return Cross(B-A, C-A);
}

//向量逆时针旋转rad度(弧度)
Vector Rotate(Vector A, double rad) {
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}

//计算向量A的单位法向量。左转90°，把长度归一。调用前确保A不是零向量。
Vector Normal(Vector A) {
    double L = Length(A);
    return Vector(-A.y/L, A.x/L);
}

/************************************************************************
 使用复数类实现点及向量的简单操作

 #include <complex>
 typedef complex<double> Point;
 typedef Point Vector;

 double Dot(Vector A, Vector B) { return real(conj(A)*B)}
 double Cross(Vector A, Vector B) { return imag(conj(A)*B);}
 Vector Rotate(Vector A, double rad) { return A*exp(Point(0, rad)); }

 ************************************************************************/

/****************************************************************************
 * 用直线上的一点p0和方向向量v表示一条指向。直线上的所有点P满足P = P0+t*v;
 * 如果知道直线上的两个点则方向向量为B-A, 所以参数方程为A+(B-A)*t;
 * 当t 无限制时, 该参数方程表示直线。
 * 当t > 0时, 该参数方程表示射线。
 * 当 0 < t < 1时, 该参数方程表示线段。
 ****************************************************************************/

//直线交点,须确保两直线有唯一交点。
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;
}

//点到直线距离
double DistanceToLine(Point P, Point A, Point B) {
    Vector v1 = B - A, v2 = P - A;
    return fabs(Cross(v1, v2) / Length(v1)); //不取绝对值，得到的是有向距离
}
```

```
//点到线段的距离
double DistanceToSegmentS(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);
    else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
    else return fabs(Cross(v1, v2)) / Length(v1);
}

//点在直线上的投影
Point GetLineProjection(Point P, Point A, Point B) {
    Vector v = B - A;
    return A+v*(Dot(v, P-A)/Dot(v, v));
}

//线段相交判定，交点不在一条线段的端点
bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2) {
    double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
    double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
    return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
}

//判断点是否在点段上，不包含端点
bool OnSegment(Point P, Point a1, Point a2) {
    return dcmp(Cross(a1-P, a2-P) == 0 && dcmp((Dot(a1-P, a2-P)) < 0));
}

//计算凸多边形面积
double ConvexPolygonArea(Point *p, int n) {
    double area = 0;
    for(int i = 1; i < n-1; i++)
        area += Cross(p[i] - p[0], p[i+1] - p[0]);
    return area/2;
}

//计算多边形的有向面积
double PolygonArea(Point *p, int n) {
    double area = 0;
    for(int i = 1; i < n-1; i++)
        area += Cross(p[i] - p[0], p[i+1] - p[0]);
    return area/2;
}

/*************************************************************************
 * Morley定理：三角形每个内角的三等分线，相交成的三角形是等边三角形。
 * 欧拉定理：设平面图的定点数，边数和面数分别为V,E,F。则V+F-E = 2;
 *************************************************************************/

struct Circle {
    Point c;
    double r;

    Circle(Point c, double r) : c(c), r(r) {}
    //通过圆心角确定圆上坐标
    Point point(double a) {
```

```cpp
        return Point(c.x + cos(a)*r, c.y + sin(a)*r);
    }
};

struct Line {
    Point p;
    Vector v;
    double ang;
    Line() {}
    Line(Point p, Vector v) : p(p), v(v) {}
    bool operator < (const Line& L) const {
        return ang < L.ang;
    }
};

//直线和圆的交点，返回交点个数，结果存在sol中。
//该代码没有清空sol。
int getLineCircleIntersecion(Line L, Circle C, double& t1, double& t2,
 vector<Point>& sol) {
    double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
    double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*d - C.r*C.r;
    double delta = f*f - 4*e*g;
    if(dcmp(delta) < 0) return 0; //相离
    if(dcmp(delta) == 0) {        //相切
        t1 = t2 = -f / (2*e);
        sol.push_back(C.point(t1));
        return 1;
    }
    //相交
    t1 = (-f - sqrt(delta)) / (2*e);
    sol.push_back(C.point(t1));
    t2 = (-f + sqrt(delta)) / (2*e);
    sol.push_back(C.point(t2));
    return 2;
}

//两圆相交
int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol) {
    double d = Length(C1.c - C2.c);
    if(dcmp(d) == 0) {
        if(dcmp(C1.r - C2.r == 0)) return -1;    //两圆完全重合
        return 0;                                //同心圆，半径不一样
    }
    if(dcmp(C1.r + C2.r - d) < 0) return 0;
    if(dcmp(fabs(C1.r - C2.r) == 0)) return -1;

    double a = Angle(C2.c - C1.c);               //向量C1C2的极角
    double da = acos((C1.r*C1.r + d*d - C2.r*C2.r) / (2*C1.r*d));
    //C1C2到C1P1的角
    Point p1 = C1.point(a-da), p2 = C1.point(a+da);
    sol.push_back(p1);
    if(p1 == p2) return 1;
    sol.push_back(p2);
    return 2;
}
```

```cpp
const double PI = acos(-1);
//过定点做圆的切线
//过点p做圆C的切线，返回切线个数。v[i]表示第i条切线
int getTangents(Point p, Circle C, Vector* v) {
    Vector u = C.c - p;
    double dist = Length(u);
    if(dist < C.r) return 0;
    else if(dcmp(dist - C.r) == 0) {
        v[0] = Rotate(u, PI/2);
        return 1;
    } else {
        double ang = asin(C.r / dist);
        v[0] = Rotate(u, -ang);
        v[1] = Rotate(u, +ang);
        return 2;
    }
}

//两圆的公切线
//返回切线的个数，-1表示有无数条公切线。
//a[i], b[i] 表示第i条切线在圆A，圆B上的切点
int getTangents(Circle A, Circle B, Point *a, Point *b) {
    int cnt = 0;
    if(A.r < B.r) {
        swap(A, B);
        swap(a, b);
    }
    int d2 = (A.c.x - B.c.x)*(A.c.x - B.c.x) + (A.c.y - B.c.y)*(A.c.y - B.c.y);
    int rdiff = A.r - B.r;
    int rsum = A.r + B.r;
    if(d2 < rdiff*rdiff) return 0;   //内含
    double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
    if(d2 == 0 && A.r == B.r) return -1;   //无限多条切线
    if(d2 == rdiff*rdiff) {          //内切一条切线
        a[cnt] = A.point(base);
        b[cnt] = B.point(base);
        cnt++;
        return 1;
    }
    //有外共切线
    double ang = acos((A.r-B.r) / sqrt(d2));
    a[cnt] = A.point(base+ang);
    b[cnt] = B.point(base+ang);
    cnt++;
    a[cnt] = A.point(base-ang);
    b[cnt] = B.point(base-ang);
    cnt++;
    if(d2 == rsum*rsum) {  //一条公切线
        a[cnt] = A.point(base);
        b[cnt] = B.point(PI+base);
        cnt++;
    } else if(d2 > rsum*rsum) {   //两条公切线
        double ang = acos((A.r + B.r) / sqrt(d2));
        a[cnt] = A.point(base+ang);
        b[cnt] = B.point(PI+base+ang);
        cnt++;
        a[cnt] = A.point(base-ang);
        b[cnt] = B.point(PI+base-ang);
        cnt++;
    }
    return cnt;
}

typedef vector<Point> Polygon;

//点在多边形内的判定
int isPointInPolygon(Point p, Polygon poly) {
    int wn = 0;
    int n = poly.size();
    for(int i = 0; i < n; i++) {
        if(OnSegment(p, poly[i], poly[(i+1)%n])) return -1; //在边界上
        int k = dcmp(Cross(poly[(i+1)%n]-poly[i], p-poly[i]));
        int d1 = dcmp(poly[i].y - p.y);
        int d2 = dcmp(poly[(i+1)%n].y - p.y);
        if(k > 0 && d1 <= 0 && d2 > 0) wn++;
        if(k < 0 && d2 <= 0 && d1 > 0) wn++;
    }
    if(wn != 0) return 1;       //内部
    return 0;                   //外部
}

//凸包
/*************************************************************
 * 输入点数组p, 个数为p, 输出点数组ch。 返回凸包顶点数
 * 不希望凸包的边上有输入点，把两个<= 改成 <
 * 高精度要求时建议用dcmp比较
 * 输入点不能有重复点。函数执行完以后输入点的顺序被破坏
 *************************************************************/
int ConvexHull(Point *p, int n, Point* ch) {
    sort(p, p+n);      //先比较x坐标，再比较y坐标
    int m = 0;
    for(int i = 0; i < n; i++) {
        while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    int k = m;
    for(int i = n-2; i >= 0; i++) {
        while(m > k && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    if(n > 1) m--;
    return m;
}

//用有向直线A->B切割多边形poly, 返回“左侧”。 如果退化，可能会返回一个单点或者线段
//复杂度O(n2);
Polygon CutPolygon(Polygon poly, Point A, Point B) {
    Polygon newpoly;
    int n = poly.size();
    for(int i = 0; i < n; i++) {
        Point C = poly[i];
        Point D = poly[(i+1)%n];
        if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push_back(C);
        if(dcmp(Cross(B-A, C-D)) != 0) {
            Point ip = GetLineIntersection(A, B-A, C, D-C);
            if(OnSegment(ip, C, D)) newpoly.push_back(ip);
        }
    }
    return newpoly;
}

//半平面交

//点p再有向直线L的左边。（线上不算）
bool Onleft(Line L, Point p) {
    return Cross(L.v, p-L.p) > 0;
}

//两直线交点，假定交点唯一存在
Point GetIntersection(Line a, Line b) {
    Vector u = a.p - b.p;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.p+a.v*t;
}

int HalfplaneIntersection(Line* L, int n, Point* poly) {
    sort(L, L+n);                //按极角排序

    int first, last;             //双端队列的第一个元素和最后一个元素
    Point *p = new Point[n];     //p[i]为q[i]和q[i+1]的交点
    Line *q = new Line[n];       //双端队列
    q[first = last = 0] = L[0]; //队列初始化为只有一个半平面L[0]
    for(int i = 0; i < n; i++) {
        while(first < last && !Onleft(L[i], p[last-1])) last--;
        while(first < last && !Onleft(L[i], p[first])) first++;
        q[++last] = L[i];
        if(fabs(Cross(q[last].v, q[last-1].v)) < EPS) {
            last--;
            if(Onleft(q[last], L[i].p)) q[last] = L[i];
        }
        if(first < last) p[last-1] = GetIntersection(q[last-1], q[last]);
    }
    while(first < last && !Onleft(q[first], p[last-1])) last--;
    //删除无用平面
    if(last-first <= 1) return 0;   //空集
    p[last] = GetIntersection(q[last], q[first]);

    //从deque复制到输出中
    int m = 0;
    for(int i = first; i <= last; i++) poly[m++] = p[i];
    return m;
}
int main() {

    return 0;
}
```