```cpp
//########################################################################
//############################ GEOMETRIA COMPUTACIONAL ###################
//########################################################################

#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 abs(a.x - b.x) < EPS && abs(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(abs(a.x - b.x) > EPS) return a.x < b.x;
    return a.y + EPS < b.y;
}

//### FUNCIONES BASICAS ######################

double dist(const Point &A, const Point &B)    { return hypot(A.x - B.x, A.y - B.y); }
double cross(const Vector &A, const Vector &B) { return A.x * B.y - A.y * B.x; }
double dot(const Vector &A, const Vector &B)   { return A.x * B.x + A.y * B.y; }
double area(const Point &A, const Point &B, const Point &C) { return cross(B - A, C - A); }

// Heron triangulo y cuadrilatero ciclico
// http://mathworld.wolfram.com/CyclicQuadrilateral.html
// http://www.spoj.pl/problems/QUADAREA/

double areaHeron(double a, double b, double c) {
  double s = (a + b + c) / 2;
  return sqrt(s * (s-a) * (s-b) * (s-c));
}

double circumradius(double a, double b, double c) { return a * b * c / (4 * areaHeron(a, b, c)); }

double areaHeron(double a, double b, double c, double d) {
  double s = (a + b + c + d) / 2;
  return sqrt((s-a) * (s-b) * (s-c) * (s-d));
}

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)); }

//### DETERMINA SI P PERTENECE AL SEGMENTO AB ###############
bool between(const Point &A, const Point &B, const Point &P) {
    return  P.x + EPS >= min(A.x, B.x) && P.x <= max(A.x, B.x) + EPS &&
            P.y + EPS >= min(A.y, B.y) && P.y <= max(A.y, B.y) + EPS;
}

bool onSegment(const Point &A, const Point &B, const Point &P) {
    return abs(area(A, B, P)) < EPS && between(A, B, P);
}

//### DETERMINA SI EL SEGMENTO P1Q1 SE INTERSECTA CON EL SEGMENTO P2Q2 ####################
//funciona para cualquiera P1, P2, P3, P4
bool intersects(const Point &P1, const Point &P2, const Point &P3, const Point &P4) {
```

```
    double A1 = area(P3, P4, P1);
    double A2 = area(P3, P4, P2);
    double A3 = area(P1, P2, P3);
    double A4 = area(P1, P2, P4);

    if( ((A1 > 0 && A2 < 0) || (A1 < 0 && A2 > 0)) &&
        ((A3 > 0 && A4 < 0) || (A3 < 0 && A4 > 0)))
            return true;

    else if(A1 == 0 && onSegment(P3, P4, P1)) return true;
    else if(A2 == 0 && onSegment(P3, P4, P2)) return true;
    else if(A3 == 0 && onSegment(P1, P2, P3)) return true;
    else if(A4 == 0 && onSegment(P1, P2, P4)) return true;
    else return false;
}

//### DETERMINA SI A, B, M, N PERTENECEN A LA MISMA RECTA #
bool sameLine(Point P1, Point P2, Point P3, Point P4) {
  return area(P1, P2, P3) == 0 && area(P1, P2, P4) == 0;
}
//### SI DOS SEGMENTOS O RECTAS SON PARALELOS #############
bool isParallel(const Point &P1, const Point &P2, const Point &P3, const
     Point &P4) {
  return cross(P2 - P1, P4 - P3) == 0;
}

//### PUNTO DE INTERSECCION DE DOS RECTAS NO PARALELAS
    ################################
Point lineIntersection(const Point &A, const Point &B, const Point &C,
    const Point &D) {
    return A + (B - A) * (cross(C - A, D - C) / cross(B - A, D - C));
}

Point circumcenter(const Point &A, const Point &B, const Point &C) {
  return (A + B + (A - B).ort() * dot(C - B, A - C) / cross(A - B, A - C
      )) / 2;
}

//### FUNCIONES BASICAS DE POLIGONOS
    ##############################################
bool isConvex(const vector <Point> &P) {
    int n = P.size(), pos = 0, neg = 0;
    for(int i=0; i<n; i++)
    {
        double A = area(P[i], P[(i+1)%n], P[(i+2)%n]);
        if(A < 0) neg++;
        else if(A > 0) pos++;
    }
    return neg == 0 || pos == 0;
}

double area(const vector <Point> &P) {
    int n = P.size();
    double A = 0;
    for(int i=1; i<=n-2; i++)
        A += area(P[0], P[i], P[i+1]);
    return abs(A/2);
}

bool pointInPoly(const vector <Point> &P, const Point &A) {
    int n = P.size(), cnt = 0;
    for(int i=0; i<n; i++)     {
        int inf = i, sup = (i+1)%n;
        if(P[inf].y > P[sup].y) swap(inf, sup);
        if(P[inf].y <= A.y && A.y < P[sup].y)
            if(area(A, P[inf], P[sup]) > 0)
                cnt++;
    }
    return (cnt % 2) == 1;
}

//### CONVEX HULL
    ###########################################################################

// O(nh)
vector <Point> ConvexHull(vector <Point> S) {
  sort(all(S));

  int it=0;
  Point primero = S[it], ultimo =  primero;

  int n = S.size();
  vector <Point> convex;
  do {
    convex.push_back(S[it]);
    it = (it + 1)%n;
```

```
153
154     for(int i=0; i<S.size(); i++) {
155       if(S[i]!=ultimo && S[i]!=S[it]) {
156         if(area(ultimo, S[it], S[i]) < EPS) it = i;
157       }
158     }
159
160     ultimo=S[it];
161   }while(ultimo!=primero);
162
163   return convex;
164 }
165
166 // O(n log n)
167 vector <Point> ConvexHull(vector <Point> P) {
168     sort(P.begin(),P.end());
169     int n = P.size(),k = 0;
170     Point H[2*n];
171
172     for(int i=0;i<n;++i){
173         while(k>=2 && area(H[k-2],H[k-1],P[i]) <= 0) --k;
174         H[k++] = P[i];
175     }
176
177     for(int i=n-2,t=k;i>=0;--i){
178         while(k>t && area(H[k-2],H[k-1],P[i]) <= 0) --k;
179         H[k++] = P[i];
180     }
181
182     return vector <Point> (H,H+k-1);
183 }
184
185 //### DETERMINA SI P ESTA EN EL INTERIOR DEL POLIGONO CONVEXO A
    ########################
186
187 // O (log n)
188 bool isInConvex(vector <Point> &A, const Point &P) {
189   int n = A.size(), lo = 1, hi = A.size() - 1;
190
191   if(area(A[0], A[1], P) <= 0) return 0;
192   if(area(A[n-1], A[0], P) <= 0) return 0;
193
194   while(hi - lo > 1)
195   {
196     int mid = (lo + hi) / 2;
197
198     if(area(A[0], A[mid], P) > 0) lo = mid;
199     else hi = mid;
200   }
201
202   return area(A[lo], A[hi], P) > 0;
203 }
204
205 // O(n)
206 Point norm(const Point &A, const Point &O)
207 {
208     Vector V = A - O;
209     V = V * 10000000000.0 / V.mod();
210     return O + V;
211 }
212
213 bool isInConvex(vector <Point> &A, vector <Point> &B)
214 {
215     if(!isInConvex(A, B[0])) return 0;
216     else
217     {
218         int n = A.size(), p = 0;
219
220         for(int i=1; i<B.size(); i++)
221         {
222             while(!intersects(A[p], A[(p+1)%n], norm(B[i], B[0]), B[0]))
                  p = (p+1)%n;
223
224             if(area(A[p], A[(p+1)%n], B[i]) <= 0) return 0;
225         }
226
227         return 1;
228     }
229 }
230
231 //##### SMALLEST ENCLOSING CIRCLE O(n)
    ##############################################
232 // http://www.cs.uu.nl/docs/vakken/ga/slides4b.pdf
233 // http://www.spoj.pl/problems/ALIENS/
234
235 pair <Point, double> enclosingCircle(vector <Point> P)
```

```
{
    random_shuffle(P.begin(), P.end());

    Point O(0, 0);
    double R2 = 0;

    for(int i=0; i<P.size(); i++)
    {
        if((P[i] - O).mod2() > R2 + EPS)
        {
            O = P[i], R2 = 0;
            for(int j=0; j<i; j++)
            {
                if((P[j] - O).mod2() > R2 + EPS)
                {
                    O = (P[i] + P[j])/2, R2 = (P[i] - P[j]).mod2() / 4;
                    for(int k=0; k<j; k++)
                        if((P[k] - O).mod2() > R2 + EPS)
                            O = circumcenter(P[i], P[j], P[k]), R2 = (P[
                                k] - O).mod2();
                }
            }
        }
    }
    return make_pair(O, sqrt(R2));
}

//##### CLOSEST PAIR OF POINTS
    ##########################################################
bool XYorder(Point P1, Point P2)
{
  if(P1.x != P2.x) return P1.x < P2.x;
  return P1.y < P2.y;
}
bool YXorder(Point P1, Point P2)
{
  if(P1.y != P2.y) return P1.y < P2.y;
  return P1.x < P2.x;
}
double closest_recursive(vector <Point> vx, vector <Point> vy)
{
  if(vx.size()==1) return 1e20;
  if(vx.size()==2) return dist(vx[0], vx[1]);

  Point cut = vx[vx.size()/2];

  vector <Point> vxL, vxR;
  for(int i=0; i<vx.size(); i++)
    if(vx[i].x < cut.x || (vx[i].x == cut.x && vx[i].y <= cut.y))
      vxL.push_back(vx[i]);
    else vxR.push_back(vx[i]);

  vector <Point> vyL, vyR;
  for(int i=0; i<vy.size(); i++)
    if(vy[i].x < cut.x || (vy[i].x == cut.x && vy[i].y <= cut.y))
      vyL.push_back(vy[i]);
    else vyR.push_back(vy[i]);

  double dL = closest_recursive(vxL, vyL);
  double dR = closest_recursive(vxR, vyR);
  double d = min(dL, dR);

  vector <Point> b;
  for(int i=0; i<vy.size(); i++)
    if(abs(vy[i].x - cut.x) <= d)
      b.push_back(vy[i]);

  for(int i=0; i<b.size(); i++)
    for(int j=i+1; j<b.size() && (b[j].y - b[i].y) <= d; j++)
      d = min(d, dist(b[i], b[j]));

  return d;
}
double closest(vector <Point> points)
{
  vector <Point> vx = points, vy = points;
  sort(vx.begin(), vx.end(), XYorder);
  sort(vy.begin(), vy.end(), YXorder);

  for(int i=0; i+1<vx.size(); i++)
    if(vx[i] == vx[i+1])
      return 0.0;

  return closest_recursive(vx,vy);
}
```

```
// INTERSECCION DE CIRCULOS
vector <Point> circleCircleIntersection(Point O1, double r1, Point O2,
    double r2)
{
  vector <Point> X;

  double d = dist(O1, O2);

  if(d > r1 + r2 || d < max(r2, r1) - min(r2, r1)) return X;
  else
  {
    double a = (r1*r1 - r2*r2 + d*d) / (2.0*d);
    double b = d - a;
    double c = sqrt(abs(r1*r1 - a*a));

    Vector V = (O2-O1).unit();
    Point H = O1 + V * a;

    X.push_back(H + V.ort() * c);

    if(c > EPS) X.push_back(H - V.ort() * c);
  }

  return X;
}

// LINEA AB vs CIRCULO (O, r)
// 1. Mucha perdida de precision, reemplazar por resultados de formula.
// 2. Considerar line o segment

vector <Point> lineCircleIntersection(Point A, Point B, Point O, long
    double r)
{
  vector <Point> X;

  Point H1 = O + (B - A).ort() * cross(O - A, B - A) / (B - A).mod2();
  long double d2 = cross(O - A, B - A) * cross(O - A, B - A) / (B - A).
      mod2();

  if(d2 <= r*r + EPS)
  {
    long double k = sqrt(abs(r * r - d2));

    Point P1 = H1 + (B - A) * k / (B - A).mod();
    Point P2 = H1 - (B - A) * k / (B - A).mod();

    if(between(A, B, P1)) X.push_back(P1);

    if(k > EPS && between(A, B, P2)) X.push_back(P2);
  }

  return X;
}

//### PROBLEMAS BASICOS
    ##################################################################
void CircumscribedCircle()
{
  int x1, y1, x2, y2, x3, y3;
  scanf("%d␣%d␣%d␣%d␣%d␣%d", &x1, &y1, &x2, &y2, &x3, &y3);

  Point A(x1, y1), B(x2, y2), C(x3, y3);

  Point P1 = (A + B) / 2.0;
  Point P2 = P1 + (B-A).ort();
  Point P3 = (A + C) / 2.0;
  Point P4 = P3 + (C-A).ort();

  Point CC = lineIntersection(P1, P2, P3, P4);
  double r = dist(A, CC);

  printf("(%.6lf,␣%.6lf,␣%.6lf)\n", CC.x, CC.y, r);
}

void InscribedCircle()
{
  int x1, y1, x2, y2, x3, y3;
  scanf("%d␣%d␣%d␣%d␣%d␣%d", &x1, &y1, &x2, &y2, &x3, &y3);

  Point A(x1, y1), B(x2, y2), C(x3, y3);

  Point AX = A + (B-A).unit() + (C-A).unit();
  Point BX = B + (A-B).unit() + (C-B).unit();

  Point CC = lineIntersection(A, AX, B, BX);
  double r = abs(area(A, B, CC) / dist(A, B));
```

```
402
403   printf("(%.6lf, %.6lf, %.6lf)\n", CC.x, CC.y, r);
404 }
405
406 vector <Point>  TangentLineThroughPoint(Point P, Point C, long double r)
407 {
408   vector <Point> X;
409
410   long double h2 = (C - P).mod2();
411   if(h2 < r*r) return X;
412   else
413   {
414     long double d = sqrt(h2 - r*r);
415
416     long double m1 = (r*(P.x - C.x) + d*(P.y - C.y)) / h2;
417     long double n1 = (P.y - C.y - d*m1) / r;
418
419     long double n2 = (d*(P.x - C.x) + r*(P.y - C.y)) / h2;
420     long double m2 = (P.x - C.x - d*n2) / r;
421
422     X.push_back(C + Point(m1, n1)*r);
423     if(d != 0) X.push_back(C + Point(m2, n2)*r);
424
425     return X;
426   }
427 }
428
429 void TangentLineThroughPoint()
430 {
431   int xc, yc, r, xp, yp;
432   scanf("%d␣%d␣%d␣%d␣%d", &xc, &yc, &r, &xp, &yp);
433
434   Point C(xc, yc), P(xp, yp);
435
436   double hyp = dist(C, P);
437   if(hyp < r) printf("[]\n");
438   else
439   {
440     double d = sqrt(hyp * hyp - r*r);
441
442     double m1 = (r*(P.x - C.x) + d*(P.y - C.y)) / (r*r + d*d);
443     double n1 = (P.y - C.y - d*m1) / r;
444     double ang1 = 180 * atan(-m1/n1) / PI + EPS;
445     if(ang1 < 0) ang1 += 180.0;
446
447     double n2 = (d*(P.x - C.x) + r*(P.y - C.y)) / (r*r + d*d);
448     double m2 = (P.x - C.x - d*n2) / r;
449     double ang2 = 180 * atan(-m2/n2) / PI + EPS;
450     if(ang2 < 0) ang2 += 180.0;
451
452     if(ang1 > ang2) swap(ang1, ang2);
453
454     if(d == 0) printf("[%.6lf]\n", ang1);
455     else printf("[%.6lf, %.6lf]\n", ang1, ang2);
456   }
457 }
458
459 void CircleThroughAPointAndTangentToALineWithRadius()
460 {
461   int xp, yp, x1, y1, x2, y2, r;
462   scanf("%d␣%d␣%d␣%d␣%d␣%d␣%d", &xp, &yp, &x1, &y1, &x2, &y2, &r);
463
464   Point P(xp, yp), A(x1, y1), B(x2, y2);
465
466   Vector V = (B - A).ort() * r / (B - A).mod();
467
468   Point X[2];
469   int cnt = 0;
470
471   Point H1 = P + (B - A).ort() * cross(P - A, B - A) / (B - A).mod2() +
        V;
472   double d1 = abs(r + cross(P - A, B - A) / (B - A).mod());
473
474   if(d1 - EPS <= r)
475   {
476     double k = sqrt(abs(r * r - d1 * d1));
477
478     X[cnt++] = Point(H1 + (B - A).unit() * k);
479
480     if(k > EPS) X[cnt++] = Point(H1 - (B - A).unit() * k);
481   }
482
483   Point H2 = P + (B - A).ort() * cross(P - A, B - A) / (B - A).mod2() -
        V;
484   double d2 = abs(r - cross(P - A, B - A) / (B - A).mod());
485
```

```
  if(d2 - EPS <= r)
  {
    double k = sqrt(abs(r * r - d2 * d2));

    X[cnt++] = Point(H2 + (B - A).unit() * k);

    if(k > EPS) X[cnt++] = Point(H2 - (B - A).unit() * k);
  }

  sort(X, X + cnt);

  if(cnt == 0) printf("[]\n");
  else if(cnt == 1) printf("[(%.6lf,%.6lf)]\n", X[0].x, X[0].y);
  else if(cnt == 2) printf("[(%.6lf,%.6lf),(%.6lf,%.6lf)]\n", X[0].x, X
      [0].y, X[1].x, X[1].y);
}

void CircleTangentToTwoLinesWithRadius()
{
  int x1, y1, x2, y2, x3, y3, x4, y4, r;
  scanf("%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d", &x1, &y1, &x2, &y2, &x3, &y3, &x4,
       &y4, &r);

  Point A1(x1, y1), B1(x2, y2), A2(x3, y3), B2(x4, y4);

  Vector V1 = (B1 - A1).ort() * r / (B1 - A1).mod();
  Vector V2 = (B2 - A2).ort() * r / (B2 - A2).mod();

  Point X[4];
  X[0] = lineIntersection(A1 + V1, B1 + V1, A2 + V2, B2 + V2);
  X[1] = lineIntersection(A1 + V1, B1 + V1, A2 - V2, B2 - V2);
  X[2] = lineIntersection(A1 - V1, B1 - V1, A2 + V2, B2 + V2);
  X[3] = lineIntersection(A1 - V1, B1 - V1, A2 - V2, B2 - V2);

  sort(X, X + 4);
  printf("[(%.6lf,%.6lf),(%.6lf,%.6lf),(%.6lf,%.6lf),(%.6lf,%.6lf)]\n",
      X[0].x, X[0].y, X[1].x, X[1].y, X[2].x, X[2].y, X[3].x, X[3].y);
}

void CircleTangentToTwoDisjointCirclesWithRadius()
{
  int x1, y1, r1, x2, y2, r2, r;
  scanf("%d␣%d␣%d␣%d␣%d␣%d␣%d", &x1, &y1, &r1, &x2, &y2, &r2, &r);

  Point A(x1, y1), B(x2, y2);

  r1 += r;
  r2 += r;

  double d = dist(A, B);

  if(d > r1 + r2 || d < max(r1, r2) - min(r1, r2)) printf("[]\n");
  else
  {
    double a = (r1*r1 - r2*r2 + d*d) / (2.0*d);
    double b = d - a;
    double c = sqrt(abs(r1*r1 - a*a));

    Vector V = (B-A).unit();
    Point H = A + V * a;

    Point P1 = H + V.ort() * c;
    Point P2 = H - V.ort() * c;

    if(P2 < P1) swap(P1, P2);

    if(P1 == P2) printf("[(%.6lf,%.6lf)]\n", P1.x, P1.y);
    else printf("[(%.6lf,%.6lf),(%.6lf,%.6lf)]\n", P1.x, P1.y, P2.x, P2.
        y);
  }
}
```