//Point Representation

```
#include<bits/stdc++.h>
using namespace std;
   typedef double T;
   typedef complex<T> pt;
   #define x real()
   #define y imag()
   int main(){
      pt p{3,-4}, q= {6,9};
      cout << p.x << " " << p.y << "\n"; // 3 -4
      // Can be printed out of the box
      cout << p << "\n"; // (3,-4)
      pt p2{-3,2};
      //p2.x = 1; // doesn't compile
      p2 = {1,2}; // correct
      cout<<p2<<"\n";
      pt a{3,1}, b{1,-2};
      a += 2.0*b; // a = (5,-3)
      cout<<a<<"\n";
      cout << a*b << " " << a/-b << "\n"; // (-1,-13) (-2.2,-1.4)
      pt p3{4,3};
      // Get the absolute value and argument of point (in [-pi,pi])
      cout << abs(p3) << " " << arg(p3) << "\n"; // 5 0.643501 Make a point from polar coordinates
      cout << polar(2.0, -M_PI/4.0) << "\n"; // (1.41421,-1.41421)
      cout<<M_PI<<"\n";
      cout<<norm(complex<double>(2.0,1.0))<<"\n";
      cout<<(norm(complex<double>(2.0,1.0)) == 5.0)<<"\n";
      cout<<(norm(complex<double>(2.0,1.0)) == 5)<<"\n";
      cout<<(5==5.0)<<"\n";
   }
template <class Y> int sgn(Y x) {return (Y(0) < x) - (x < Y(0));}
struct pt{
   T x,y;
   pt operator+(pt p) {return {x+p.x, y+p.y};}
   pt operator-(pt p) {return {x-p.x, y-p.y};}
   pt operator*(T d) {return {x*d, y*d};}
   pt operator/(T d) {return {x/d, y/d};} // only for floating-
   bool operator==(pt a) {return a.x == x && a.y == y;}
   bool operator!=(pt a) {return a.x != x || a.y != y;}
   T sq(pt p) {return p.x*p.x + p.y*p.y;}
   double abs(pt p) {return sqrt(sq(p));}
};
ostream& operator<<(ostream& os, pt p) {return os << "("<< p.x << "," << p.y << ")";}
int main(){
   pt a{3,4}, b{2,-1};
   pt c=  a*5;
   cout<<c.x<<endl;
   cout << a+b << " " << a-b << "\n"; // (5,3) (1,5)
   cout << a*-1 << " " << b/2 << "\n"; // (-3,-4) (1.5,2)
   cout<<(a==b)<<"\n";
   cout<<(a!=b)<<"\n";
}
```

//Line Representation

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

typedef double T;
typedef complex<T> pt;
#define x real()
#define y imag()

T orient(pt a, pt b, pt c) {return cross(b-a,c-a);}

pt translate(pt v, pt p) {return p+v;}

pt scale(pt c, double factor, pt p) {return c + (p-c)*factor;}

pt rot(pt p, double a) {
   return {p.x*cos(a) - p.y*sin(a), p.x*sin(a) + p.y*cos(a)};
}
pt rotation(pt p, double a) {return p * polar(1.0, a);}

pt perp(pt p) {return {-p.y, p.x};}

T dot(pt v, pt w) {return v.x*w.x + v.y*w.y;}

bool isPerp(pt v, pt w) {return dot(v,w) == 0;}

double angle(pt v, pt w) {
   double cosTheta = dot(v,w) / abs(v) / abs(w);
   return acos(max(-1.0, min(1.0, cosTheta)));
}

T cross(pt v, pt w) {return v.x*w.y - v.y*w.x;}

T orient(pt a, pt b, pt c) {return cross(b-a,c-a);}

bool inAngle(pt a, pt b, pt c, pt p) {
   assert(orient(a,b,c) != 0);
   if (orient(a,b,c) < 0) swap(b,c);
   return orient(a,b,p) >= 0 && orient(a,c,p) <= 0;
}

double orientedAngle(pt a, pt b, pt c) {
   if (orient(a,b,c) >= 0)
      return angle(b-a, c-a);
   else
      return 2*M_PI - angle(b-a, c-a);
}
```

```
bool isConvex(vector<pt> p) {
   bool hasPos=false, hasNeg=false;
   for (int i=0, n=p.size(); i<n; i++) {
      int o = orient(p[i], p[(i+1)%n], p[(i+2)%n]);
      if (o > 0) hasPos = true;
      if (o < 0) hasNeg = true;
   }
   return !(hasPos && hasNeg);
}

struct line{
   pt v;
   T c;
   // From direction vector v and offset c
   line(pt v, T c) : v(v), c(c) {}
   // From equation ax+by=c
   line(T a, T b, T c) : v({b,-a}), c(c) {}
   // From points P and Q
   line(pt p, pt q) : v(q-p), c(cross(v,p)) {}
};

T side(pt p) {return cross(v,p)-c;}

double dist(pt p) {return abs(side(p)) / abs(v);}
double sqDist(pt p) {return side(p)*side(p) / (double)sq(v);}


line perpThrough(pt p) {return {p, p + perp(v)};}



line translate(pt t) {return {v, c + cross(v,t)};}


line shiftLeft(double dist) {return {v, c + dist*abs(v)};}

bool inter(line l1, line l2, pt &out) {
   T d = cross(l1.v, l2.v);
   if (d == 0) return false;
   out = (l2.v*l1.c - l1.v*l2.c) / d; // requires floating-point coordinates
   return true;
}

pt proj(pt p) {return p - perp(v)*side(p)/sq(v);}
pt refl(pt p) {return p - perp(v)*2*side(p)/sq(v);}


bool inDisk(pt a, pt b, pt p) {return dot(a-p, b-p) <= 0;}
bool onSegment(pt a, pt b, pt p) {
   return orient(a,b,p) == 0 && inDisk(a,b,p);
}
```

```
bool properInter(pt a, pt b, pt c, pt d, pt &out) {
    double oa = orient(c,d,a),
    ob = orient(c,d,b),
    oc = orient(a,b,c),
    od = orient(a,b,d);
    // Proper intersection exists iff opposite signs
    if (oa*ob < 0 && oc*od < 0) {
        out = (a*ob - b*oa) / (ob-oa);
        return true;
    }
return false;
}
bool cmpProj(pt p, pt q) {return dot(v,p) < dot(v,q);}
double segPoint(pt a, pt b, pt p) {
    if (a != b) {
        line l(a,b);
        if (l.cmpProj(a,p) && l.cmpProj(p,b)) // if closest to projection
        return l.dist(p); // output distance to line
    }
    return min(abs(p-a), abs(p-b)); // otherwise distance to A or B
}
double segSeg(pt a, pt b, pt c, pt d) {
    pt dummy;
    if (properInter(a,b,c,d,dummy))
        return 0;
    return min({segPoint(a,b,c), segPoint(a,b,d),
        segPoint(c,d,a), segPoint(c,d,b)});
}
double areaTriangle(pt a, pt b, pt c) {return abs(cross(b-a, c-a)) / 2.0;}
double areaPolygon(vector<pt> p) {
    double area = 0.0;
    for (int i = 0, n = p.size(); i < n; i++) {
        area += cross(p[i], p[(i+1)%n]); // wrap back to 0 if i == n-1
    }
    return abs(area) / 2.0;
}
// true if P at least as high as A (blue part)
bool above(pt a, pt p) {return p.y >= a.y;}

// check if [PQ] crosses ray from A
bool crossesRay(pt a, pt p, pt q) {return (above(a,q) - above(a,p)) * orient(a,p,q) > 0;}

// if strict, returns false when A is on the boundary
bool inPolygon(vector<pt> p, pt a, bool strict = true) {
    int numCrossings = 0;
    for (int i = 0, n = p.size(); i < n; i++) {
        if (onSegment(p[i], p[(i+1)%n], a))
            return !strict;
        numCrossings += crossesRay(a, p[i], p[(i+1)%n]);
    }
    return numCrossings & 1; // inside if odd number of crossings
}
```

```
pt circumCenter(pt a, pt b, pt c) {
   b = b-a, c = c-a; // consider coordinates relative to A
   assert(cross(b,c) != 0); // no circumcircle if A,B,C aligned
   return a + perp(b*sq(c) - c*sq(b))/cross(b,c)/2;
}

int circleLine(pt o, double r, line l, pair<pt,pt> &out) {
   double h2 = r*r - l.sqDist(o);
   if (h2 >= 0) { // the line touches the circle
      pt p = l.proj(o); // point P
      pt h = l.v*sqrt(h2)/abs(l.v); // vector parallel to l, of length h
      out = {p-h, p+h};
   }
   return 1 + sgn(h2);
}

int circleCircle(pt o1, double r1, pt o2, double r2, pair<pt,pt> &out) {
   pt d=o2-o1; double d2=sq(d);
   if (d2 == 0) {assert(r1 != r2); return 0;} // concentric circles
   double pd = (d2 + r1*r1 - r2*r2)/2; // = |O_1P| * d
   double h2 = r1*r1 - pd*pd/d2; // = hˆ2
   if (h2 >= 0) {
      pt p = o1 + d*pd/d2, h = perp(d)*sqrt(h2/d2);
      out = {p-h, p+h};
   }
   return 1 + sgn(h2);
}

int tangents(pt o1, double r1, pt o2, double r2, bool inner, vector<pair<pt,pt>> &out) {
   if (inner) r2 = -r2;
   pt d = o2-o1;
   double dr = r1-r2, d2 = sq(d), h2 = d2-dr*dr;
   if (d2 == 0 || h2 < 0) {assert(h2 != 0); return 0;}
   for (double sign : {-1,1}) {
      pt v = (d*dr + perp(d)*sqrt(h2)*sign)/d2;
      out.push_back({o1 + v*r1, o2 + v*r2});
   }
   return 1 + (h2 > 0);
}

int main(){
   line a{{3,5},6};
   cout<<a.v<<" "<<a.c<<"\n";
   line b{3,5,6};
   cout<<b.v<<" "<<b.c<<"\n";
   line z{{3,5},{6,9}};
   cout<<z.v<<" "<<z.c<<"\n";
}
```