// Implementation of Andrew's monotone chain 2D convex hull algorithm.

// Asymptotic complexity: O(n log n).

// Practical performance: 0.5-1.0 seconds for n=1000000 on a 1GHz machine.

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

typedef double coord\_t; // coordinate type

typedef double coord2\_t; // must be big enough to hold 2\*max(|coordinate|)^2

struct Point {

coord\_t x, y;

Point()

{

this->x = 0.0000000f;

this->y = 0.0000000f;

}

Point(coord\_t x,coord\_t y)

{

this->x = x;

this->y = y;

}

bool operator <(const Point &p) const {

return x < p.x || (x == p.x && y < p.y);

}

};

// 2D cross product of OA and OB vectors, i.e. z-component of their 3D cross product.

// Returns a positive value, if OAB makes a counter-clockwise turn,

// negative for clockwise turn, and zero if the points are collinear.

coord2\_t cross(const Point &O, const Point &A, const Point &B)

{

return (long)(A.x - O.x) \* (B.y - O.y) - (long)(A.y - O.y) \* (B.x - O.x);

}

// Returns a list of points on the convex hull in counter-clockwise order.

// Note: the last point in the returned list is the same as the first one.

vector<Point> convex\_hull(vector<Point> P)

{

int n = P.size(), k = 0;

vector<Point> H(2\*n);

// Sort points lexicographically

sort(P.begin(), P.end());

// Build lower hull

for (int i = 0; i < n; ++i) {

while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--;

H[k++] = P[i];

}

// Build upper hull

for (int i = n-2, t = k+1; i >= 0; i--) {

while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--;

H[k++] = P[i];

}

H.resize(k);

return H;

}

int main()

{

vector<Point>in;

Point p(-3.4,50);

Point p1(33.4,51);

Point p2(30.4,15);

Point p3(31.4,45);

Point p4(3.4,55);

Point p5(-33.4,15);

Point p6(-31.4,75);

in.push\_back(p);

in.push\_back(p1);

in.push\_back(p2);

in.push\_back(p3);

in.push\_back(p4);

in.push\_back(p5);

in.push\_back(p6);

vector<Point>out = convex\_hull(in);

for(int a=0;a<out.size();a++)

{

Point pp = out[a];

cout<<pp.x<<" "<<pp.y<<endl;

}

}