Convex Hull

```
#include <cstdio>
#include <cassert>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs)
const { return make_pair(y,x) <
make_pair(rhs.y,rhs.x); }
  bool operator==(const PT &rhs)
const { return make_pair(y,x) ==
make_pair(rhs.y,rhs.x); }
};

T cross(PT p, PT q) { return
p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) { return
cross(a,b) + cross(b,c) +
cross(c,a); }

#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT
&b, const PT &c) {
  return (fabs(area2(a,b,c)) < EPS
&& (a.x-b.x)*(c.x-b.x) <= 0 &&
(a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(),
pts.end()), pts.end());
  vector<PT> up, dn;
  for (int i = 0; i < pts.size();
i++) {
    while (up.size() > 1 &&
area2(up[up.size()-2], up.back(),
pts[i]) >= 0) up.pop_back();
    while (dn.size() > 1 &&
area2(dn[dn.size()-2], dn.back(),
pts[i]) <= 0) dn.pop_back();
    up.push_back(pts[i]);
    dn.push_back(pts[i]);
  }
  pts = dn;
  for (int i = (int) up.size() - 2;
i >= 1; i--) pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT
  if (pts.size() <= 2) return;
  dn.clear();
  dn.push_back(pts[0]);
  dn.push_back(pts[1]);
  for (int i = 2; i < pts.size();
i++) {
    if (between(dn[dn.size()-2],
dn[dn.size()-1], pts[i]))
dn.pop_back();
    dn.push_back(pts[i]);
  }
  if (dn.size() >= 3 &&
between(dn.back(), dn[0], dn[1])) {
    dn[0] = dn.back();
    dn.pop_back();
  }
  pts = dn;
#endif
}
```

Find Angle

```
// giving a,b,c triangle it will
give you the the angle abc

double findAngle(PT a, PT b, PT c )

{

double A = sqrt(sqr(b.x-c.x)+sqr(b.y-c.y));

double B = sqrt(sqr(a.x-c.x)+sqr(a.y-c.y));

double C = sqrt(sqr(a.x-b.x)+sqr(a.y-b.y));



if(2*A*C == 0) return 1e9;

double ang = acos((A*A+C*C-B*B)/(2*A*C));

return (ang*180)/PI;

}
```