0x80 计算几何

0x81 三角函数

- `sin/cos/tan` **三角函数**
  - `sin(x) / cos(x) / tan(x)`
  - 参数是由弧度表示的浮点数，返回浮点数。
- `asin/acos/atan/atan2` **反三角函数**
  - `asin(y / 斜边) / acos(x / 斜边) / atan(y / x)` 参数为浮点数，返回浮点数弧度数。
  - `atan2(y, x)` 返回返回浮点数弧度数。
- **角度转换**
  - 弧度 $\rightarrow$ 角度：$d^\circ = (d \times \frac{\pi}{180})$弧度。
  - 角度 $\rightarrow$ 弧度：d弧度 $= (d \times \frac{180}{\pi})^\circ$。

0x82 面积

- **已知三角形三个顶点坐标，求三角形面积**
  - $S = (x_1y_2 - x_1y_3 + x_2y_3 - x_2y_1 + x_3y_1 - x_2y_2)$

```
double dis(PDD a, PDD b)
{
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

double area(PII p1, PII p2, PII p3)
{
    double a = dis(p1, p2);
    double b = dis(p1, p3);
    double c = dis(p2, p3);
    double p = (a + b + c) * 0.5;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}
```

0x83 点

```
#define Vector Point // 点
#define Re register int
const double eps = 1e-8, PI = acos(-1.0);

int dcmp(double a) { return a < -eps ? -1 : (a > eps ? 1 : 0); }//处理精度
double Abs(double a) { return a * dcmp(a); }//取绝对值
```

```
struct Point {
    double x, y; Point(double X = 0, double Y = 0) { x = X, y = Y; }
    void in() { cin >> x >> y; }
    void out() { cout << x << " " << y << endl; }
};
```

0x84 向量相关的操作

```
double Dot(Vector a, Vector b) { return a.x * b.x + a.y * b.y; }//【点积】
double Cro(Vector a, Vector b) { return a.x * b.y - a.y * b.x; }//【叉积】
double Len(Vector a) { return sqrt(Dot(a, a)); }//【模长】

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 b) { return Vector(a.x * b, a.y * b); }

//【点A\向量A顺时针旋转theta(弧度制)】,点：关于原点，向量：关于起点
Point turn_P(Point a, double theta) {
    double x = a.x * cos(theta) + a.y * sin(theta);
    double y = -a.x * sin(theta) + a.y * cos(theta);
    return Point(x, y);
}

//【将点A绕点B顺时针旋转theta(弧度)】
Point turn_PP(Point a, Point b, double theta) {
    double x = (a.x - b.x) * cos(theta) + (a.y - b.y) * sin(theta) + b.x;
    double y = -(a.x - b.x) * sin(theta) + (a.y - b.y) * cos(theta) + b.y;
    return Point(x, y);
}
```

0x85 点与线段

```
//【判断点P是否在线段AB上】
int pan_PL(Point p, Point a, Point b) {
    return !dcmp(Cro(p - a, b - a)) && dcmp(Dot(p - a, p - b)) <= 0;

}
//【点P到线段AB距离】
bool operator==(Point a, Point b) { return !dcmp(a.x - b.x) && !dcmp(a.y - b.y);
}//两点坐标重合则相等
double dis_PL(Point p, Point a, Point b) {
    if (a == b) return Len(p - a);//AB重合
    Vector x = p - a, y = p - b, z = b - a;
    if (dcmp(Dot(x, z)) < 0)return Len(x);//P距离A更近
    if (dcmp(Dot(y, z)) > 0)return Len(y);//P距离B更近
    return Abs(Cro(x, z) / Len(z));//面积除以底边长
}
```

0x85 点与直线

```
//【判断点P是否在直线AB上】
int pan_PL_(Point p, Point a, Point b) {
    return !dcmp(Cro(p - a, b - a));//PA,AB共线
}
//【点P到直线AB的垂足】
Point FootPoint(Point p, Point a, Point b) {
    Vector x = p - a, y = p - b, z = b - a;
    double len1 = Dot(x, z) / Len(z), len2 = -1.0 * Dot(y, z) / Len(z);//分别计算
AP,BP在AB,BA上的投影
    return a + z * (len1 / (len1 + len2));//点A加上向量AF
}
//【点P关于直线AB的对称点】
Point Symmetry_PL(Point p, Point a, Point b) {
    return p + (FootPoint(p, a, b) - p) * 2;//将PF延长一倍即可
}
```

0x86 直线与直线或者线段的相关内容

```
//【两直线AB,CD的交点】
Point cross_LL(Point a, Point b, Point c, Point d) {
    Vector x = b - a, y = d - c, z = a - c;
    return a + x * (Cro(y, z) / Cro(x, y));//点A加上向量AF
}

 //【判断直线AB与线段CD是否相交】
int pan_cross_L_L(Point a, Point b, Point c, Point d)
{
    return pan_PL(cross_LL(a, b, c, d), c, d);//直线AB与直线CD的交点在线段CD上
}

//【判断两线段AB,CD是否相交】
int pan_cross_LL(Point a, Point b, Point c, Point d) {
    double c1 = Cro(b - a, c - a), c2 = Cro(b - a, d - a);
    double d1 = Cro(d - c, a - c), d2 = Cro(d - c, b - c);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(d1) * dcmp(d2) < 0;//分别在两侧
}
```

0x87 点是否在多边形内部

```
//【射线法】判断点A是否在任意多边形Poly以内
int PIP(Point* P, Re n, Point a) {
    Re cnt = 0; double tmp;
    for (Re i = 1; i <= n; ++i) {
        Re j = i < n ? i + 1 : 1;
        if (pan_PL(a, P[i], P[j]))return 2;//点在多边形上
```

```
        if (a.y >= min(P[i].y, P[j].y) && a.y < max(P[i].y, P[j].y))//纵坐标在该线
段两端点之间
            tmp = P[i].x + (a.y - P[i].y) / (P[j].y - P[i].y) * (P[j].x - P[i].x),
cnt += dcmp(tmp - a.x) > 0;//交点在A右方
    }
    return cnt & 1;//穿过奇数次则在多边形以内
}
//【二分法】判断点A是否在凸多边形Poly以内
inline int judge(Point a, Point L, Point R) {//判断AL是否在AR右边
    return dcmp(Cro(L - a, R - a)) > 0;//必须严格以内
}
inline int PIP_(Point* P, Re n, Point a) {
    //点按逆时针给出
    if (judge(P[1], a, P[2]) || judge(P[1], P[n], a))return 0;//在P[1_2]或P[1_n]外
    if (pan_PL(a, P[1], P[2]) || pan_PL(a, P[1], P[n]))return 2;//在P[1_2]或P[1_n]
上
    Re l = 2, r = n - 1;
    while (l < r) {//二分找到一个位置pos使得P[1]_A在P[1_pos],P[1_(pos+1)]之间
        Re mid = l + r + 1 >> 1;
        if (judge(P[1], P[mid], a))l = mid;
        else r = mid - 1;
    }
    if (judge(P[l], a, P[l + 1]))return 0;//在P[pos_(pos+1)]外
    if (pan_PL(a, P[l], P[l + 1]))return 2;//在P[pos_(pos+1)]上
    return 1;
}

//【任意多边形P的面积】
inline double PolyArea(Point* P, Re n) {
    double S = 0;
    for (Re i = 1; i <= n; ++i)S += Cro(P[i], P[i < n ? i + 1 : 1]);
    return S / 2.0;
}
```

0x88 判断两个点是否共线

```
x1y2 - x2y1 == 0
如果只是判断贡献的话，建议不要用y=kx+b的公式，太难推了
(x1-x2)/(y1-y2) == (x1-x3)/(y1-y3)
可以使用上面的公式，比较简单
```