# 凸包

```
#include <math.h>
#include <stdlib.h>

#define eps     1e-8
#define zero(x) ((x) > 0 ? (x) : -(x)) < eps

struct point {
  double x, y;
};

// 叉积
double xmult(point p1, point p2, point p0) {
  return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

// graham顺时针查找包含所有共线点的凸包
point p1, p2;

int grahamCp(const void *a, const void *b) {
  double ret = xmult(*((point *)a), *((point *)b), p1);

  return zero(ret) ? (xmult(*((point *)a), *((point *)b), p2) > 0 ? 1 : -1)
                   : (ret > 0 ? 1 : -1);
}

void _graham(int n, point *p, int &s, point *ch) {
  int i, k = 0;

  for (p1 = p2 = p[0], i = 1; i < n; p2.x += p[i].x, p2.y += p[i].y, i++) {
    if (p1.y - p[i].y > eps || (zero(p1.y - p[i].y) && p1.x > p[i].x)) {
      p1 = p[k = i];
    }
  }

  p2.x /= n;
  p2.y /= n;

  p[k] = p[0];
  p[0] = p1;

  qsort(p + 1, n - 1, sizeof(point), grahamCp);

  for (ch[0] = p[0], ch[1] = p[1], ch[2] = p[2], s = i = 3; i < n;
       ch[s++] = p[i++]) {
    for (; s > 2 && xmult(ch[s - 2], p[i], ch[s - 1]) < -eps; s--)
      ;
  }
}

/**
 * @brief    构造凸包构造函数，
 * @param   n                原始点集大小
 * @param   p                点集
 * @param   convex           凸包点集合
 * @param  maxsize          =1含共线节点，=0不含共线节点
 * @param  dir              =1顺时针构造，=0逆时针构造
 * @return int
 * */
int graphm(int n, point *p, point *convex, int maxsize = 1, int dir = 1) {
  point *tmp = new point[n];
  int    s, i;
  _graham(n, p, s, tmp);

  for (convex[0] = tmp[0], n = 1, i = (dir ? 1 : (s - 1)); dir ? (i < s) : i;
       i += (dir) ? 1 : -1) {
    if (maxsize || !zero(xmult(tmp[i - 1], tmp[i], tmp[(i + 1) % s]))) {
      convex[n++] = tmp[i];
    }
  }

  delete[] tmp;
  return n;
}
```

# 网格

```
int gcd(int a, int b) {
  return b ? gcd(b, a % b) : a;
}

// 多边形上网格点数
int gridOnedge(int n, point *p) {
  int i, ret = 0;
  for (i = 0; i < n; i++) {
    ret += gcd(abs(p[i].x - p[(i + 1) % n].x), abs(p[i].y - p[(i + 1) % n].y));
  }

  return ret;
}

// 多边形内网格数
int gridInsidle(int n, point *p) {
  int i, ret = 0;

  for (i = 0; i < n; i++) {
    ret += p[(i + 1) % n].y * (p[i].x - p[(i + 1) % n].x);
  }

  return (abs(ret) - gridOnedge(n, p)) / 2 + 1;
}
```