```cpp


/*
 *              PRINCIPAIS FUNCOES DE POLIGONOS
 *
 */




/*
  considere os poligonos com os vertices em sentido anti-horario e P[0] == P[n-1]
  exceto quando especificado
*/

vector<pv> P; //poligono usado nas funçoes.

double perimetro(){
    double ans = 0.0;
    for(int i=0; i<P.size()-1; i++){
        ans += dist(P[i], P[i+1]);
    }
    return ans;
}




double area(){
    double ans=0.0;
    for(int i=0; i<P.size()-1; i++){
        ans += cross(P[i], P[i+1]);
    }
    return fabs(ans)*0.5;
}



bool is_convex(){

    if(P.size()<=3) return false;//size<=3 significa ponto ou linha. oq n forma poligono
    bool turn = ccw(P[0], P[1], P[2]);

    for(int i=1; i<P.size()-1; i++){
        if(ccw(P[i], P[i+1], P[ (i+2 == P.size()) ? 1 : i+2 ] )  != turn ) return false;
    }

    return true;

}



```

```cpp

bool inside_polygon(pv p){//funciona pra poligono concavo e convexo
    pv u, v;
    int ans = 0;


    for(int i=0, j=P.size()-1; i<P.size(); j = i++){
        u = P[i]; v = P[j];
        if(u.y > v.y) swap(u, v);//u e o mais baixo, e v o mais alto

        if(cross(p-v, u-v) == 0.0 && dot(p-v, u-v)/dot(u-v, u-v) >= 0.0   &&
           dot(p-v, u-v)/dot(u-v, u-v) <= 1.0)     return true;//trata pontos sobre arestas horizontais



        if(u.y == v.y) continue;

        if( (p.y >= u.y && p.y < v.y)  &&  cross(p-v, u-v) >= 0.0) {
            ans^=1;
        }

    }
    return ans;
}




vector<pv> cutPolygon(pv a, pv b) {//corta o poligono P pela reta ab, e retorna a parte esquerda

    vector<pv> LSIDE;


    for (int i = 0; i < (int)Q.size(); i++) {

        double left1 = cross(b-a, P[i]-a), left2 = 0;

        if (i != P.size()-1) left2 = cross(b-a, P[i+1]-a);

        if (left1 >= 0.0) LSIDE.push_back(P[i]);//ponto P[i] ta à esquerda ou na linha ab

        if (left1 * left2 < -eps) {//quando a linha ab intercepta a aresta entre P[i] e P[i+1]
            pv c;
            line_intersection(line(P[i], P[i+1]-P[i]), line(a, b-a),  c);
            LSIDE.push_back(c);
        }
    }
    if (!LSIDE.empty() && !(LSIDE.back() == LSIDE.front()))    LSIDE.push_back(LSIDE.front());//P[0] == P[n-1]

    return LSIDE;
}
```

```
114
115
116 void ConvexHull(){//substitui o vetor P pela convex hull. se nao quiser modificar
    o vetor P, basta retornar a convex hull
117
118     vector<int> d, u;
119     d.resize(n+1); u.resize(n+1);
120     int i, j=i=0;
121
122
123     for(int k=0; k<n; k++){
124
125         while(i>1 && ccw(P[u[i-2]], P[u[i-1]], P[k])) i--;
126         while(j>1 && !ccw(P[d[j-2]], P[d[j-1]], P[k])) j--;
127         u[i++] = k;
128         d[j++] = k;
129     }
130
131
132     vector<pv> CH;
133     for(int k=0; k<j; k++){
134         CH.push_back(P[d[k]]);
135     }
136     for(int k=i-2; k>=0; k--){
137         CH.push_back(P[u[k]]);
138     }
139     P = CH;
140 }
141
142
143
144
145
146 pv center_mass(){//retorna o centro de massa do poligono
147
148     double area=0.0, a;
149     pv centro;
150     for(int i=0; i<P.size()-1; i++){
151
152         a = cross(P[i], P[i+1]);
153         area+=a;
154         centro.x+= (P[i].x + P[i+1].x)*a;
155         centro.y+= (P[i].y + P[i+1].y)*a;
156     }
157     area*=0.5;
158
159     centro.x /= 6.0*area;
160     centro.y /= 6.0*area;
161     return centro;
162 }
163
```