Polígono 2D

O polígono é representado por um vetor de pontos portanto não tem estrutura.

Funções

```
using polygon = vector<point>;
const double PI = acos(-1);

// Área do polígono, com sinal
double signedArea(polygon &P){
    double result = 0.0;
    int n = P.size();
    for(int i=0; i<n; i++){
        result+=cross(P[i], P[(i+1)%n]);
    }
    return result/2.0;
}

// Área positiva do polígono
double area(polygon &P){
    return fabs(signedArea(P));
}

// Índice do ponto mais a esquerda do vetor de pontos
int leftmostIndex(vector<point>& P){
    int ans = 0;
    for(int i=1; i<int(P.size()); i++){
        if (P[i]<P[ans]) ans = i;
    }
    return ans;
}

// Retorna o polígono sem crusamento de pontos (Eu acho kkk)
polygon make_polygon(vector<point> P){
    if(signedArea(P)<0.0)
        reverse(P.begin(), P.end());

    int li = leftmostIndex(P);
    rotate(P.begin(), P.begin()+li, P.end());
    return P;
}

// Perímetro do polígono
double perimeter(polygon &P){
    double result = 0.0;
    int n = P.size();
    for(int i=0; i<n; i++) result+=dist(P[i], P[(i+1)%n]);
    return result;
}
```

```
// Verifica se polígono é convexo
bool isConvex(polygon& P){
    int n = P.size();
    if( n<3 ) return false;
    bool left = ccw(P[0], P[1], P[2]);
    for(int i=1; i<n; i++){
        if(ccw(P[i], P[(i+1)%n], P[(i+2)%n]) != left)
            return false;
    }
    return true;
}

// Verifica se um ponto está dentro do polígono
bool inPolytgon(polygon &P, point p){
    if (P.size() == 0u) return false;
    double sum = 0.0;
    int n = P.size();
    for(int i=0; i<n; i++){
        if(P[i] == p || between(P[i], p, P[(i+1)%n]))
            return true;
        if(ccw(p, P[i], P[(i+1)%n]))
            sum+=angle(P[i], p, P[(i+1)%n]);
        else
            sum-=angle(P[i], p, P[(i+1)%n]);
    }
    return fabs(fabs(sum)-2*PI) < EPS;
}

// Polígono a direita que é formado pelo corte do polígono P pela reta
formada pelos pontos a e b

polygon cutPolygon(polygon &P, point a, point b){
    vector<point> R;
    double left1, left2;
    int n = P.size();
    for(int i=0; i<n; i++){
        left1 = cross(b-a, P[i]-a);
        left2 = cross(b-a, P[(i+1)%n]-a);
        if (left1 > -EPS) R.push_back(P[i]);
        if (left1 * left2 < -EPS)
            R.push_back(lineIntersectSeg(P[i], P[(i+1)%n], a, b));
    }
    return make_polygon(R);
}
```

Convex Hull

Dado um conjunto de pontos retorna o polígono que contém todos os pontos em O(n log n). Caso presice consederar os pontos no meio de uma aresta trocar ccw para >=0. CUIDADO: Se todos os pontos forem colineares, vai dar RTE.

```
point pivot(0, 0);

bool angleCmp(point a, point b){
    if(collinear(pivot, a, b))
        return inner(pivot-a, pivot-a) < inner(pivot-b, pivot-b);
    return cross(a-pivot, b-pivot) >=0;
}

polygon convexHull(vector<point> P){
    int i, j, n = P.size();
    if( n<=2 ) return P;
    int P0 = leftmostIndex(P);
    swap(P[0], P[P0]);
    pivot = P[0];
    sort(++P.begin(), P.end(), angleCmp);
    vector<point> S;
    S.push_back(P[n-1]);
    S.push_back(P[0]);
    S.push_back(P[1]);
    for(i=2; i<n;){
        j=int(S.size()-1);
        if(ccw(S[j-1], S[j], P[i]))
            S.push_back(P[i++]);
        else S.pop_back();
    }
    reverse(S.begin(), S.end());
    S.pop_back();
    reverse(S.begin(), S.end());
    return S;
}
```

Monotone chain

- Com esse algorítimo é possível conseguir o polígono que contém todos os pontos em O(n log n) com todos os pontos consistindo como inteiros e não ponto flutuante. **Caso presice consederar os pontos no meio de uma aresta trocar ccw para >=0**
- Nessa rotina é preciso usar a struct point como **long long.**

```
using polygon = vector<point>;

polygon monotone_chain(const vector<point> points){
    vector<point> P(points);
    sort(P.begin(), P.end());
    vector<point> lower, upper;
    for(const auto& p: P){
        int n = int(lower.size());
        while(n>=2 and ccw(lower[n-2], lower[n-1], p)){
            lower.pop_back();
            n = int(lower.size());
        }
        lower.push_back(p);
    }
    reverse(P.begin(), P.end());
    for(const auto& p: P){
        int n = int(upper.size());
        while(n>=2 and ccw(upper[n-2], upper[n-1], p)){
            upper.pop_back();
            n = int(upper.size());
        }
        upper.push_back(p);
    }
    lower.pop_back();
    lower.insert(lower.end(), upper.begin(), upper.end()-1);
    return lower;
}
```