第 3 章 计 算 几 何

3.1 多 边 形

3.1.1 计算几何误差修正

【任务】

给定一个double类型的数，判断它的符号。

【说明】

因为计算几何中经常涉及精度问题，需要对一个很小的数判断正负，所以需要引入一个极小量eps。

【接口】

int cmp(double x);

输入：　x　　判断符号的数

输出：　x的符号，-1表示x为负数，1表示x为正数，0表示x为0。

【代码】

```
1   const double eps = 1e-8;
2   int cmp(double x) {
3       if (fabs(x) < eps) return 0;
4       if (x > 0) return 1;
5       return -1;
6   }
```

【注释】

在本章中，常用基本类型（如 point）和常用基本函数（如 cmp）就不额外标出外部调用了。

【使用范例】

参见程序 POJ2653.CPP。

3.1.2　计算几何点类

【任务】

设计一个二维点类，可以进行一些向量运算。

【接口】

结构体：*point*

成员变量：

double x, y　点的坐标

重载运算符：+, −, ×, /, ==

成员函数：

input()　输入一个点

norm()　计算向量的模长

相关函数：

double sqr(double x)	计算一个数的平方
double det(const point &a, const point &b)	计算两个向量的叉积
double dot(const point &a, const point &b)	计算两个向量的点积
double dist(const point &a, const point &b)	计算两个点的距离
point rotate_point(const point &p, double A)	$\overrightarrow{op}$绕原点逆时针旋转A（弧度）

【代码】

```
const double pi = acos(-1.0);
inline double sqr(double x) {
    return x * x;
}
struct point {
    double x, y;
    point() {}
    point(double a, double b): x(a), y(b) {}
    void input() {
        scanf("%lf%lf", &x, &y);
    }
    friend point operator + (const point &a, const point &b) {
```

```
13          return point(a.x + b.x, a.y + b.y);
14      }
15      friend point operator - (const point &a, const point &b) {
16          return point(a.x - b.x, a.y - b.y);
17      }
18      friend bool operator == (const point &a, const point &b) {
19          return cmp(a.x - b.x) == 0 && cmp(a.y - b.y) == 0;
20      }
21      friend point operator * (const point &a, const double &b) {
22          return point(a.x * b, a.y * b);
23      }
24      friend point operator * (const double &a, const point &b) {
25          return point(a * b.x, a * b.y);
26      }
27      friend point operator / (const point &a,const double &b) {
28          return point(a.x / b, a.y / b);
29      }
30      double norm() {
31          return sqrt(sqr(x) + sqr(y));
32      }
33  };
34  double det(const point &a, const point &b) {
35      return a.x * b.y - a.y * b.x;
36  }
37  double dot(const point &a, const point &b) {
38      return a.x * b.x + a.y * b.y;
39  }
40  double dist(const point &a, const point &b) {
41      return (a - b).norm();
42  }
43  point rotate_point(const point &p, double A) {
44      double tx = p.x, ty = p.y;
45      return point(tx * cos(A) - ty * sin(A), tx * sin(A) + ty * cos(A));
46  }
```

【使用范例】

参见程序 POJ2653.CPP。

3.1.3　计算几何线段类

【任务】

实现一个线段类，可以完成线段的一些计算几何运算。

【说明】

为了避免精度问题，且实现起来方便，线段用一个有向线段表示。线段类的运算也都使用向量运算。

在存储时，就存下线段上的两点，用$a \to b$来表示有向线段。同样也可以用这种方式来表示直线。

【接口】

结构体：*line*

成员变量：

point a, b　线段的两个端点

相关函数：

line point_make_line(const point a,const point b);

用两个点a, b生成的一个线段或者直线

double dis_point_segment(const point p,const point s,const point t);

求p点到线段st的距离

void PointProjLine(const point p, const point s, const point t, point &cp);

求p点到线段st的垂足，保存在cp中。

bool PointOnSegment (point p, point s, point t);

判断p点是否在线段st上（包括端点）

bool parallel(line a,line b);

判断a和b是否平行。

bool line_make_point(line a,line b,point &res);

判断a和b是否相交，如果相交则返回true且交点保存在res中

line move_d(line a,const double &len);

将直线a沿法向量方向平移距离len得到的直线

【代码】

```
1   struct line {
2       point a,b;
3       line() {}
```

```
    line(point x,point y): a(x),b(y) {}
};
line point_make_line(const point a,const point b) {
    return line(a,b);
}
double dis_point_segment(const point p,const point s,const point t) {
    if (cmp(dot(p-s,t-s))<0) return (p-s).norm();
    if (cmp(dot(p-t,s-t))<0) return (p-t).norm();
    return fabs(det(s-p,t-p)/dist(s,t));
}
void PointProjLine(const point p,const point s, const point t, point &cp){
    double r=dot((t-s),(p-s))/dot(t-s,t-s);
    cp=s+r*(t-s);
}
bool PointOnSegment (point p, point s, point t) {
   return cmp(det(p-s,t-s))==0 && cmp(dot(p-s,p-t))<=0;
}
bool parallel(line a,line b) {
    return !cmp(det(a.a-a.b,b.a-b.b));
}
bool line_make_point(line a,line b,point &res) {
    if (parallel(a,b)) return false;
    double s1=det(a.a-b.a,b.b-b.a);
    double s2=det(a.b-b.a,b.b-b.a);
    res=(s1*a.b-s2*a.a)/(s1-s2);
    return true;
}
line move_d(line a,const double &len) {
    point d=a.b-a.a;
    d=d/d.norm();
    d=rotate_point(d,pi/2);
    return line(a.a+d*len,a.b+d*len);
}
```

【使用范例】

参见程序 POJ2653.CPP，POJ1584.CPP。

3.1.4　多边形类

【任务】

实现一个多边形类，完成计算多边形的面积、重心等基本操作。

【说明】

判断点在多边形内：从该点做一条水平向右的射线，统计射线与多边形相交的情况，若相交次数为偶数，则说明该点在形外，否则在形内。为了便于交点在顶点或射线与某些边重合时的判断，可以将每条边看成左开右闭的线段，即若交点为左端点则不计算。

【接口】

结构体：$polygon$

成员变量：

int n	多边形点数
point a[ ]	多边形顶点坐标（按顺时针顺序）

成员函数：

double perimeter( )	计算多边形周长
double area( )	计算多边形面积
int Point_In(point t);	判断点是否在多边形内部

复杂度：$O(N)$

输　入：t　需要判断的点t

输　出：0表示t点在多边形外

1表示t点在多边形内

2表示在t点在多边形的边界上

【代码】

```
const int maxn = 100;
struct polygon {
    int n;
    point a[maxn];
    polygon(){}
    double perimeter() {
        double sum=0;
        a[n]=a[0];
        for (int i=0;i<n;i++) sum+=(a[i+1]-a[i]).norm();
        return sum;
```

```
    }
    double area() {
        double sum=0;
        a[n]=a[0];
        for (int i=0;i<n;i++) sum+=det(a[i+1],a[i]);
        return sum/2.;
    }
    int Point_In(point t) {
        int num=0,i,d1,d2,k;
        a[n]=a[0];
        for (i=0;i<n;i++){
            if (PointOnSegment(t,a[i],a[i+1])) return 2;
            k=cmp(det(a[i+1]-a[i],t-a[i]));
            d1=cmp(a[i].y-t.y);
            d2=cmp(a[i+1].y-t.y);
            if (k>0 && d1<=0 && d2>0) num++;
            if (k<0 && d2<=0 && d1>0) num--;
        }
        return num!=0;
    }
};
```

【使用范例】

参见程序 POJ1584.CPP，ZOJ1081.CPP。

3.1.5 多边形的重心

【任务】

多边形类中的成员函数，求多边形的重心。

【说明】

将多边形分割为三角形的并，对每个三角形求重心（三角形重心即为三点坐标的平均值），然后以三角形的有向面积为权值求加权平均即可。

【接口】

point polygon::MassCenter();

复杂度：O(N)

输　出：多边形的重心坐标

【代码】

```
point polygon::MassCenter() {
    point ans=point(0,0);
    if (cmp(area())==0) return ans;
    a[n]=a[0];
    for (int i=0;i<n;i++) ans=ans+(a[i]+a[i+1])*det(a[i+1],a[i]);
    return ans/area()/6.;
}
```

【注释】

当多边形面积为0时重心没有定义，需要特别处理。

【使用范例】

参见程序 POJ1385.CPP。

3.1.6　多边形内格点数

【任务】

给出多边形的顶点（整点），求多边形内以及多边形边界上格点的个数。

【说明】

Pick公式：

给定顶点坐标均是整点的简单多边形，有：

$$\text{面积}=\text{内部格点数目}+\text{边上格点数目}/2-1$$

边界上的格点数：

把每条边当做左开右闭的区间以避免重复，一条左开右闭的线段$(x1,y1)\rightarrow(x2,y2)$上的格点数为：$\gcd(x2-x1,y2-y1)$。

【接口】

polygon::Border_Int_Point_Num( );

输入：a　　全局变量，表示多边形顶点坐标

输出：多边形边界上的格点个数

polygon::Inside_Int_Point_Num( );

输入：a　　全局变量，表示多边形顶点坐标

输出：多边形内的格点个数

【代码】

```
int polygon::Border_Int_Point_Num() {
    int num=0;
    a[n]=a[0];
    for (int i=0;i<n;i++)
        num+=gcd(abs(int(a[i+1].x-a[i].x)),abs(int(a[i+1].y
        -a[i].y)));
    return num;
}
int polygon::Inside_Int_Point_Num(){
    return int(area())+1-Border_Int_Point_Num()/2;
}
```

【使用范例】

参见程序 POJ1265.CPP。

3.1.7 凸多边形类

【任务】

实现一个凸多边形类，可以求出凸包、判断点是否在凸包内。

【说明】

为了避免精度问题，求凸包采用的是水平序的求法。

判断点是否在凸包内实现了一个复杂度O(n)的和一个复杂度O(log$n$)的。

【接口】

polygon_convex convex_hull(vector<point> a);

复杂度：O($n$log$n$)

输　入：a　所有的点

输　出：用a中的点求出的凸包（逆时针顺序）

bool containOn(const polygon_convex &a,const point &b);

复杂度：O(n)

输　入：&a　一个凸包

　　&b　一个点

输　出：点b是否在凸包a中，true表示点在凸包内部或者在边界上

int containOlogn(const polygon_convex &a,const point &b);

复杂度：O(log$n$)

输　入：&a　一个凸包

　　&b　一个点

输　出：点b是否在凸包a中，true表示点在凸包内部或者在边界上

【代码】

```
struct polygon_convex {
    vector <point> P;
    polygon_convex(int Size=0) {
        P.resize(Size);
    }
};

bool comp_less(const point &a,const point &b) {
    return cmp(a.x-b.x)<0 || cmp(a.x-b.x)==0 && cmp(a.y-b.y)<0;
}
polygon_convex convex_hull(vector<point> a) {
    polygon_convex res(2*a.size()+5);
    sort(a.begin(),a.end(),comp_less);
    a.erase(unique(a.begin(),a.end()),a.end());
    int m=0;
    for (int i=0;i<a.size();++i){
        while (m>1 && cmp(det(res.P[m-1]-res.P[m-2],a[i]-res.P[m-2]))
        <=0) --m;
        res.P[m++]=a[i];
    }
    int k=m;
    for (int i=int(a.size())-2;i>=0;--i) {
        while (m>k && cmp(det(res.P[m-1]-res.P[m-2],a[i]-res.P[m-2]))
        <=0) --m;
        res.P[m++]=a[i];
    }
    res.P.resize(m);
    if (a.size()>1) res.P.resize(m-1);
    return res;
}

bool containOn(const polygon_convex &a,const point &b) {
    int n=a.P.size();
```