#include <iostream>

using namespace std;

#define INF 10000

struct Point

{

double x;

double y;

};

bool onSegment(Point p, Point q, Point r)

{

if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&

q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))

return true;

return false;

}

int orientation(Point p, Point q, Point r)

{

int val = (q.y - p.y) \* (r.x - q.x) -

(q.x - p.x) \* (r.y - q.y);

if (val == 0) return 0; // colinear

return (val > 0)? 1: 2; // clock or counterclock wise

}

bool doIntersect(Point p1, Point q1, Point p2, Point q2)

{

// Find the four orientations needed for general and

// special cases

int o1 = orientation(p1, q1, p2);

int o2 = orientation(p1, q1, q2);

int o3 = orientation(p2, q2, p1);

int o4 = orientation(p2, q2, q1);

if (o1 != o2 && o3 != o4)

return true;

if (o1 == 0 && onSegment(p1, p2, q1)) return true;

if (o2 == 0 && onSegment(p1, q2, q1)) return true;

if (o3 == 0 && onSegment(p2, p1, q2)) return true;

if (o4 == 0 && onSegment(p2, q1, q2)) return true;

return false; // Doesn't fall in any of the above cases

}

bool isInside(Point polygon[], int n, Point p)

{

if (n < 3) return false;

Point extreme = {INF, p.y};

int count = 0, i = 0;

do

{

int next = (i+1)%n;

if (doIntersect(polygon[i], polygon[next], p, extreme))

{

if (orientation(polygon[i], p, polygon[next]) == 0)

return onSegment(polygon[i], p, polygon[next]);

count++;

}

i = next;

} while (i != 0);

return count&1; // Same as (count%2 == 1)

}

int main()

{

Point polygon1[] = {{1, 0}, {8, 3}, {8, 8}, {1, 5}};

int n = sizeof(polygon1)/sizeof(polygon1[0]);

Point p = {3, 5};

isInside(polygon1, n, p)? cout << "True \n": cout << "False \n";

Point polygon2[] = {{-3, 2}, {-2, -0.8}, {0, 1.2}, {2.2,0},{2,4.5}};

p = {0, 0};

n = sizeof(polygon2)/sizeof(polygon2[0]);

isInside(polygon2, n, p)? cout << "True \n": cout << "False \n";

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\*\*\*\*\*\*

**$g++ -o main \*.cpp**  
**$main**  
True

False