```cpp
/******   point on ploygon *******/
#include <iostream>
#include <string>
#include <vector>
#include <iterator>
#include <algorithm>
#include <cstdio>
using namespace std;

// NB: max_int would cause overflow in the orientation test
const int INF = 1000002;

struct Point {
  int x, y;
};

ostream& operator<<(ostream& os, const Point& p) {
  os << "{" << p.x << "," << p.y << "}";
  return os;
}
ostream& operator<<(ostream& os, const vector<Point>& p) {
  os << "{";
  copy(p.begin(), p.end(), ostream_iterator<Point>(os));
  os << "}";
  return os;
}

int orientation(const Point& p1, const Point& p2, const Point& q1) {
  int val = (p2.y - p1.y) * (q1.x - p2.x) - (q1.y - p2.y) * (p2.x - p1.x);
  if (val == 0)
    return 0;
  else
    return (val < 0) ? -1 : 1;
}

// Returns true if q lies on p1-p2
bool onSegment(const Point& p1, const Point& p2, const Point& q) {
  if (min(p1.x, p2.x) <= q.x && q.x <= max(p1.x, p2.x)
    && min(p1.y, p2.y) <= q.y && q.y <= max(p1.y, p2.y))
    return true;
  else
    return false;
}

bool intersectionTest(const Point& p1, const Point& p2,
  const Point& p3, const Point& p4) {
  int o1 = orientation(p1, p2, p3);
  int o2 = orientation(p1, p2, p4);
  int o3 = orientation(p3, p4, p1);
  int o4 = orientation(p3, p4, p2);

  // General case
  if (o1 != o2 && o3 != o4)
    return true;

  // Special cases
  if (o1 == 0 && onSegment(p1, p2, p3))
```

```
    return true;
  if (o2 == 0 && onSegment(p1, p2, p4))
    return true;
  if (o3 == 0 && onSegment(p3, p4, p1))
    return true;
  if (o4 == 0 && onSegment(p3, p4, p2))
    return true;

  return false;
}

bool pointInPolygon(const Point& p, const vector<Poiht>& polygon) {

  if (polygon.size() < 3)
    return false; // Flawed polygon

  Point PtoInfinity = { INF , p.y };

  int intersectionsCount = 0;
  int i = 0, j = i + 1;
  do {

    if (intersectionTest(p, PtoInfinity, polygon[i], polygon[j]) == true) {

      ++intersectionsCount;

      if (orientation(polygon[i], polygon[j], p) == 0) { // Collinear
        if (onSegment(polygon[i], polygon[j], p) == true)
          return true;
        else {
          // Exception case when point is collinear but not on segment
          // e.g.
          //                *  ************
          //                 /             \
          //                k               w
          // The collinear segment is worth 0 if k and w have the same
          // vertical direction

          int k = (((i - 1) >= 0) ? // Negative wraparound
            (i - 1) %
static_cast<int>(polygon.size()) :
            static_cast<int>(polygon.size()) + (i - 1));
          int w = ((j + 1) % polygon.size());

          if ((polygon[k].y <= polygon[i].y && polygon[w].y <= polygon[j].y)
            || (polygon[k].y >= polygon[i].y && polygon[w].y >= polygon[j].y))
            --intersectionsCount;
        }
      }

      i = (++i % polygon.size());
      j = (++j % polygon.size());

    } while (i != 0);

```

```
  return (intersectionsCount % 2 != 0);
}

int main() {


  int a,b,c,d,e,i,j,p,q;
  bool v;
  Point z;

  vector<Point> polygon;
  cin>>a;

  for(i=1;i<=a;i++)
  {   cout<<"Case "<<i<<": "<<endl;
      polygon.clear();
      scanf("%d",&b);
      for(j=0;j<b;j++)
      {   scanf("%d %d",&z.x,&z.y);
          polygon.push_back(z);
      }
      scanf("%d",&c);
      for(j=0;j<c;j++)
      {
          scanf("%d %d",&z.x,&z.y);
          v=pointInPolygon(z,polygon);
          if(v==true)
            cout<<"Yes"<<endl;
          else
            cout<<"No"<<endl;
      }
  }



  return 0;
}
```