Temă pentru acasă. Structuri

Vom considera în continuare figuri geometrice plane înţelese ca mulţimi de puncte. De exemplu, triunghiul ΔABC este format din cele trei vârfuri A, B, şi C care îl determină, la care se adaugă toate punctele de pe cele trei laturi şi toate punctele interioare triunghiului.

Pentru a clasifica poziţiile relative a două figuri, definim enumerarea:

```
enum PozRelativa{suntDisjuncte, seAting, seSuprapun}
```

Două figuri geometrice *se suprapun* dacă au puncte interioare comune, *se ating* dacă au puncte comune dar nici unul dintre acestea nu este interior ambelor figuri şi *sunt disjuncte* dacă nu au nici un punct comun.

I. Definiţi structura **Punct** care determină un punct prin coordonatele sale **x** şi **y**, şi structura **Segment** care determină un segment prin **Punct**-ele **A** şi **B**, capetele sale. Capetele unui segment pot să coincidă, caz în care spunem că avem un segment degenerat.

Implementaţi următoarele funcţii

1. `Punct initPunct(double x, double y)` – iniţializează un punct;
2. `void scrie(Punct P)` – afişează pe monitor coordonatele lui **P**;
3. `double dist(Punct P, Punct Q)` – calculează distanţa dintre **P** şi **Q**;
4. `Segment initSegment(Punct A, Punct B)` – iniţializează un segment;
5. `Segment initSegment(double xA, double yA, double xB, double yB)` – iniţializează un segment dat prin coordonatele capetelor.
6. `void scrie(Segment s)` – afişează pe monitor coordonatele capetelor segmentului **s**;
7. `bool esteDegenerat(Segment s)` – decide dacă segmentul **s** este degenerat. Atenţie: compararea cu zero a unui număr real α se efecuează cu un test de forma $|\alpha| < \varepsilon$ cu $\varepsilon \simeq 10^{-12}$;
8. `Punct mijloc(Segment s)` – returnează mijlocul segmentului **s**;
9. `Punct raport(Segment s, double lambda)` – returnează punctul **P** care împarte segmentul **s** în raportul λ (mai precis, P este singurul punct de pe dreapta AB pentru care $\overrightarrow{AP} = \lambda\overrightarrow{AB}$;

10. `bool contine(Segment s, Punct P)` – decide dacă punctul **P** se află pe segmentul **s**. *Indicaţie*: punctul $P(x_P, y_P)$ se află pe dreapta determinată de $A(x_A, y_A)$ şi $B(x_B, y_B)$, cu $A \neq B$, dacă şi numai dacă

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_P & y_P & 1 \end{vmatrix} = 0.$$

    Ordinea punctelor de pe o dreapă coincide cu ordinea proiecţiilor lor pe oricare dintre axele de coordonate.

11. `bool inter(Segment s, Segment t, Punct & Q)` – returnează ***false*** dacă unul dintre segmentele **s** sau **t** este degenerat sau dacă segmentele sunt paralele, altfel returnează ***true*** şi încarcă în variabila **Q**, primită prin referinţă, coordonatele punctului de intersecţie a dreptelor suport ale celor două segmente.

12. `PozRelativa pozRelativa(Segment s, Segment t)` – stabileşte poziţia relativă a segmentelor **s** şi **t** după cum ele au zero, unu sau mai multe puncte în comun (punctele interioare unui segment nedegenerat sunt cele situate între capetele segmentului pe dreapta determinată de acestea);

II. Definiţi structura **Triunghi** care determină un triunghi prin **Punct**-ele **A**, **B** şi **C**, vârfurile sale. Un triunghi este degenerat dacă are vârfuri care coincid sau dacă vârfurile sunt coliniare.

Implementaţi următoarele funcţii

1. `Triunghi initTriunghi(Punct A, Punct B, Punct C)` – iniţializează un triunghi;

2. `Triunghi initTriunghi(double xA, double yA, double xB, double yB, double xC, double yC)` – iniţializează un triunghi dat prin coordonatele vârfurilor;

3. `void scrie(Triunghi t)` – afişează coordonatele vârfurilor triunghiului **t**.

4. `double aria(Triunghi t)` – calculează aria triunghiului **t**. *Indicaţie*: aria triunghiului ΔABC poate fi calculată cu formula lui Heron

$$\mathcal{A}_{\Delta ABC} = \sqrt{p(p-a)(p-b)(p-c)},$$

    unde $p = (a+b+c)/2$ este semiperimetrul triunghiului, sau cu formula

$$\mathcal{A}_{\Delta ABC} = \frac{1}{2}|\det \begin{pmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_P & y_P & 1 \end{pmatrix}|.$$

5. `bool esteDegenerat(Triunghi t)` – decide dacă triunghiul **t** este degenerat;

6. `Triunghi mijloc(Triunghi t)` – returnează triunghiul determinat de mijloacele laturilor AB, BC şi CA ale triunghiului **t**, în această ordine;

7. `void transleaza(Triunghi & t, Punct P)` – translează triunghiul **t** primit prin referinţă astfel încât punctul **t.A** să ajungă în **P**;

8. `bool contine(Triunghi t, Punct P)` – decide dacă **P** se află pe una din laturile triunghiului **t** sau în interiorul acestuia.

9. `PozRelativa pozRelativa(Triunghi s, Triunghi t)` – stabileşte poziţia relativă a triunghiurilor **s** şi **t**.

III. Definiţi structura **Cerc** care determină un cerc prin **Punct**-ul **Q**, centrul cercului, şi raza sa **r**. Un cerc este degenerat dacă are raza negativă sau nulă.

Implementaţi următoarele funcţii:

1. `Cerc initCerc(Punct Q, double r)` – iniţializează un cerc;

2. `Cerc initCerc(double xQ, double yQ, double r)` – iniţializează un cerc dat de coordonatele centrului şi raza sa;

3. `void scrie(Cerc c)` – afişează raza şi coordonatele centrului cercului **c**.

4. `double aria(Cerc c)` – calculează aria cercului **c**.

5. `PozRelativa pozRelativa(Cerc c1, Cerc c2)` – stabileşte poziţia relativă a cercurilor **c1** şi **c2**.

6. `PozRelativa pozRelativa(Cerc c, Triunghi t)` – stabileşte poziţia relativă a figurilor **c** şi **t**.

7. `double ariaComuna(Cerc c1, Cerc c2)` – calculează aria intresecţiei cercurilor **c1** şi **c2**, dacă acestea se intersectează, altfel returnează 0.

8. `Cerc cercCircumscris(Triunghi t)` – returnează cercul circumscris triunghiului **t** (dacă **t** este degenerat, returnează un cerc degenerat);

9. `Cerc cercInscris(Triunghi t)` – returnează cercul înscris în triunghiul **t** (dacă **t** este degenerat, returnează un cerc degenerat);

Exemplu de rezolvare:

```
#include<iostream>
#include <math.h>
using namespace std;

enum PozRelativa{suntDisjuncte, seAting, seSuprapun};

struct Punct{
     double x;
     double y;
};
struct Segment{
     Punct A;
     Punct B;
};
```

```
Punct initPunct(double x, double y){
    Punct P={x,y};
    return P;
}
double dist(Punct P, Punct Q){
    double a=P.x-Q.x, b=P.y-Q.y;
    return sqrt(a*a+b*b);
}
Segment initSegment(Punct A, Punct B){
    Segment s={A,B};
    return s;
}
bool esteDegenerat(Segment s){
    return dist(s.A,s.B)<1.0e-12;
}
Punct mijloc(Segment s){
    return initPunct((s.A.x+s.B.x)/2.0,(s.A.y+s.B.y)/2.0);
}

bool contine(Segment s, Punct P){
    Punct A=s.A, B=s.B;
    double det=0;
    det+=A.x*B.y+B.x*P.y+P.x*A.y;
    det-=B.x*A.y+P.x*B.y+A.x*P.y;
    if(abs(det)>=1.0e-12) return false;
    //A, B si P sunt coliniare
    if(A.x<=P.x && P.x<=B.x) return true;
    if(A.x>=P.x && P.x>=B.x) return true;
    return false;
}
int main(){
    Segment s=initSegment(initPunct(1,2),initPunct(7,3));
    if(contine(s,mijloc(s))) cout<<"contine"<<endl;
    else cout<<"NU contine"<<endl;
    return 0;
}
```