```
//Definir el tad intervaloR, representando un intervalo en R
//con dos atributos de tipo real, representando sus extremos.
//Definir las siguientes funciones y operadores:
//longitud, devuelve la longitud de un intervalo.
// < devuelve true si el 1er op está incluido
//en el segundo
// / devuelve true si los operadores son disjuntos
// * devuelve la intersección
// = asignacion
// << inserción
#include<iostream>
using namespace std;
class intervaloR{
  float a,b;
public:
  intervaloR(void);
  intervaloR(float x,float y);
  intervaloR(const intervaloR &i);
  //devuelve la longitud del intervalo
  float longitud(void) const;
  //devuelve true si el 1ero contenido en el 2ndo
  bool operator<(const intervaloR &i) const;
  //devuelve true si el 1ero y el 2ndo son disjuntos
  bool operator/(const intervaloR &i) const;
  //devuelve la interseccion (0,0) es el vacio
  intervaloR operator*(const intervaloR &i) const;
  void operator=(const intervaloR &i);
  friend ostream &  operator<<(ostream &os, const intervaloR &i);
};

//constructor por defecto
intervaloR::intervaloR(void)
{
  a=0.0;
  b=0.0;
};

//constructor desde dos reales
//asigna el mas pequeño al primer atributo
intervaloR::intervaloR(float x,float y)
{
  if (x<y)
    {
      a=x;
      b=y;
    }
  else
    {
      a=y;
      b=x;
    }
}

//constructor de copia
intervaloR::intervaloR(const intervaloR &i)
{
  a=i.a;
  b=i.b;
}

//devuelve la lingitud de un intervalo
float intervaloR::longitud(void) const
{
//por como es el constructor, siempre b>a
  return b-a;
}
//sobrecarga de < para representar la inclusion
bool intervaloR::operator<(const intervaloR &i) const
{
  //i.a a b i.b
  if (a>=i.a&&a<=i.b&&b>=i.a&&b<=i.b)
    return true;
  return false;
}

//devuelve true si dos intervalos tienen interseccion vacia
bool intervaloR::operator/(const intervaloR &i) const
{
  //a b i.a i.b || i.a i.b a b
  if ((a<=i.a&&b<=i.a)||(a>=i.b&&b>=i.b))
    return true;
  return false;
}

//interseccion de intervalos
intervaloR intervaloR::operator*(const intervaloR &i) const
{
  intervaloR r;
  //i.a a b i.b
  if (a>=i.a&&a<=i.b&&b>=i.a&&b<=i.b)
    {
      r=intervaloR(a,b);
      return r;
    }
  //a i.a i.b b
  if (i.a>=a&&i.a<=b&&i.b>=a&&i.b<=b)
    {
      r=intervaloR(i.a,i.b);
      return r;
    }
  //a b i.a i.b || i.a i.b a b
  if ((a<i.a&&b<i.a)||(a>i.b&&b>i.b))
    {
      r=intervaloR(0.0,0.0);
      return r;
    }
  //i.a a i.b b
  if ((a>=i.a&&a<=i.b)&&b>=i.b)
    {
      r=intervaloR(a,i.b);
      return r;
    }
  //a i.a b i.b
  if (a<=i.a&&(b>=i.a&&b<=i.b))
    {
      r=intervaloR(i.a,b);
      return r;
    }
}

//asignacion
void intervaloR::operator=(const intervaloR &i)
{
  a=i.a;
  b=i.b;
}

//insercion
ostream &  operator<<(ostream &os, const intervaloR &i)
{
  os<<i.a<<' '<<i.b;
  return os;
}
```

```
//Definir el tad intervaloR2, con dos atributos de tipo
//intervaloR, con las siguientes funciones y operadores:
// area, devuelve el area de un intervaloR2
// < devuelve true si el op izq esta incluido en el dcho
// * devuelve la intersección de dos intervaloR2
// = asignación
// << inserción
class intervaloR2{
  intervaloR x,y;
public:
  intervaloR2(void);
  intervaloR2(const intervaloR2 &a);
  intervaloR2(intervaloR a, intervaloR b);
  intervaloR2(float x1, float x2, float y1, float y2);
  float area(void) const;
  //devuelve true si el 1ero esta contenido en el 2ndo
  bool operator<(const intervaloR2 &i) const;
  //devuelve la interseccion (0,0,0,0) es el vacio
  intervaloR2 operator*(const intervaloR2 &i) const;
  void operator=(const intervaloR2 &i);
  friend ostream & operator<<(ostream &os,const intervaloR2 &i);
};

//constructor por defecto, llama a los constructores de x e y (inicializadores
intervaloR2::intervaloR2(void):x(),y(){}
intervaloR2::intervaloR2(intervaloR a, intervaloR b)
{
  x=a;
  y=b;
}

//constructor desde 4 reales, llama a los constructores de x e y (inicializadore
s
intervaloR2::intervaloR2(float x1, float x2,
                         float y1, float y2):x(x1,x2),y(y1,y2)
{
}
//sin inicializadores podria hacerse asi
//intervaloR2::intervaloR2(float x1, float x2,
//                         float y1, float y2)
//{
//  x=intervaloR(x1,x2);
//  y=intervaloR(y1,y2);
//}

//constructor de copia
intervaloR2::intervaloR2(const intervaloR2 &a)
{
  x=a.x;
  y=a.y;
}

//devuelve el area de un intervaloR2
//se calcula como producto de la longitud de
//los intervalos x e y
float intervaloR2::area(void) const
{
//llama a intervaloR::longitud
  return x.longitud()*y.longitud();
}

//sobrecarga de < para representar la inclusion
bool intervaloR2::operator<(const intervaloR2 &a) const
{
  //si los intervaloR que engendran el intervaloR2 a la izqd  de <
  //estan contenidos en los intervaloR que engendran el intervaloR2
  //a la derecha  de <devuelve true
  if (x<a.x&&y<a.y)
    return true;
  //en caso contrario devuelve false
  return false;
}

//sobrecarga de * para representar la interseccion
intervaloR2 intervaloR2::operator*(const intervaloR2 &a) const
{
  intervaloR2 r;
  //si alguno es disjunto devuelve el vacio
  if (x/a.x||y/a.y)
    {
      r=intervaloR2(0,0,0,0);
      return r;
    }
  //si no devuelve la intersecion que se obtiene
  //como el intervaloR2 engendrado por las intersecciones
  //de los respectivos intervaloR
  r=intervaloR2(x*a.x,y*a.y);
  return r;
}

//asignacion
void intervaloR2::operator=(const intervaloR2 &i)
{
  x=i.x;
  y=i.y;
}

//insercion
ostream & operator<<(ostream &os, const intervaloR2 &i)
{
  os<<i.x<<' '<<i.y;
  return os;
}

int main()
{
  intervaloR x(1,5),y(2,4),z;
  if (y<x) cout<<"y dentro de x"<<endl;
  z=x*y;
  cout<<x<<'*'<<y<<'='<<z<<endl;
  y=intervaloR(2,6);
  z=x*y;
  cout<<x<<'*'<<y<<'='<<z<<endl;
  y=intervaloR(6,10);
  z=x*y;
  cout<<x<<'*'<<y<<'='<<z<<endl;
  y=intervaloR(0,0.5);
  z=x*y;
  cout<<x<<'*'<<y<<'='<<z<<endl;
  intervaloR2 a(intervaloR(1,10),intervaloR(5,20)),
    b(intervaloR(2,3),intervaloR(6,7)),c;
  c=a*b;
  cout<<a<<'*'<<b<<'='<<c<<endl;
  cout<<a.area()<<endl;
  if (b<a) cout<<"b dentro de a"<<endl;
  else cout<<"b NO dentro de a"<<endl;
  b=intervaloR2(1,10,5,20);
  a=intervaloR2(2,3,6,7);
  c=a*b;
  cout<<a<<'*'<<b<<'='<<c<<endl;
  cout<<a.area()<<endl;
  if (b<a) cout<<"b dentro de a"<<endl;
  else cout<<"b NO dentro de a"<<endl;
  b=intervaloR2(5,10,2,20);
```

```
  a=intervaloR2(2,3,6,10);
  c=a*b;
  cout<<a<<'*'<<b<<'='<<c<<endl;
  cout<<a.area()<<endl;
  if (b<a) cout<<"b dentro de a"<<endl;
  else cout<<"b NO dentro de a"<<endl;
  b=intervaloR2(5,10,2,20);
  a=intervaloR2(2,7,6,10);
  c=a*b;
  cout<<a<<'*'<<b<<'='<<c<<endl;
  cout<<a.area()<<endl;
  if (b<a) cout<<"b dentro de a"<<endl;
  else cout<<"b NO dentro de a"<<endl;
}
// ejemplo de ejecucion
// salida por pantalla:
// y dentro de x
// 1 5*2 4=2 4
// 1 5*2 6=2 5
// 1 5*6 10=0 0
// 1 5*0 0.5=0 0
// 1 10 5 20*2 3 6 7=2 3 6 7
// 135
// b dentro de a
// 2 3 6 7*1 10 5 20=2 3 6 7
// 1
// b NO dentro de a
// 2 3 6 10*5 10 2 20=0 0 0 0
// 4
// b NO dentro de a
// 2 7 6 10*5 10 2 20=5 7 6 10
// 20
// b NO dentro de a


//Definir el tad punto2d, con dos atributos de tipo punto2d y
//la función miembro distancia, que devuelve la distancia entre
//dos puntos.
#include<iostream>
using namespace std;
#include<cmath>

class punto2d {
  float x,y;
public:
  punto2d();
  punto2d(float, float);
  float distancia(const punto2d&) const;
};

//constructor por defecto
punto2d::punto2d()
{
  x=0;
  y=0;
}

//constructor desde dos reales
punto2d::punto2d(float a, float b)
{
  x=a;
  y=b;
}

//distancia entre dos puntos
float punto2d::distancia(const punto2d &b) const
{
  return sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y));
}

//Definir el tad segmento, con dos atributos de tipo punto2d.
//Definir la función miembro longitud, devuelve la longitud
//de un segmento.
//Definir el operador < de modo que devuelve true si el
//segmento de la izqda es menor que el de la dcha.
class segmento {
  punto2d a,b;
public:
  segmento();
  segmento(float, float, float, float);
  segmento(punto2d,punto2d);
  float longitud(void) const;
  bool operator<(const segmento &) const;
};

//constructor por defecto, llamadas a inicializadores
segmento::segmento():a(),b()
{
}

//constructor desde 4 reales, llamadas a inicializadores
segmento::segmento(float r, float s, float t, float u):a(r,s),b(t,u)
{
}

//constructor desde dos punto2d
segmento::segmento(punto2d e1,punto2d e2)
{
  a=e1;
  b=e2;
}

//longitud
float segmento::longitud(void) const
{
  //distancia entre los atributos
  return a.distancia(b);
}

//sobrecarga de < para comparar dos segmentos segun
//su longitud
bool segmento::operator<(const segmento &s) const
{
  return longitud()<s.longitud();
  //alternativamente
  //return a.distancia(b)<s.longitud();
}

int main()
{
  segmento s1(2,2,3,3),s2(punto2d(10,10),punto2d(5,5));
  cout<<s1.longitud()<<' '<<s2.longitud()<<' ';
  if (s1<s2) cout<<"menor"<<endl;
  else cout<<"mayor";
}
```

```
//Definir el tad circa, circunferencia, con dos atributos, un punto2d
//y un float representando el centro y el radio respectivamente.
//Definir el operador < de modo que devuelva true si la circa de la izqda
//esta contenida en la de la derecha.
//Definir la función tangente, que devuelve la tangente a una circa dada
//conocido el centro de la tgt (si hay varias soluciones, escoja la que desee).
#include<iostream>
using namespace std;
#include<cmath>
//El tad punto2d ya visto...
class punto2d {
  float x,y;
public:
  punto2d(){x=0;y=0;}
  punto2d(float a, float b){x=a;y=b;}
  float distancia(const punto2d &b) const {
    return sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y));
  }
};

class circa{
  punto2d c;
  float r;
public:
  //constructor por defecto, inicializadores
  circa():c(){
    r=0;
  }

  //constructor desde un punto2d y un float
  circa(punto2d centro, float radio){
    c=centro;
    r=radio;
  }

  //constructor desde tres float, inicializadores
  circa(float uno, float dos, float tres):c(uno,dos){
    r=tres;
  }

  //sobrecarga de < para representar la inclusion
  //está contenida si la distancia entre centros
  //mas el radio de la circa de la izqd es menor que el
  //radio de la de la dcha
  bool operator<(const circa &e) const {
    return c.distancia(e.c)+r<e.r;
  }

  //circunferencia tangente a una dada conociendo el centro
  //dos casos, tangente externa e interna
  circa tangente(const punto2d &p) const {
    //si la distancia del punto al centro
    //de la circa es mayor que el radio
    //tgt externa
    if (p.distancia(c)>r)
      //la tangente de centro p tiene de radio
      //la distancia de p a c menos el radio de
      //la otra circa
      return circa(p,p.distancia(c)-r);
    //en caso contrario, interna
    else
      //la tgt de centro p tiene de radio
      //el radio de la otra menos la distancia
      //de p a c
      return circa(p,r-p.distancia(c));
  }
};

int main()
{
  circa a(10,10,10),b(punto2d(5,5),2),c;
  punto2d p(1,1);
  if (b<a) cout<<"b menor que a"<<endl;
  c=a.tangente(p);
}
```