```
// testB.C

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

class Point
{
public :
        Point ()                                                { x = 0; y = 0; }
        Point (float x0, float y0)              { x = x0; y = y0; }
        float getx () const                             { return x; }
        float gety () const                             { return y; }
        float operator- (const Point& p2) const;
        bool operator== (const Point& p2) const;

private :
        float x, y;
};

float Point::operator- (const Point& p2) const
{
        float dx = x - p2.x;
        float dy = y - p2.y;
        return sqrt (dx * dx + dy * dy);
}

bool Point::operator== (const Point& p2) const
{
    return x == p2.x && y == p2.y;
}

const int ROUGE = 1;
const int VERT = 2;
const int BLEU = 3;

class Forme
{
public :
        Forme (int coul)                                { couleur = coul; }
        int getCouleur () const                 { return couleur; }
        void setCouleur (int coul)              { couleur = coul; }
        virtual void afficher () const = 0;
        virtual float perimetre () const = 0;
        virtual float surface () const = 0;
        virtual bool pointDedans (const Point& p) const = 0;

private :
        int couleur;
};

// constructeurs de Rectangle et Cercle : dans la liste d'initialisation, appel
// des constructeurs des objets membres (BG ou centre) et du constructeur de la
// classe "mère" Forme

class Rectangle : public Forme
{
public :
        Rectangle (const Point& pointBG, float larg, float haut);
        Rectangle (const Point& pointBG, const Point& pointHD);
        void afficher () const;
        float perimetre () const;
    float surface () const;
        bool pointDedans (const Point& p) const;

private :
        Point BG;
        float largeur, hauteur;
};

// pour BG, appel du constructeur par copie de Point
```

```
Rectangle::Rectangle (const Point& pointBG, float larg, float haut)
        : Forme(BLEU), BG(pointBG)
{
        largeur = larg;
        hauteur = haut;
}

Rectangle::Rectangle (const Point& pointBG, const Point& pointHD)
        : Forme(BLEU), BG(pointBG)
{
        largeur = pointHD.getx() - pointBG.getx();
        hauteur = pointHD.gety() - pointBG.gety();
}

void Rectangle::afficher () const
{
        cout << "Rectangle BG=(" << BG.getx() << "," << BG.gety() << ")"
                << ", largeur=" << largeur << ", hauteur=" << hauteur
                << ", couleur=" << getCouleur() << endl;
}

float Rectangle::perimetre () const
{
    return 2 * (largeur + hauteur);
}

float Rectangle::surface () const
{
    return largeur * hauteur;
}

bool Rectangle::pointDedans (const Point& p) const
{
        return  p.getx() > BG.getx()  &&  p.getx() < BG.getx() + largeur
                &&  p.gety() > BG.gety()  &&  p.gety() < BG.gety() + hauteur;
}

const float PI = 3.14f;

class Cercle : public Forme
{
public :
        Cercle (const Point& cen, float ray);
        Cercle (const Point& p1, const Point& p2);
        void afficher () const;
        float perimetre () const;
    float surface () const;
        bool pointDedans (const Point& p) const;

private :
        Point centre;
        float rayon;
};

// pour centre, appel du constructeur par copie de Point
Cercle::Cercle (const Point& cen, float ray)
        : Forme(BLEU), centre(cen)
{
        rayon = ray;
}

// le centre est le milieu de P1P2; pour l'objet centre, appel du constructeur
// Point(float,float)
Cercle::Cercle (const Point& p1, const Point& p2)
        : Forme(BLEU), centre( (p1.getx()+p2.getx())/2, (p1.gety()+p2.gety())/2 )
{
        rayon = (p1 - p2) / 2;      // le diamètre est la distance P1P2
}

void Cercle::afficher () const
{
        cout << "Cercle centre=(" << centre.getx() << "," << centre.gety() << ")"
```

```
                << ", rayon=" << rayon
                << ", couleur=" << getCouleur() << endl;
}
float Cercle::perimetre () const
{
    return 2 * PI * rayon;
}

float Cercle::surface () const
{
    return PI * rayon* rayon;
}

bool Cercle::pointDedans (const Point& p) const
{
    // le point est à l'intérieur si sa distance au centre est < au rayon
        return (p - centre) < rayon;
}

class ListeFormes
{
public :
        ListeFormes ();
        void ajouter (Forme* pForme);
        void afficher ();
        void changerCouleur (const Point& p, int coul);

private :
        Forme* formes [100];    // liste de pointeurs Forme* pointant sur des
                                // objets Rectangle ou des objets Cercle
        int nb;                 // nombre d'objets Forme ajoutés
};

ListeFormes::ListeFormes ()
{
        nb = 0;
}

// en paramètre, l'adresse de l'objet Forme (Rectangle ou Cercle) à ajouter
void ListeFormes::ajouter (Forme* pForme)
{
        formes[nb] = pForme;
        nb++;
}


void ListeFormes::afficher ()
{
    // la fonction afficher étant virtuelle, c'est la fonction afficher de
    // Rectangle qui est appelée si on pointe sur un Rectangle, celle de Cercle
    // si on pointe sur un Cercle
        for (int i = 0; i < nb; i++)
                formes[i]->afficher();
}

void ListeFormes::changerCouleur (const Point& p, int coul)
{
        for (int i = 0; i < nb; i++)
        {
        // pour la fonction pointDedans, même remarque que pour la fonction afficher
                if (formes[i]->pointDedans(p))
                        formes[i]->setCouleur (coul);
        }
}

int main ()
{
        // test Point
        cout << "TEST POINT" << endl;
        Point P0, P1(2,3), P2(5,7), P3(2,3);
        cout << "P0=" << P0.getx() << "," << P0.gety()
                << ", P1=" << P1.getx() << "," << P1.gety()
```

```
                << ", P2=" << P2.getx() << "," << P2.gety()
                << ", distance P1P2 = " << P1 - P2 << endl;
        if (P1 == P3 && !(P1 == P2))
                cout << "operateur == ok" << endl;

        // test Rectangle, Cercle, Forme
        cout << "\nTEST RECTANGLE, CERCLE, FORME" << endl;
        Point A(4,5), B(2,4), C(6,9), D(6.5,8);

        Rectangle rect1 (A, 3, 4);
        Rectangle rect2 (B, C);
        rect1.afficher();
        cout << "\tperimetre=" << rect1.perimetre()
                << ",surface=" << rect1.surface() << endl;
        rect2.afficher();
        if (rect1.pointDedans(D) && !rect2.pointDedans(D))
                cout << "\npointDedans rectangle ok" << endl;

        cout << endl;
        Cercle cer1 (A, 3);
        Cercle cer2 (B, C);
        cer1.afficher();
        cout << "\tperimetre=" << cer1.perimetre()
                << ",surface=" << cer1.surface() << endl;
        cer2.afficher();
        if (cer2.pointDedans(D) && !cer1.pointDedans(D))
                cout << "\npointDedans cercle ok" << endl;

        // test ListeFormes
        cout << "\nTEST LISTE_FORMES" << endl;
        ListeFormes liste;
        liste.ajouter (&rect1);
        liste.ajouter (&cer1);
        liste.ajouter (&rect2);
        liste.ajouter (&cer2);
        cout << "**** liste ****" << endl;
        liste.afficher();

        liste.changerCouleur (D, ROUGE);
        cout << "\n**** liste apres changement couleur ****" << endl;
        liste.afficher();

        return 0;
}
```