Classe Point et collection de Point

1- Soit la classe Point :

```
class Point
{
        double x ;
        double y ;
        public :
        // Q1 : le constructeur avec et sans arguments, il permet de fait : Point P(5,8) ; Point Q ; le point Q est initialisé à (0,0). Le constructeur sans arguments est absolument nécessaire car il est appelé lors de la création de tableau d'objets (de point).
        point(int a=0, int b=0) ;

        // Q2 : Permet de calculer la distance entre deux points : d=P.distance(Q).
        double distance(const Point & P) ;

        // Q3 : Permet d'afficher un point : P.afficher() : (4,5) par exemple
        void afficher() ;
};
```

Écrire les fonctions de la classe Point

2- Soit la classe col_points qui permet de gérér une collection de points.

```
class col_points
{
        Point *T ; // Tableau dynamique de points, alloué dans le constructeur
        int nbe ;  // le nombre de points dans la collection à un moment donné
        int cap ; // la taille du tableau T (donné au constructeur)
        public :

        // Q4 : un constructeur avec un argument qui initialise la collection à vide, alloue le tableau T. La valeur par défaut de cap est 100 : si cap n'est pas donnée, alors elle prend 100 par défaut
        col_points(int cap=100) ;
```

```
// Q5 : le constructeur par recopie : nécessaire dans le cas de retour par valeur
col_points(const col_points & C) ;

// Q6 : La fonction d'affichage: doit appeler la fonction d'affichage de la classe point
void afficher()

// Q7 : l'opérateur d'affectation : libère la partie dynamique de (*this) ; crée la partie dynamique de (*this) avec la bonne taille ; transfère les infos de C vers (*this).
col_points & operator=(const col_points & C) ;

// Q8 :le destructeur : doit libérer toute la mémoire allouée par new
~col_points() ;

// Q9 Fonction qui insère un point dans la collection (bool au cas où la collection est pleine)
bool inserer(const Point & P) ;

// Q10 Fonction qui insère un point dans la collection ; si la collection est pleine alors la fonction doit allouer un espace deux fois plus grand ; doit transférer les données vers le nouvel espace ; doit libérer l'espace d'origine et le faire remplacer par le nouveau. Et finalement doit procéder à l'insertion.
bool inserer_bis(const Point & P) ;

// Q11 : l'opérateur + :  C=H+K implique que C contient H union K
col_points operator+(const col_points & C) ;

// Q12 : l'opérateur [] permet d'accéder (en lecture et en écriture) au ième Point : P=C[i] ; C[i]=P ;
Point & operator[](int i) ;
```

```
        // Q13 : cette fonction permet de renvoyer le centre de gravité d'une collection de points : G=moyenne(collection).
        Point centre_gravite() ;

        // Q14 : cette fonction renvoie un tableau de points (alloué dynamiquement) contenant tous les points (de la collections) qui se trouvent au voisinage (de rayon R donné) d'un point donné. N'oublier de renvoyer la taille de ce tableau :taille_new_tableau .
        Point * voisinage(const Point & P, double R, int & taille_new_tableau) ;
} ;
```

Écrire les fonctions de la classe col_points

Votre main doit montrer le test de toutes les fonctions demandées.