Université Hassan II de Casablanca
ENSET de Mohammedia
Département Mathématiques et Informatique
Filière d'Ingénieurs : GLSID 1 & BDCC 1
Prof. M. K. MANSOURI

TP 3
PROGRAMMATION ORIENTEE OBJET EN C++
MEMBRES STATIQUES, PROPRIETES DES FONCTIONS MEMBRE

Exercice 1 :

1- Créer une classe ***compteur*** permettant à tout moment de connaître le nombre d'objets existants,
2- Ecrire une fonction ***essai*** dans laquelle, vous créez deux objets u et v de classe ***compteur***,
3- Ecrire la fonction ***main*** dans laquelle, cous créez un objet a, vous appelez ensuite la fonction ***essai***, puis vous créez un objet b,
4- Compiler et exécuter le programme. Conclure.

Exercice 2 :

1- Reprendre la classe ***point***, en utilisant la surdéfinition des fonctions membres pour créer plusieurs constructeur de cette classe.

- Premier constructeur : **`point();`** sans paramètres, il initialise x et y à 0,
- Premier constructeur : **`point(int);`** avec un seul paramètre, il initialise x et y à la valeur de ce paramètre,
- Premier constructeur : **`point(int, int);`** avec deux paramètres, il initialise x à la valeur du premier paramètre et y à la valeur du second paramètre,

2- Ecrire les deux fonctions membres surchargées suivante :

- Une fonction : **`affiche();`** sans paramètres, permettant d'afficher un point à l'écran,
- Une fonction : **`affiche(char *);`** à un sel paramètre, une chaîne de caractère, permettant le nom du point et d'appeler la fonction : **`affiche()ci dessus`**.

3- Proposer une fonction ***main()*** créant des objets de classe ***point*** et faisant appel à ces deux fonctions membres.

Exercice 3 :

1- Reprendre la classe ***point***, écrite dans l'exercice 2 ci-dessus en écrivant les différents constructeurs et fonctions membres sous forme de fonctions « ***inline*** ».
2- Comparer la taille du fichier objet « ***.obj*** » de cet exemple et celui de l'exemple précédent. Conclure ?

Exercice 4 :

Reprenons la classe ***point*** dans laquelle vous allez introduire une fonction membre nommée « ***coïncidence*** ». Cette fonction permet de détecter la coïncidence éventuelle entre deux points et qui a comme paramètre un objet de classe ***point***.

Exercice 5 : Passage de paramètres par adresse :

1- Modifier la fonction membre « ***coïncidence*** » de l'exemple précédent de sorte que son prototype devienne ***int point:: coïncidence (point *adpt),***
2- Ré-écrire le programme principal en conséquence.

Exercice 6 : Passage de paramètres par référence :

1- Modifier la fonction membre « ***coïncidence*** » de l'exemple précédent de sorte que son prototype devienne ***int point:: coïncidence (point &adpt),***
2- Ré-écrire le programme principal en conséquence.

Exercice 7 : Classe vecteur 1 :

Soit une classe ***vecteur*** définie de la manière suivante :

```
class vecteur
{
   float x,y;
   public: vecteur(float,float);
      void homotethie(float);
      void affiche();
};

vecteur::vecteur(float abs =0.,float ord = 0.)
{
    x=abs;
    y=ord;
}

void vecteur::homotethie(float val)
{
    x = x*val;
    y = y*val;
}

void vecteur::affiche()
{
    cout<<"x = "<<x<<"  y = "<<y<<"\n";
}
```

1- Mettre cette classe en en oeuvre dans un programme principal ***void main()***, en ajoutant une fonction membre ***float det(vecteur)*** qui retourne le déterminant des deux vecteurs (celui passé en paramètre et celui de l'objet),
2- Modifier la fonction ***déterminant*** de sorte de passer le paramètre par adresse.
3- Modifier la fonction ***déterminant*** de sorte de passer le paramètre par référence.

Exercice 8 : Classe vecteur 2 :

Reprenons la classe ***vecteur*** définie ci-dessus.

1- Modifier la fonction « ***homotéthie***» qui retourne le vecteur modifié par valeur. (prototype: **vecteur vecteur::homotethie(float val)**).
2- Modifier la fonction « ***homotéthie***» qui retourne le vecteur modifié par adresse.
3- Modifier la fonction « **homotéthie**» qui retourne le vecteur modifié par référence.