Tp 3 de C++. Master 1 Physique

1 Les classes en tant que agrégat de données

Traditionnellement, pour plus de clarté, lorsqu'on construit une classe on crée deux fichiers portant le même nom de base que la classe : l'un contenant les déclarations (d'extension `.hpp`) et l'autre les définitions (d'extension `.cpp`). Dans la suite, dans tous les TPs, on utilisera cette convention.

On se propose d'écrire deux classes :

— une classe `Point` permettant de manipuler des points de coordonnées de type `double` sur un plan.

— une classe `Vecteur` permettant de manipuler des vecteurs alloués dynamiquement

Exercise 1

1. Écrire une classe `Point` ayant comme données membres deux réels (`double`) x et y (respectivement son abscisse et son ordonnée) et disposant des méthodes (fonctions membres) permettant :
   — l'affichage d'un point,
   — le calcul de la distance entre deux points,
   On pourra s'appuyer sur le fichier `Point.hpp` suivant :

```
class Point
{
  double x; // abscisse
  double y; // ordonnée
  public:
  // les constructeurs
  Point();  // constructeur par défaut
  Point (double,double); //constructeur avec deux arguments
  Point (const Point &); // constructeur par recopie
  // les fonctions membres
  void affiche(); // affichage
};
```

2. Écrire le fichier `Point.cpp` correspondant en complétant le modèle suivant :

```
//constructeur par défaut
Point::Point()
{
```

```
}
//constructeur avec deux arguments
Point::Point(double a, double b)
{
}
//constructeur par recopie
Point::Point (const Point& Q)
{
}
```

3. Écrire un fichier `main_Point.cpp` permettant d'utiliser la classe `Point`
4. Écrire un `makefile` permettant de compiler tout le projet. On utilisera l'exemple donnée en cours.
5. Tester votre programme et analyser les résultats (adresses et valeurs).

Exercise 2

On reprend la classe `Point` de l'exercice précédent.

1. Les données membres x et y sont-elles publiques ou privées ?
2. Sont-elles accessibles directement depuis `main()` ?
3. Munissez la classe d'un jeu d'accesseurs en lecture et en écriture (*getters* et *setters*), méthodes publiques qui permettent aux non-membres de la classe d'accéder aux champs privés, par exemple, avec les

```
double  getx(); // récupère la valeur du champ x du point
void setx(double); // affecte la valeur du champ x du point
double gety(); // récupère la valeur du champ y du point
void sety(double); // affecte la valeur du champ y du point
```

4. Dans quels cas est-il plus avantageux d'utiliser les accesseurs en lecture et en écriture plutôt que d'accéder directement aux champs de l'objet ?

Exercise 3

Faites une classe `Vecteur`, qui aura comme champ un entier n et un pointeur sur un tableau de `double ptab`.

```
int n;
double* ptab;
```

On veillera à ce que ces champs soient privés.

1. Munissez-la :
   — d'un constructeur sans argument, qui affecte n à 1, crée dynamiquement le `double` associé à `ptab` (qui est un scalaire pour l'instant), et affecte l'élément à 0.0
   — d'un constructeur qui prend en arguments la taille du tableau, créer dynamiquement le tableau associé à `ptab` de dimension n, et affecte les éléments à 0.0

— d'un constructeur qui reçoit en argument la dimension de `ptab` puis un tableau permettant d'initialiser `ptab` : `Vecteur (int, double *);` Ce qui doit permettre de faire :

```
double tabl[4]={1.1,1.2,1.3,1.4};
Vecteur v3(4,tabl);
```

2. Ajoutez une méthode `affiche()` qui affiche la taille du vecteur et la valeur de ses éléments.
3. Le programme suivant ne devrait pas compiler. Pourquoi ? Justifiez votre réponse.

```
int main()
{
  double taba[4]={1.1,1.2,1.3,1.4};
  Vecteur v3(4,taba);
  for (int i=0; i<4; i++)
  cout << *(v3.ptab+i) << endl;
  return 0;
}
```

4. À présent faites en sorte que toutes les données membres soient publiques. Le programme suivant doit-il compiler correctement ?

```
int main()
{
  double taba[4]={1.1,1.2,1.3,1.4};
  {
    Vecteur v3(4,taba);
  }
  for (int i=0; i<4; i++)
  cout << *(v3.ptab+i) << endl;
  return 0;
}
```

5. Munissez la classe `Vecteur` d'une donnée membre `static` dans le but de sauvegarder l'adresse du dernier vecteur crée. La partie déclaration des données membres de la classe `Vecteur` doit ressembler à :

```
public :
 int n;
 double* ptab;
 static double* ptabSauv;
```

Ajouter la ligne `psauv = ptab;` dans les définitions de tous vos constructeurs. Assurez-vous que votre programme fonctionne correctement.

6. Initialisez `ptabSauv` juste **après** la déclaration de la classe `Vecteur` avec la commande suivante : `double* Vecteur::psauv = 0;` et essayer d'exécuter le programme suivant :

```
int main()
{
  double taba[4]={1.1,1.2,1.3,1.4};
  {
    Vecteur v3(4,taba);
  }
  for (int i=0; i<4; i++)
  cout << *(Vecteur::ptabSauv+i) << endl;
  return 0;
}
```

Commentez le résultat.