TD MOO/Langage C++
Séance 1/6

TD1 – Classe Tableau

Le but : réaliser une classe qui gère automatiquement un tableau dynamique de double.
Utilisation : encapsulation (une classe), allocation dynamique, passage par référence.
Compréhension des mécanismes par défaut du C++, operator=, constructeur copie.
Rôle et intérêt du débogage.
Utilisation de QtCreator.

Préparation : TD0 et exercice 1

Exercice 1 : La classe TableauDouble, modélisation

Dans cet exercice, il s'agit de modéliser la classe `TableauDouble` permettant la gestion des tableaux dynamiques de double.

1-a Modéliser en UML la classe `TableauDouble`. Les champs seront
- *Pt* : Un pointeur sur des ***double***
- *Taille* : le nombre de ***double*** stockés
- *Nom* : un nom pour le tableau (utilisé pour l'affichage des valeurs)

Les fonctionnalités de base de la classe seront :
- l'accès en lecture **et** en écriture (cf. question 1-d) à la taille du tableau,
- l'accès en lecture **et** en écriture à chaque élément du tableau,
- l'accès en lecture **et** en écriture au nom du tableau,
- le produit scalaire avec un autre objet `TableauDouble`
- l'affichage du contenu du tableau et de son nom,
- un constructeur utilisateur, auquel on passe la taille du tableau,
- un constructeur de copie,
- un destructeur.

1-b Quelles relations et multiplicités modélisent les liens entre la classe `TableauDouble` et les classes de ses champs ? Justifier.

1-c Quelles visibilités utiliser pour les 3 champs de `TableauDouble`?

1-d En C++, quelles sont les étapes à réaliser par la méthode d'accès en écriture au champ *Taille* ?

1-e En C++, pourquoi le passage de paramètre par copie **ne** doit **pas** être utilisé pour passer des objets de type `TableauDouble`?

1-f En C++, pourquoi faut-il écrire un destructeur dans cette classe ? Quel est l'intérêt de le faire figurer dans le diagramme de classe UML ?

Exercice 2 : Réalisation et tests

2-a Récupérer les sources fournies (squelette), créer le projet sous QtCreator et ajouter les commentaires à la fonction *TableauDouble* ::*Copie(…)*.

2-b Compléter les fonctions *GetTaille* et *SetNom* dans *TableauDouble.cpp*. Utiliser le *main* fourni pour tester vos méthodes et debugger. Ci-après un exemple d'exécution du *main* : (en gras : les valeurs rentrées par l'utilisateur)

```
Entrer le nom du tableau : Toto
Entrer la taille de Toto : 4
Entrer les 4 valeurs de Toto :
Toto[0]= 1
Toto[1]= 2
Toto[2]= 3
Toto[3]= 4
-----
Toto[4] = [1, 2, 3, 4]
```

2-c Ecrire la fonction *TableauDouble* ::*SetTaille* en vous inspirant de la fonction *TableauDouble* ::*Copie*. Tester la fonction dé-commentant le *main*.

2-d Toujours en s'inspirant de la fonction *TableauDouble ::Copie*, **créer le constructeur de copie**. Pourquoi est-il obligatoire pour la classe *TableauDouble* ?

2-e Ajouter et implémenter la fonction membre ***Plus*** qui réalise la somme termes à termes de deux tableaux de même taille. Son utilisation (dans le *main*) sera :

```
TableauDouble a(10),b(10) ;
// initialisation des valeurs des tableaux
…

TableauDouble v( a.Plus(b) ) ;
v.Affiche() ;
```

(**2-e'** Sur le même principe d'utilisation, ajouter et implémenter la fonction membre ***ProduitScalaire*** pour le calcul du produit scalaire de deux tableaux.)

2-f L'utilisation des fonctions *Plus* n'est pas très commode… On préférerait utiliser les lignes suivantes :

```
TableauDouble w(1) ;
w = a.Plus(b) ;
w.Affiche() ;
```

Exécuter pas à pas (debug) les lignes précédentes et déterminer quelles sont les méthodes exécutées par ces 3 lignes de code (ordre chronologique) ? Pourquoi ce code ne marche pas?

2-g Pour résoudre le problème précédent, on propose de surcharger l'opérateur = du C++. Son rôle sera de dupliquer un objet en mémoire. Les mécanismes du C++ imposent que cet opérateur retourne une référence sur l'objet courant. Voici sa déclaration :

```
TableauDouble & operator=(const TableauDouble &t);
```

Sa définition suit le modèle recommandé d'operator=, rappelé ci-après :

```
TableauDouble & TableauDouble ::operator=(const TableauDouble &t)
{ if( this != &t )
          { // instructions pour désallouer l'objet en cours
            // puis réallouer à la taille de 't'
            // puis copier, valeurs par valeurs les éléments de t
           }
  return (*this);
}
```

➔ Implémenter et tester votre ***operator=*** .