Université Paris Diderot — Langages à Objets Avancés
Master 1 — Année 2015-2016

TP n°5

Redéfinition d'opérateurs - Constructeurs de copie/affectation

Exercice 1 Dans ce TP nous allons essentiellement travailler sur la classe `Bilan`

1. Définissez une classe `Bilan` qui contient un champs nom et un champs score, et les deux fichiers `Bilan.h` et `Bilan.cpp`
2. Dans le constructeur et le destructeur procédez à un affichage.
3. Dans un fichier `main.cpp` définissez un objet x de la classe `Bilan` dans un `main` et vérifiez les messages de création et de destruction.

Exercice 2 [Redéfinition du flux pour l'affichage]

1. Dans votre main, constatez l'échec de l'exécution de l'instruction `cout << x << endl;`
2. En amphi vous avez déjà redéfinit l'opérateur $<<$, nous rappelons la marche à suivre :
   — Déclarez dans la partie public de `Bilan.h`
   `friend ostream& operator<<(ostream&, Bilan &);`
   (ne vous préoccupez pas de la signification du mot clé `friend`)
   — Puis dans `Bilan.cpp` écrivez le code :
   ```
   ostream& operator<<(ostream& o, Bilan &x) {
     o << '(' << x.nom << ',' << x.score << ')';
     return o;
   }
   ```
   — quel est à présent le résultat de `cout << x << endl;` ?

Exercice 3 [Constructeur de copie]

1. Dans `main.cpp` écrivez une fonction `void testParametre(Bilan b)` qui se contente d'afficher b. Testez là, combien d'appel à un destructeur avez vous vu ? Puis remplacez le type de l'argument par une reference, et testez là à nouveau.
2. Vous avez constaté que la première version crée et détruit un nouvel objet `Bilan`, et que la création de celui ci n'a pas été faite par le constructeur que nous avons nous même écrit. Ce qui s'est passé c'est que le passage des arguments s'est fait par copie, avec l'utilisation d'un contructeur de copie par défaut. Nous allons réécrire ce constructeur pour qu'il fasse ce qu'on attend de lui.
   — Dans `Bilan.h` déclarez le constructeur `Bilan(const Bilan& other);`
   — et dans `Bilan.cpp` écrivez le code attendu, en le complétant d'un affichage pour signifier que le constructeur de copie à été appelé.
   — vérifiez le comportement de `testParametre`

3. Ajoutez les déclaration suivante à votre `main()`

```
Bilan y(x);
Bilan z=x;
```

Quel est le sens de ces instructions ?

Exercice 4 [Redéfinition des opérateurs ++]

1. Constatez que `++x` n'est pas une instruction valable.
   — Déclarez `Bilan& operator++()` dans `Bilan.h`
   — Ecrivez le code correspondant dans `Bilan.cpp` qui aura pour effet d'incrémenter le score
   — Vérifier à présent l'effet de `++x`.
2. Qu'en est-il de `x++` ? Procédez comme vous l'avez vu en cours pour permettre cette opération.
3. On peut remarquer qu'il faudrait faire le même travail pour permettre les opérations `x--` et `--x` (mais ne le faites pas)
4. L'instruction `++x++;` devient acceptable dans le langage. Quel résultat obtenez vous ? Qu'est ce à dire pour les priorités de ces opérateurs ? Et que se passe t'il pour `++++x++++;` ?

Exercice 5 [Opérateur d'affectation - Corrigez un bug volontaire]

Dans le code suivant :

```
Bilan y(x);
Bilan z=x;
z=x;
if (&z!=&x) cout << "Ce sont des objets differents" << endl;
```

La première égalité fait appel au constructeur de copie, mais la seconde ne construit rien, et pour bien faire la différence avec java on peut vérifier que z et x sont bien des objets différents comme en atteste l'affichage.

Nous allons essayer de garde une trace visible de l'utilisation de l'affectation sur les objets de la classe `Bilan`.

1. Redéfinissez l'opérateur = en déclarant `Bilan& operator=(const Bilan& other);` dans `Bilan.h`, et en écrivant dans `Bilan.cpp` le code suivant inspiré des exemples du cours :

```
Bilan& Bilan::operator=(const Bilan& rhs)
{
    cout << " appel a l'operateur = sur le bilan " << rhs.nom ;
    cout << " de score " << rhs.score << endl;
    return *this;
}
```

2. On s'intéresse à une classe `Enregistrement` qui associe un Bilan à une date donnée. (Pour simplifier, la date sera exprimée en nombre de jour). Un enregistrement permettra de suivre l'évolution du score d'une personne à des étapes différentes, elle contient donc une copie d'un objet `Bilan`
   Construisez une classe `Enregistrement` avec ses deux fichiers .h et .cpp.
   — Comment déclarez vous le constructeur d'Enregistrement pour minimiser le nombre de copies de Bilan ?
   — Ecrivez une méthode `incScore()` qu incremente le score d'un enregistrement.
   — Ecrivez une méthode `incDate()` qui incremente la date d'un enregistrement
3. On s'interesse au code suivant :

```
// les deux premiers jours il ne se passe rien
 Enregistrement hier(x,0);
 Enregistrement aujourdhui(x,1);
 // puis le temps passe et on veut se rappeller des enregistrements
 // des deux derniers jours
 //  pendant 5 jours de suite le score augmente.
 for (int i=0;i<5;i++) {
     hier=aujourdhui;
     aujourdhui.incScore();
     aujourdhui.incDate();
 }
 cout << "hier : " << hier << endl;
 cout << "aujourd'hui " << aujourdhui << endl;
```

   Observez attentivement ce qu'on obtient pour l'affichage d'hier ?

   Debuggez en corrigeant l'ommission faite dans la redéfinition de l'opérateur d'affectation de la classe `Bilan`.