TD/TP 2 C++

Polytech' Paris-Sud
Et4 info
Aleksandr Setkov et Marc Evrard

12 janvier 2015

1 Partie 1 : Classe Vector

Il faut reprendre la classe **Vector** que vous avez implémentée au dernier cours. Pour cette classe il faut :

1. Implémenter : **copy constructor, copy assignment, move constructor, move assignment**

```
Vector (const Vector& x)             // copy constructor
Vector& operator= (const Vector& x) // copy assignment
Vector (Vector&& x)                  // move constructor
Vector& operator= (Vector&& x)       // move assignment
```

2. Tester ces membres dans **main()**
3. Surcharger les opérateurs :

```
// acces a un element de vector
int& operator [] (int n);
// addition
Vector operator + (const Vector& x) const;
// soustraction
Vector operator - (const Vector& x) const;
// produit scalaire
double operator * (const Vector& x) const;
// comparison
int    operator == (const Vector& x) const;
// comparison
int    operator != (const Vector& x) const;
// multiplication par -1
Vector operator - () const;
// ajouter un valeur a vector
Vector operator + (double value) const;
// soustraire un valeur a vector
Vector operator - (double value) const;
```

4. Tester les opérateurs surchargés dans **main()**

2 Partie 2 : Classe Date

1. Créer la classe **Date**

Nom	Type
year	int
month	int
day	int
today	**static** Date

2. Implémenter un constructeur, un destructeur, une méthode pour afficher la date

3. Implémenter une méthode **static** pour écrire et lire la date actuelle

4. Implémenter une méthode **getAge**(const Date& d) pour calculer le nombre d'années entre la date contenue dans l'objet et la date d

   * Pour obtenir la date actuelle on peut utiliser :

```
#include <ctime>
...
time_t now = time(0);
tm* dt = localtime(&now);
dt->tm_mday, dt->tm_mon+1, dt->tm_year+1900
```

3 Partie 3 : Héritage simple : Classes Personne, Employe, Etudiant

1. Créer la classe **Personne** avec les membres :

Nom	Type
nom	string
prenom	string
birthDate	Date

2. Implémenter un constructeur, un destructeur, les méthodes pour lire et écrire les membres

3. Implémenter une méthode **getAge()** pour calculer l'age de la personne

4. Implémenter une méthode **print()** pour afficher toutes les données

5. Créer la classe **Etudiant** qui hérite de la classe **Personne** et qui contient les membres supplémentaires :

Nom	Type
numCarteEtud	string
nomUniversite	string
nomSpecialite	string

6. Implémenter un constructeur, un destructeur, les méthodes pour lire et écrire les membres

7. Créer la classe **Employe** qui hérite de la classe **Personne** et qui contient les membres supplémentaires :

Nom	Type
salaire	float
nomEntreprise	string

8. Implémenter un constructeur, un destructeur, les méthodes pour lire et écrire les membres

9. Pour toutes les classes implémenter une méthode **print()** pour afficher toutes les données

10. Tester les classes et les méthodes dans **main()**

11. Implémenter une fonction **affiche_personne(const Personne* p)** qui va appeler la méthode **print()** de l'objet. Dans **main()** tester cette fonction avec les objets **Personne, Employe, Etudiant**

4 Partie 4 : Héritage multiniveaux : Classe Manager

1. Créer la classe **Manager** qui hérite de la classe **Employe** et qui contient les membres supplémentaires :

Nom	Type
group	Employe[GROUP_SIZE]
float	primeAnnuelle

2. Implémenter un constructeur, un destructeur, les méthodes pour lire et écrire les membres, une méthode **print()**

3. Implémenter une méthode pour ajouter les employés dans le groupe (Employe[GROUP_SIZE])

4. Définir le constructeur comme **private** et implémenter un **"named constructor"**. Cela permet de limiter la possibilité d'héritage à partir de cette classe.

```
static Manager MakeManager(...) {return Manager(...);}
```