TP 3 POO C++: Héritage

IMT Atlantique – TAF Robin/Ascii

Programmation en langage C/C++

Objectifs :
Les objectifs de ce tp sont de découvrir la notion d'héritage en C++.

1 Héritage

Exercice 1 – Un peu d'entraînement

1. Créez deux classes, `A` et `B`, avec des constructeurs par défaut qui s'annoncent eux-mêmes sur `cout`. Héritez une nouvelle classe `C` à partir de `A`, et créez un objet membre de type `B` dans `C`, mais ne créez pas de constructeur pour `C`. Créez un objet de classe `C` et observez les résultats.
2. Faites une hiérarchie de classes à trois niveaux avec des constructeurs par défaut, accompagnés de destructeurs, qui s'annoncent tous sur `cout`. Vérifiez que pour un objet de type le plus dérivé, les trois constructeurs et destructeurs sont appelés automatiquement. Expliquez l'ordre dans lequel les appels sont effectués.
3. Créez deux classes appelées `Traveler` et `Pager` sans constructeur par défaut, mais avec des constructeurs qui prennent un argument de type `string`, qu'ils copient simplement dans une variable membre `string`. Pour chaque classe, écrivez le constructeur par copie et l'opérateur d'affectation corrects. Héritez maintenant une classe `BusinessTraveler` de `Traveler` et donnez-lui un objet membre de type `Pager`. Écrivez le constructeur par défaut correct, le constructeur qui prend un argument `string`, un constructeur par recopie, et un opérateur d'affectation.

Exercice 2 – Classe Véhicule

Le but de cet exercice est d'implémenter la classe `Vehicule` et de définir deux spécialisations de cette classe via le lien d'héritage.

1. Dans un fichier `Vehicule.cc`, définissez une classe `Vehicule` qui a pour attributs (pour tout type de véhicule) : sa marque ; sa date d'achat ; son prix d'achat ; son prix courant.

2. Définissez un constructeur prenant en paramètre les trois attributs correspondant à : la marque, la date d'achat et le prix d'achat (le prix courant sera calculé plus tard).

3. Définissez une méthode publique `void affiche(ostream&) const;` qui affiche l'état de l'instance, i.e. la valeur de ses attributs.

4. Définissez ensuite deux classes `Voiture` et `Avion`, héritant de la classe `Vehicule` et ayant les attributs supplémentaires suivants :
   - pour la classe `Voiture` :
     — sa cylindrée ;
     — son nombre de portes ;
     — sa puissance ;
     — son kilométrage ;
   - pour la classe `Avion` :
     — son type (hélices ou réaction) ;
     — son nombre d'heures de vol.

5. Pour chacune de ces classes, définissez un constructeur qui permette l'initialisation explicite de l'ensemble des attributs, ainsi qu'une méthode affichant la valeur des attributs. Constructeurs et méthode d'affichage devront utiliser les méthodes appropriées de la classe parente !

6. Ajoutez ensuite à la classe `Vehicule`, une méthode `void calculePrix()` qui donne le prix courant. On calculera ce prix courant en soustrayant au prix d'achat 1% par année écoulée depuis la date d'achat.

7. Redéfinissez cette méthode dans les deux sous-classes `Voiture` et `Avion`, de sorte à calculer le prix courant en fonction de certains critères, et mettre à jour l'attribut correspondant au prix courant :
   - Pour une voiture, le prix courant est égal au prix d'achat, moins :
     — 2% pour chaque année depuis sa fabrication jusqu'à la date actuelle ;
     — 5% pour chaque tranche de 10'000 km parcourus (on arrondit à la tranche la plus proche) ;
     — 10% s'il s'agit d'un véhicule de marque « Renault » ou « Fiat » (choix totalement arbitraire qu'on est bien sûr libre de modifier) ;
     — et plus 20% s'il s'agit d'un véhicule de marque « Ferrari » ou « Porsche » (même remarque que ci-dessus).
   - Pour un avion, le prix courant est égal au prix d'achat, moins : :
     — 10% pour chaque tranche de 1000 heures de vol s'il s'agit d'un avion à réaction ;
     — 10% pour chaque tranche de 100 heures de vol s'il s'agit d'un avion à hélices.

   Le prix doit rester positif (i.e., s'il est négatif, on le met à 0).

8. Afin de tester les méthodes implémentées ci-dessus, utilisez le fichier `main_vehicule.cpp` fourni.
   Exemple de déroulement

```
– – – Voiture – – –
marque : Peugeot, date d'achat : 1998, prix d'achat : 147326,
prix actuel : 88395.5
2.5 litres, 5 portes, 180 CV, 12000 km.

– – – Voiture – – –
marque : Porsche, date d'achat : 1985, prix d'achat : 250000,
prix actuel : 48350
6.5 litres, 2 portes, 280 CV, 81320 km.

– – – Voiture – – –
marque : Fiat, date d'achat : 2001, prix d'achat : 7327.3,
prix actuel : 4433.02
1.6 litres, 3 portes, 65 CV, 3000 km.

– – – Avion a helices – – –
marque : Cessna, date d'achat : 1972, prix d'achat : 1.23067e+06,
prix actuel : 923005
250 heures de vol.

– – – Avion a reaction – – –
marque : Nain Connu, date d'achat : 1992, prix d'achat : 4.3211e+06,
prix actuel : 3.75936e+06
1300 heures de vol.
```

Exercice 3 – Vecteurs 3D

On s'intéresse ici à créer la classe représentant les vecteurs en dimension 3. Du point de vue conception, la classe `Vecteur` hérite naturellement d'une classe `Point3D` représentant un point dans l'espace par ses trois coordonnées, et possédant les méthodes suivantes :

— un constructeur, permettant d'initialiser les trois coordonnées d'un objet `Point3D` à partir de trois valeurs de type double reçus en paramètres ;
— pour la surcharge de l'opérateur de flux de sortie, permettant l'affichage des coordonnées d'un objet Point3D ;
— pour la surcharge de l'opérateur =, permettant de comparer (tester l'égalité des coordonnées) l'objet courant à un autre objet de type Point3D passé en paramètre.

1. Définissez la classe `Point3D`.
2. Implémentez ensuite la classe `Vecteur`, qui en plus de toutes les méthodes héritées, doit pouvoir effectuer les opérations suivantes :

   - addition et soustraction de deux vecteurs :
     — `Vecteur& operator+=(const Vecteur&);`
     — `Vecteur& operator-=(const Vecteur&);`
     — `const Vecteur operator+(Vecteur, const Vecteur&);`
     — `const Vecteur operator-(Vecteur, const Vecteur&);`
   - opposé d'un vecteur :
     — `const Vecteur Vecteur::operator-() const;`
   - multiplication par un scalaire :
     — `Vecteur& operator*=(double);`
     — `const Vecteur operator*(Vecteur, double);`
     — `const Vecteur operator*(double, const Vecteur&);`
   - produit scalaire de deux vecteurs :
     — `double operator*(const Vecteur&, const Vecteur&);`
   - calcul de la norme du vecteur (racine carrée du produit scalaire avec lui-même) ;

3. Testez ces opérations sur des vecteurs de votre choix, par exemple :
   - (1,2,-0.1) + (2.6, 3.5, 4.1) = (3.6, 5.5, 4)
   - (2.6, 3.5, 4.1) + (1, 2, -0.1) = (3.6, 5.5, 4)
   - (1, 2, -0.1) + (0, 0, 0) = (1, 2, -0.1)
   - (0, 0, 0) + (1, 2, -0.1) = (1, 2, -0.1)
   - (1, 2, -0.1) - (2.6, 3.5, 4.1) = (-1.6, -1.5, -4.2)
   - (2.6, 3.5, 4.1) - (2.6, 3.5, 4.1) = (0 0 0)
   - - (1, 2, -0.1) = (-1, -2, 0.1)
   - - (2.6, 3.5, 4.1) + (1, 2, -0.1) = (-1.6, -1.5, -4.2)
   - 3 * (1, 2, -0.1) = (3, 6, -0.3)
   - (1, 2, -0.1) * (2.6, 3.5, 4.1) = 9.19
   - $||(1, 2, -0.1)|| = 2.23830292855994$
   - $||(2.6, 3.5, 4.1)|| = 5.98498120297800$
4. On souhaite maintenant écrire une classe représentant des vecteurs unitaires (i.e. des vecteurs de norme 1). Cette classe hérite de la classe `Vecteur` (un vecteur unitaire « est un » vecteur).
   - Définissez la nouvelle classe, les constructeurs associés ainsi que la redéfinition de tous les opérateurs qui risquent de transformer un vecteur unitaire en un vecteur non unitaire.

   Il faut en effet garantir que les vecteurs unitaires construits restent unitaires... À ce stade deux options sont possibles lorsque le vecteur cesse d'être unitaire :
   — afficher un message d'erreur ;
   — transformer le vecteur non unitaire en un vecteur unitaire colinéaire et de même sens, en divisant simplement celui-ci par sa norme (définir une méthode `normalise`) ;