Université de Reims Champagne-Ardenne
UFR Sciences Exactes et Naturelles
Licence Informatique
INFO0402 : Programmation Orientée Objet

Devoir sur table
du 23 mars 2016

Aucun document n'est autorisé.

Questions de cours

1. Donner tous les usages possibles de `&`.
2. Donner l'ensemble des cas où un constructeur par copie est appelé.
3. Un entier peut-il toujours toujours être converti en flottant sans perte de précision. Si oui, on expliquera pourquoi. Si non, on donnera un exemple.
4. Dans les cas où cela est possible, pourquoi l'utilisation une fonction inline est préférable à une macro du précompilateur ?
5. Pourquoi les macros du précompilateur restent un outil important d'écriture d'un code en `C++` ? On donnera un exemple concret.
6. Quelle est l'utilité de `assert` ?
7. Expliquer la différence entre la notion de lien anticipé et de lien tardif.
8. Est-il strictement équivalent de passer un paramètre par pointeur ou par référence non constante ? A savoir, est-il toujours possible de substituer l'un à l'autre ? On justifiera.
9. Quelle est la différence entre une classe de base virtuelle pure et une interface virtuelle pure ?
10. Quelle est la valeur de la fonction sizeof(T) si le type `T` est défini de la manière suivante :
```
class T {
    static int  v;
    const  char c[2];
    double      *pt;
};
```
Si la réponse dépend de paramètres non précisés dans la question, on indiquera lesquels et on donnera la valeur de `sizeof(T)` dans le contexte choisi.
11. Qu'est-ce-que l'upcasting ? On donnera un exemple concret.
12. Pourquoi, lorsqu'un objet peut être polymorphique, son destructeur doit absolument être déclaré comme virtuel.

Exercice : approche objet

1. Pourquoi dit-on qu'une architecture doit être à forte cohésion et à faible couplage ?
2. Quelle est la différence entre une agrégation et une composition ?
3. Comment cette différence se traduit-elle lorsque l'on implémente une classe en `C++` ?
4. Donner le diagramme UML d'une hiérarchie polymorphique pour laquelle les objets spécialisés pourront avoir trois formes différentes. Les contraintes suivantes devront être respectées :
   - l'exemple ne devra pas être le même qui celui réalisé en TP,
   - les objets devront avoir du sens,
   - les relations suivantes devront être présentes dans le diagramme (et avoir un sens) : agrégation, composition, association simple,
   - une interface virtuelle pure devra apparaître.
   - la classe de base devra être virtuelle pure.
   - les membres (champs et méthodes) devront apparaître sur le diagramme (les limiter au minimum possible).

   Attention : une implémentation partielle de ce diagramme UML est ensuite demandée. L'exemple devra donc être le plus synthétique possible.

5. Quelles sont les deux implémentations possibles d'une composition en C++ ? On précisera les contraintes liées à la durée de vie du membre associée à chacune des implémentations.
6. Écrire en C++ la déclaration des classes issues du diagramme de la question précédente (aucune implémentation de méthode ou de fonction n'est demandée ici).
7. Pour l'une des méthodes virtuelles déclarées, donner le code de ses spécialisations.
8. Dans votre hiérarchie, écrire un bout de code qui permet de faire apparaître un même objet sous toutes ses formes polymorphiques possibles. On choisira un objet avec l'un des types qui dans votre hiérarchie permet le plus de forme possible.

Problème : conteneur

On voudrait réaliser un conteneur `Box` permettant de stocker un ensemble de personnes en fonction de leur age. Pour ceci, on dispose d'une classe `Person` dont les champs sont :

- le nom sous forme d'une chaine de caractères (un `char*`). Ce pointeur devra toujours pointer vers une zone mémoire dynamique privée (i.e. c'est le seul pointeur du programme pointant à cette adresse). Les fonctions classiques du `C` de manipulation de chaines pourront être utilisées (`strdup`, `strcmp`, `strlen`, ...).
- la date de naissance stockée sous la forme d'un entier sous la forme suivante : AAAAMMJJ où AAAA est l'année, MM le mois, et JJ le jour. On supposera qu'aucune année n'est inférieure à 1900.

Le conteneur stocke les personnes de la façon suivante :

- toutes les personnes nées la même année seront stockées dans une liste chainée (afin de simplifier l'exercice, l'insertion s'effectuera toujours en première position). Une cellule `Node` de cette liste chainée devra contenir le champ `Person` et le pointeur vers l'élément suivant de la liste.
- l'ensemble des listes chainées seront ensuite stockées dans un tableau dynamique (par exemple nommée `year`), où `year[i]` contient un pointeur vers la tête de liste chainée qui stocke les personnes nées en 1900+i.

On pose alors les questions suivantes :

1. Écrire la définition minimale de la classe `Person` avec les méthodes suivantes : le constructeur à partir d'un age et d'un nom, le constructeur par copie, destructeur, surcharge de la redirection dans un `std::ofstream` et getter (pas de setter). On interdira l'assignation par copie.
2. Écrire le code des méthodes de la classe `Person` qui n'ont pas été définies à la question précédente.
3. Écrire la définition minimale de la classe `Node` avec les méthodes suivantes : constructeur à partir d'un age, d'un nom et du pointeur sur l'élément suivant, le constructeur par copie, le destructeur et les getters. On interdira l'assignation par copie.
4. Écrire le code des méthodes de la classe `Node` qui n'ont pas été définies à la question précédente.
5. Écrire la définition minimale de la classe `Box` avec la méthodes suivantes : le constructeur par défaut, le constructeur par copie, le destructeur, l'ajout d'une personne dans le conteneur, la vérification si une personne est présente dans le conteneur à partir du nom et de son année de naissance (les codes sont demandés aux questions suivantes), l'assignation par copie, et la surcharge de l'opérateur `<<` pour un `std::ofstream`.
6. Écrire le code du constructeur par défaut de `Box`.
7. Écrire le code de la méthode d'ajout d'une personne dans un conteneur `Box`.
8. Écrire le code du destructeur de `Box`.
9. Écrire le code de la recherche d'une personne dans un conteneur `Box`.
10. Écrire le code du constructeur par copie de `Box`.
11. Écrire le code de l'assignation par copie de `Box`.
12. Écrire le code de la surcharge de l'opérateur `<<` permettant la redirection d'un objet de type `Box` dans un `std::ofstream`. On affichera les personnes par années de naissance.