

# Examen – Jeudi 21 Juin 2012

Tout document papier est autorisé. Les téléphones portables, ordinateurs, comme tout autre moyen de communication vers vos voisins ou l'extérieur sont strictement interdits. Le temps à disposition est de 3 heures. Justifier correctement vos réponses. Lire le sujet dans son intégralité avant de commencer...

## 1 Exercice

Soit le code C++ suivant :

```

1  class O {
2  private:
3      const int value;
4  public:
5      O(int v) : value(v) { value += 10; }
6      const int &getValue() { return value; }
7      friend void f();
8  };
9  void f() {
10     const O oo(56);
11     const int *pi;
12     int i = oo.value;
13
14     pi = &(oo.getValue());
15     pi = &i;
16     *pi = 34;
17     const int &k = oo.getValue();
18 }
19 int main() {
20     O o(45);
21     int i;
22
23     i = o.getValue();
24     int &j = o.getValue();
25     i = o.value;
26     f();
27     return 0;
28 }
```

Pour quelles lignes un compilateur C++ indiquerait-il des erreurs lors d'une compilation ? Pour chaque erreur expliquer clairement pourquoi c'est une erreur C++. (Note : aucune erreur de syntaxe n'est présente dans le code.)

## 2 Exercice

On sait que dans une **Université** on trouve : des **Étudiants**, des **Personnels** (des **Enseignants**, des **Chercheurs**, des **Administratifs**, des **Ingénieurs**, des **Techniciens**) qui sont affectés à des **UFR** (on s'arrête là pour simplifier).

1. Comment ces différents concepts sont-ils reliés, exprimer ces relations en UML.
2. Si l'on sait qu'il existe des *enseignants-chercheurs*, comment cela s'exprime t-il en UML ?
3. On sait qu'un étudiant possède un numéro d'étudiant (unique) et un nom, écrire en C++ les classes (attributs, code des constructeurs et d'éventuelles méthodes annexes) **Université**, **Étudiants**, de sorte qu'à l'inscription d'un étudiant, un nouveau numéro d'étudiant lui soit attribué automatiquement.
4. Écrire en C++ les classes (attributs, code des constructeurs et méthodes) **EnseignantChercheur**, **Chercheur**, **Enseignant**, **Personnel** et **UFR**, sachant que toute personne possède un nom et est affectée à une **UFR**. De plus, il doit être possible de connaître à tout instant le nombre de personnes affectées à une **UFR** donnée.

## 3 Exercice

```

class Color {
private:
    int r,g,b;
```

```

public:
    Color(int r,int g,int b) { this->r=r; this->g=g; this->b=b; }
};
int main() {
    Color a(255,0,0), b(255,0,0);
}

```

1. Compléter le code de sorte que l'on puisse comparer **a** et **b** et que ce test renvoie vrai (puisque les deux couleurs sont les mêmes).
2. Compléter le code de sorte que l'on puisse exécuter des instructions comme :

```

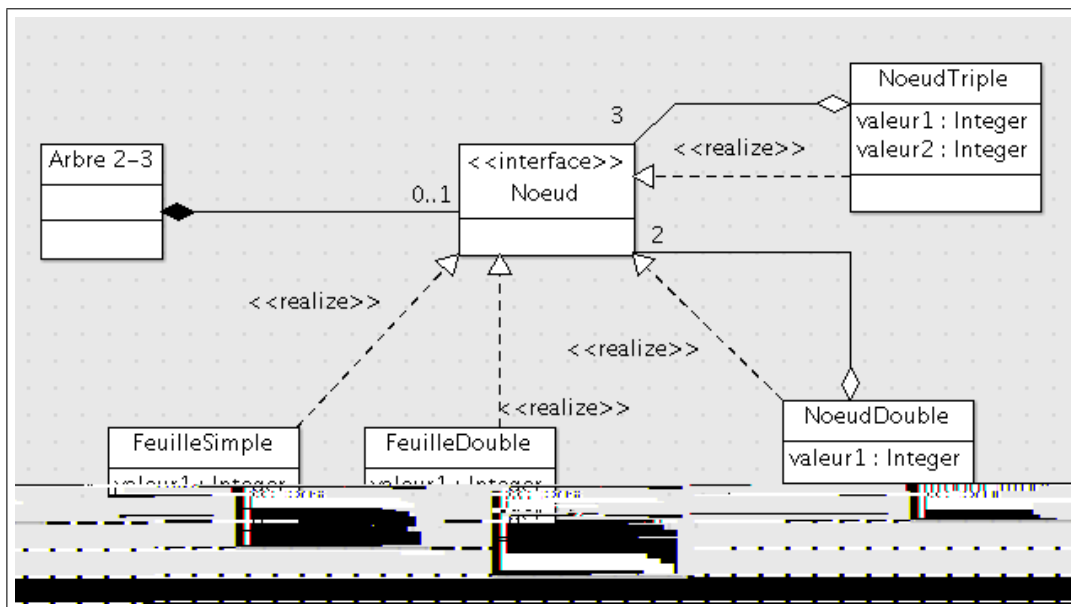
Color c(140,56,203);
cout << "Voici une couleur " << c << endl;

```

et que l'affichage produit soit **Voici une couleur [140:56:203]**.
3. Modifier le code de sorte que l'on puisse assurer qu'il n'existe jamais qu'un seul exemplaire de la couleur ROUGE (r=255,g=0,b=0). (Indice : cacher le constructeur et utiliser une usine)...
4. Modifier la méthode d'affichage de sorte que lorsqu'on affiche la couleur ROUGE, le texte affiché soit **RED**.

## 4 Exercice

Soit la définition UML suivante des arbres 2-3 (structure de données parfois utilisée) :



Le schéma ne rend pas compte de certaines propriétés liant les valeurs contenues dans les nœuds des arbres 2-3 et que l'on oubliera ici.

1. donner un exemple d'arbre 2-3 possédant au moins un nœud double et un nœud triple.
2. pour l'arbre donné en exemple à la question précédente, donner les instructions C++ qui permettraient de le construire (on ne souhaite pas ici le code des méthodes, etc; mais uniquement les instructions qui construisent les nœuds, les assemblent pour obtenir l'arbre attendu).
3. écrire la déclaration C++ de toutes les classes.
4. écrire la définition des méthodes des classes des déclarations ci-dessus.
5. on souhaite, en plus, qu'il existe une méthode de nom **incr** permettant d'augmenter de 1 toutes les valeurs contenues dans l'arbre, modifier la déclaration des classes en conséquence (ne décrire que les ajouts!)
6. écrire le code de la méthode **incr** (et de celles qu'elle utilise en annexe) sans jamais utiliser d'instruction **if** (*i.e.* le parcours de l'arbre ne doit pas utiliser d'instruction conditionnelle).
7. généraliser la définition C++ des arbres 2-3 à l'aide de templates afin d'autoriser à avoir des feuilles et nœuds de degré quelconque (on ne s'occupera que des déclarations des classes et de leurs attributs.)