ISIMA 3ème année, Examen d'Objets Avancés 2004 durée 2 heures

Questions de compréhension (1 h sans documents, 12 points)

Répondre aux questions suivantes, en les justifiant :

Concepts de base C++

1) Opérateurs : quels sont les inconvénients de la surcharge d'opérateur ?
2) Exceptions :

a. Est-ce que le mécanisme d'exception proposé en C++ est réellement utile ?

b. Pensez-vous que la classe *exception* de la STL apporte un enrichissement du mécanisme de base ?

3) Analyse d'erreur : on dispose des morceaux de code suivant

toto.h

```
#ifndef __TOTO_H__
#define __TOTO_H__

#include <iostream>
using std::ostream;

class Toto
{
  protected :
    int x;

  public :
    Toto (int i=0) : x(i) {}
    ~Toto () {}

    virtual ostream & affiche (ostream & os) const;
;
};

ostream & Toto::affiche (ostream & os) const { return (os << x);}

ostream & operator << (ostream & os, const Toto & t)
{
  return t.affiche(os);
}

#endif
```

Main.cpp

```
#include "toto.h"

int main (int, char **)
{
  Toto t1;

  std::cout << t1 << std::endl;

  return EXIT_SUCCESS;
}
```

La compilation génère l'erreur suivante

```
warning: `class Toto' has virtual functions but non-virtual destructor
```

Expliquez la faute et justifiez pourquoi ce brave **g++** a raison d'être méchant.

STL

1) Qu'est-ce qu'un foncteur ? Quelle est son utilité ? Quel est le *design pattern* sous-jacent ?
2) Pris d'une crise de paranoïa aiguë, vous ne faites plus confiance au mécanisme de tri de la STL.
    3) Proposez une implémentation d'un tri simple vu en première année (sélection, insertion, bulle…) sur une liste (*list* en STL) d'objets d'une classe A
    a. Que doit posséder la classe A ?

b. Au fait, quel est le nom de ce dangereux algorithme de la STL ?

Conception objet

1. Le *design pattern* adapteur :

a. L'adapteur permet de transformer l'interface d'une classe donnée en une interface plus conforme à nos besoins. Par exemple, on dispose d'une classe appartenant à une bibliothèque développée par une société polonaise et toutes les méthodes sont en polonais (du sud). On souhaiterait les traduire en français. Proposez deux solutions conceptuelles différentes pour y parvenir. Vous donnerez pour chacune un diagramme UML ainsi que les détails en C++ nécessaires.

b. Vos solutions permettent-elles de continuer à utiliser l'ancienne interface ? Si oui, comment faire pour l'interdire ?

2. Généricité / Objet :

    a. La généricité (template en C++) est habituellement considérée comme un mécanisme orthogonal au paradigme objet. D'après vous, quel est le concept objet fondamental qui est mis en opposition à la généricité ? Proposez un exemple C++ (pas besoin de tous les détails) qui mette en relief les deux approches.

b. En partant de cet exemple, indiquez les limites de chacune de ces deux approches.

Conception - développement (1 h avec documents, 8 points)

Documents autorisés : notes de cours et supports de cours ISIMA uniquement.

Le design pattern Mediateur

La feuille recto verso fournie en annexe décrit le *design pattern* Médiateur (*Mediator* pour les anglophiles)

1. Expliquez le design pattern en quelques lignes.
2. Proposez un schéma UML qui met en relation les deux superclasses Médiateur et Widget, dont les descendants sont la ListBox et l'EntryField pour l'une et le FontDialogDirector pour l'autre..
3. Justifiez votre schéma en expliquant son principe
4. Fournir une implémentation C++ minimale des classes correspondant à votre schéma. Pour déterminer le jeu de méthodes, vous pouvez vous aider du second schéma.

Pensez-vous qu'un Mediator pourrait aider un Ultra-Méchant Loup face à des Petits Cochons Vampires ? Ou alors doit-il se sniffer un bon coup de Mutator ? Les Power Ranger sont-ils des Visitor ou des Decorator ? Est-ce que Stroustrup va nous gronder ?

Bon stage à tous !