

Correction du partiel CMP1

EPITA – Ing1 promotion 2015 – **Sans documents ni machine**

Janvier 2013 (1h00)

Correction: Le sujet et sa correction ont été écrits par Roland Levillain.

Répondre aux questions sur les formulaires de QCM ; aucune réponse manuscrite ne sera corrigée. Ne pas oublier de renseigner les champs d'identité. Il y a exactement une et une seule réponse juste pour ces questions. Si plusieurs réponses sont valides, sélectionner la plus restrictive. Par exemple s'il est demandé si 0 est *nul*, *non nul*, *positif*, ou *négalif*, cocher *nul* qui est plus restrictif que *positif* et *négalif*, tous deux vrais. Répondre incorrectement à une question à choix multiples est plus pénalisé que de ne pas répondre. Cette correction contient en tout 4 pages.

1 C++ et programmation orientée objet

Q.1 Les mots clefs `public`, `protected` et `private` servent en C++

- à l'héritage.
- à l'instanciation.
- au masquage de données.
- à l'encapsulation.

Q.2 En C++, l'opérateur `new`

- sert à instancier sur le tas.
- sert à instancier sur la pile.
- renvoie une référence.
- est utilisé pour définir un nouveau constructeur.

Q.3 La résolution d'un appel de méthode surchargée non virtuelle ne dépend pas

- du type statique de la cible (argument « zéro »).
- des autres arguments de la méthode.
- du type de retour de la méthode.
- du fait que la cible soit `const`.

Q.4 Considérez le bout de code C++ suivant :

```
int i = 42, j = 51;
int& k = i;
k = j;
++i;
j = 237;
```

Quelles sont les valeurs de `i` et `j` et `k` à la fin de ce programme ?

- `i == 43; j == 237; k == 51;`
- `i == 43; j == 237; k == 237;`
- `i == 52; j == 237; k == 52;`
- `i == 237; j == 237; k == 237;`

Q.5 Considérons un programme C++ contenant deux classes, `shape` et `circle`, telles que la seconde dérive de la première. La classe `shape` (resp. `circle`) est déclarée dans '`shape.hh`' (resp. '`circle.hh`'), définie dans '`shape.cc`' (resp. '`circle.cc`') et compilée sous la forme du fichier '`shape.o`' (resp. '`circle.o`'). On souhaite rajouter à ce programme une classe `rectangle` héritant de `shape`, déclarée dans '`rectangle.hh`', définie dans '`rectangle.cc`' et compilée sous la forme du fichier objet '`rectangle.o`'. Parmi les affirmations suivantes, laquelle est vraie ?

- La compilation de '`rectangle.o`' nécessite la recompilation de '`circle.o`'.
- La compilation de '`rectangle.o`' nécessite la recompilation de '`shape.o`'.
- L'ajout d'une méthode à `rectangle` nécessite la recompilation de '`shape.o`'.
- L'ajout d'une méthode à `shape` nécessite la recompilation de '`rectangle.o`'.

Q.6 En reprenant les éléments de la question 5, que peut-on dire du programme ci-dessous ?

```
1 circle c(3.14, 2.72, 6.55957);
2 shape& s = c;
3 const rectangle* r = dynamic_cast<const rectangle*> (&s);
```

- Il provoque une erreur de compilation à la ligne 2.
- Il provoque une erreur de compilation à la ligne 3.
- Le pointeur '`r`' est nul après l'exécution de la ligne 3.
- Une exception `std::bad_cast` est levée lors de l'exécution de la ligne 3.

Q.7 Voici l'interface de la classe `shape` mentionnée à la question 5 :

```
class shape
{
public:
    shape(float x, float y);
    virtual ~shape();
    float x_get() const; void x_set(float x);
    float y_get() const; void y_set(float y);
protected:
    float x_, y_;
};
```

Quel type de polymorphisme est mis en œuvre dans le code ci-dessous ?

```
void translate(shape& s,
              float dx, float dy)
{
    s.x_set(s.x_get() + dx);
    s.y_set(s.y_get() + dy);
}
```

```
int main()
{
    circle c(1.f, 2.f, 3.f);
    translate(c, -5.1f, 4.2f);
}
```

- Polymorphisme d'inclusion
- Polymorphisme paramétrique
- Polymorphisme de coercion
- Polymorphisme de surcharge

Q.8 On décide d'ajouter la méthode suivante dans la déclaration de la classe `shape` montrée à la question 7 :

```
virtual std::ostream& print (std::ostream& ostr) const = 0;
```

Il s'agit d'une déclaration de méthode

- de classe.
- virtuelle.
- virtuelle pure.
- surchargée.

Q.9 Considérons le code ci-dessous, qui utilise à nouveau les types introduits à la question 5.

```
const circle c(1.f, 2.f, 3.f);
const shape& s = c;
```

Laquelle des affirmations suivantes est correcte ?

- 'c' a pour type statique 'circle' et pour type dynamique 'shape'.
- 'c' a pour type statique 'shape' et pour type dynamique 'circle'.
- 's' a pour type statique 'circle' et pour type dynamique 'shape'.
- 's' a pour type statique 'shape' et pour type dynamique 'circle'.

Q.10 Le code C++ ci-dessous :

```
namespace bar
{
    template <int a>
    double foo::f () { return cos(a * this->x_); }
}
```

définit

- une méthode de classe.
- une fonction membre virtuelle.
- un patron de fonction membre.
- un patron de fonction (autonome).

Q.11 Lorsque l'on écrit une hiérarchie de classe et que l'on utilise le transtypage ascendant, il est recommandé, pour éviter les fuites mémoire

- d'écrire des constructeurs virtuels.
- d'écrire des destructeurs virtuels.
- d'écrire des constructeurs virtuels purs.
- d'écrire des destructeurs virtuels purs.

Q.12 En C++, on appelle objet-fonction

- une méthode.
- un objet disposant d'un operator().
- un objet construit à l'intérieur d'une fonction.
- un fichier de code compilé ('foo.o') ne contenant qu'une seule fonction (foo()).

Q.13 Le code C++ ci-dessous :

```
#include <complex>
#include <vector>
#include <algorithm>

int
main()
{
    std::vector< std::complex<float> > v;
    v.push_back (std::complex<float>(1.f, 1.f));
    v.push_back (std::complex<float>(42.f, 51.f));
    v.push_back (std::complex<float>(4.3f, -2.2f));
    std::sort (v.begin(), v.end());
}
```

- compile correctement et s'exécute sans erreur.
- compile correctement mais provoque une erreur à l'exécution.
- provoque une erreur de compilation mentionnant l'absence d'un 'operator<'.
- provoque une erreur de compilation mentionnant explicitement l'inadéquation entre `std::complex<float>::iterator` et le concept `RandomAccessIterator`.

2 Compilation

- Q.14 Laquelle de ces étapes n'appartient pas à la partie frontale (*front end*) d'un compilateur ?
- ✗ Le calcul et la vérification des types
 - ✓ L'allocation de registres
 - ✗ L'analyse lexicale
 - ✗ La traduction vers une représentation intermédiaire.
- Q.15 Laquelle de ces étapes n'appartient pas à la partie terminale (*back end*) d'un compilateur ?
- ✗ La sélection d'instructions (génération de code)
 - ✗ L'analyse de la vivacité des variables
 - ✓ L'analyse et la liaison des noms
 - ✗ L'allocation de registres
- Q.16 Pour lequel de ces langages le compilateur génère des programmes sous forme de *bytecode* nécessitant l'usage d'une machine virtuelle pour leur exécution ?
- ✗ Bash (shell scripts)
 - ✗ C++
 - ✗ Haskell
 - ✓ Java
- Q.17 Bison est un
- ✗ analyseur lexical.
 - ✗ analyseur syntaxique.
 - ✗ générateur d'analyseurs lexicaux.
 - ✓ générateur d'analyseurs syntaxiques.
- Q.18 Que signifie le sigle EBNF ?
- ✓ Extended Backus-Naur Form
 - ✗ Extended Binary Nested Form
 - ✗ Enhanced Bison Normal Format
 - ✗ Elements of Bison N' Flex
- Q.19 Dans un analyseur lexical et syntaxique, que signifie le terme « valeur sémantique » ?
- ✗ Un synonyme de « token »
 - ✗ Une information de typage calculée lors de l'analyse sémantique
 - ✗ Une information de localisation dans le fichier ou le flux d'entrée
 - ✓ Une information associée à un symbole produit par le scanner ou le parser
- Q.20 Que signifie le terme '@\$' dans une action sémantique de Bison associée à la règle de production 'exp : INT' ?
- ✓ Il s'agit de la localisation du symbole de la partie gauche de la règle.
 - ✗ Il s'agit de la localisation du symbole de la partie droite de la règle.
 - ✗ Il s'agit de la valeur sémantique associée au symbole de la partie gauche de la règle.
 - ✗ Il s'agit de la valeur sémantique associée au symbole de la partie droite de la règle.