CMP1 – Construction des compilateurs

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

Janvier 2013 (1h00)

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 *négatif*, cocher *nul* qui est plus restrictif que *positif* et *négatif*, tous deux vrais. Répondre incorrectement à une question à choix multiples est plus pénalisé que de ne pas répondre. Ce sujet contient en tout 4 pages.

1 C++ et programmation orientée objet

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

a. à l'héritage.
b. à l'instanciation.
c. au masquage de données.
d. à l'encapsulation.

Q.2 En C++, l'opérateur `new`

a. sert à instancier sur le tas.
b. sert à instancier sur la pile.
c. renvoie une référence.
d. est utilisé pour définir un nouveau constructeur.

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

a. du type statique de la cible (argument « zéro »).
b. des autres arguments de la méthode.
c. du type de retour de la méthode.
d. 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 ?

a. `i == 43; j == 237; k == 51;`
b. `i == 43; j == 237; k == 237;`
c. `i == 52; j == 237; k == 52;`
d. `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 ?

a. La compilation de '`rectangle.o`' nécessite la recompilation de '`circle.o`'.
b. La compilation de '`rectangle.o`' nécessite la recompilation de '`shape.o`'.
c. L'ajout d'une méthode à `rectangle` nécessite la recompilation de '`shape.o`'.
d. 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);
```

a. Il provoque une erreur de compilation à la ligne 2.
b. Il provoque une erreur de compilation à la ligne 3.
c. Le pointeur '`r`' est nul après l'exécution de la ligne 3.
d. 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);
}
```

a. Polymorphisme d'inclusion
b. Polymorphisme paramétrique
c. Polymorphisme de coercition
d. 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

a. de classe. b. virtuelle. c. virtuelle pure. d. 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 ?

a. 'c' a pour type statique '`circle`' et pour type dynamique '`shape`'.
b. 'c' a pour type statique '`shape`' et pour type dynamique '`circle`'.
c. 's' a pour type statique '`circle`' et pour type dynamique '`shape`'.
d. '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

a. une méthode de classe.
b. une fonction membre virtuelle.
c. un patron de fonction membre.
d. 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

a. d'écrire des constructeurs virtuels.
b. d'écrire des constructeurs virtuels purs.
c. d'écrire des destructeurs virtuels.
d. d'écrire des destructeurs virtuels purs.

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

a. une méthode.
b. un objet disposant d'un `operator()`.
c. un objet construit à l'intérieur d'une fonction.
d. 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());
}
```

a. compile correctement et s'exécute sans erreur.
b. compile correctement mais provoque une erreur à l'exécution.
c. provoque une erreur de compilation mentionnant l'absence d'un '`operator<`'.
d. 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 ?

a. Le calcul et la vérification des types
b. L'allocation de registres
c. L'analyse lexicale
d. La traduction vers une représentation intermédiaire.

Q.15 Laquelle de ces étapes n'appartient pas à la partie terminale (*back end*) d'un compilateur ?

a. La sélection d'instructions (génération de code)
b. L'analyse de la vivacité des variables
c. L'analyse et la liaison des noms
d. 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 ?

a. Bash (shell scripts)
b. C++
c. Haskell
d. Java

Q.17 Bison est un

a. analyseur lexical.
b. analyseur syntaxique.
c. générateur d'analyseurs lexicaux.
d. générateur d'analyseurs syntaxiques.

Q.18 Que signifie le sigle EBNF ?

a. Extended Backus-Naur Form
b. Extended Binary Nested Form
c. Enhanced Bison Normal Format
d. Elements of Bison N' Flex

Q.19 Dans un analyseur lexical et syntaxique, que signifie le terme « valeur sémantique » ?

a. Un synonyme de « token »
b. Une information de typage calculée lors de l'analyse sémantique
c. Une information de localisation dans le fichier ou le flux d'entrée
d. 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`' ?

a. Il s'agit de la localisation du symbole de la partie gauche de la règle.
b. Il s'agit de la localisation du symbole de la partie droite de la règle.
c. Il s'agit de la valeur sémantique associée au symbole de la partie gauche de la règle.
d. Il s'agit de la valeur sémantique associée au symbole de la partie droite de la règle.