Correction du Partiel TYLA
Typologie des Langages

EPITA – Apprentis promo 2013 – **Sans documents ni machine**

Juin 2011 (1h00)

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

Barème: Se reporter à la feuille de calcul pour les coefficients des questions.

Répondre sur les formulaires de QCM; aucune réponse manuscrite ne sera corrigée. Renseigner les champs d'identité. Bien lire les questions, chaque mot est important. 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.

1 Programmation orientée objet

Q.1 Le type dynamique d'un objet

a. ✓ est un sous-type de son type statique.
b. est un sur-type de son type statique.
c. est connu à la compilation.
d. est utilisé pour distinguer des fonctions/méthodes surchargées.

Correction: Le type dynamique (ou exact) d'un objet est dérivé de son type statique (connu à la compilation). Les conversions du premier vers le dernier ne posent pas de problème (puisque le type statique est plus général que le type dynamique). Les conversions descendantes (du type statique vers le type dynamique) nécessitent en revanche des vérifications dynamiques (par exemple à l'aide `dynamic_cast` en C++ ou de `instanceof` en Java).

Q.2 Dans quel langage les appels de méthodes ne sont pas vérifiés statiquement?

a. C++ b. Java c. Simula d. ✓ Smalltalk

Correction: C++, Java et Simula disposent tous d'un système de typage à la compilation qui vérifie entre autres les signatures des appels de méthodes. Smalltalk utilise un système de liaison dynamique de méthodes qui empêche toute vérification statique.

Q.3 Qu'appelle-t-on une métaclasse en Smalltalk ?

a. Une classe abstraite.
b. Une classe qui hérite d'elle-même.
c. Une classe ayant des méta-méthodes.
d. ✓ Une classe dont les instances sont des classes.

Correction: À ne pas confondre avec les *class templates* du C++, qui sont parfois appelées métaclasses également.

Q.4 Les multiméthodes permettent

a. aux méthodes de retourner plusieurs résultats.
b. ✓ le polymorphisme dynamique sur plusieurs arguments de fonctions.
c. à une classe d'avoir des méthodes portant le même nom.
d. d'avoir des méthodes polymorphes (virtuelles) dans une hiérarchie de classe utilisant l'héritage multiple.

2 Programmation générique

Q.5 Les templates de classes du C++ sont

a. des collections de templates de fonctions libres.
b. ✓ des générateurs de classes.
c. des classes dont toutes les méthodes sont virtuelles.
d. des classes dont toutes les méthodes sont virtuelles pures.

Correction: Du fait qu'un template de classe représente un "modèle" de classe, muni de paramètres, et dont les instanciations avec diverses valeurs de paramètres effectifs produisent des types différents, on peut considérer qu'il s'agit d'un outil pour générer des classes.

Q.6 Parmi les termes suivants, lequel ne peut pas être utilisé comme paramètre effectif d'une classe paramétrée ?

a. ✓ `const`.
b. Une constante entière.
c. `unsigned`.
d. Un type classe défini par l'utilisateur.

Correction: Un paramètre de template peut être soit un nom de type, auquel cas son nom est précédé du mot-clef `typename` ou `class` dans la définition du template, soit un type numérique (comme `bool`, `int`, etc.).

Q.7 Parmi les lignes C++ suivantes, laquelle est invalide ?

a. ✓ `std::pair p1 (42, 51);`
b. `std::pair<float, int> p2 (42, 3.14f);`
c. `std::pair<int, float> p3 = std::make_pair (42, 3.14f);`
d. `std::pair<int, float> p4 = std::make_pair<int, float> (2.72f, 51);`

Correction: La première réponse est invalide, car `std::pair` ne constitue pas un type valide : il s'agit d'un template de class non instancié (c'est-à-dire, dont tous les paramètres n'ont pas été valués).

Q.8 Les concepts du C++ ISO 2003

a. se définissent grâce au mot clef `concept`.
b. sont compilés automatiquement aux sites d'utilisations.
c. sont vérifiés explicitement par le compilateur.
d. ✓ expriment des contraintes sur les paramètres de templates.

Correction: Un concept (au sens de la programmation générique) est une entité abstraite qui définit des contraintes d'ordre syntaxique et sémantique sur un (ou plusieurs) type(s). En C++ 2003 (de même que dans le prochain standard C++), les concepts n'existent pas en tant que tels dans le langage : ils sont uniquement présents dans la documentation, même s'il existe des techniques pour vérifier certaines de leurs contraintes (par exemple, la Boost Concept Checking Library).

3 Programmation fonctionnelle

Q.9 On dit d'un langage qu'il est fonctionnel

a. s'il n'effectue aucun effet de bord.
b. ✓ lorsqu'il permet de manipuler des fonctions comme n'importe quel autre entité/objet.
c. s'il supporte le concept de fonction récursive.
d. lorsqu'il dispose d'un compilateur implémenté et en état de marche.

Q.10 Un langage fonctionnel est dit pur lorsque

a. ✓ il proscrit tout effet de bord.
b. il ne contient aucune construction orientée objet.
c. ses fonctions ont au plus un argument.
d. ses expressions sont évaluées paresseusement.

Correction: On appelle effet de bord (*side effect*) une modification (affectation de variable, entrées/sorties, etc.) produite lors de l'évaluation d'une expression. Un langage sans (resp. avec) effet de bord est dit pur (resp. impur).
Pour information, un langage dont les expressions sont évaluées à la demande (*call-by-need*) est dit "paresseux" (*lazy*). C'est notamment le cas d'Haskell. Un langage qui évalue systématiquement ses expressions est dit "strict". C'est le cas d'Objective Caml, mais aussi de C++, Java, etc.

Q.11 En C++, on appelle objet-fonction

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

Q.12 On appelle fermeture

a. une fonction qui n'est pas récursive.

b. ✓ une fonction qui capture des références à des variables libres dans l'environnement lexical.

c. une fonction qui a été mise en ligne (*inlined*).

d. une fonction passée en argument à une autre fonction.

Correction: À titre d'exemple, dans le code ci-dessous, `incr` est une fermeture (*closure*) capturant une référence à la valeur 1.

```
let add x =
  let add_x y = x + y
in
  add_x
;;

let incr = add 1
in
  print_int (incr 42);
  print_newline()
;;
```

4 C++

Q.13 La liaison dynamique en C++

a. ✓ a un rapport avec `virtual`.

b. est liée à la surcharge des opérateurs.

c. repose sur `template`.

d. s'appuie sur `dynamic_cast`.

Correction: On appelle aussi cette liaison dynamique due à `virtual` "liaison retardée" (*late binding*) ou *dynamic dispatch*.

Q.14 Surcharge vs méthodes virtuelles: quelle est la bonne réponse ?

a. La surcharge et les méthodes virtuelles sont des mécanismes dynamiques.

b. La surcharge et les méthodes virtuelles sont des mécanismes statiques.

c. ✓ La surcharge est un mécanisme statique, les méthodes virtuelles un mécanisme dynamique.

d. La surcharge est un mécanisme dynamique, les méthodes virtuelles un mécanisme statique.

Q.15 Lequel de ces éléments n'entre pas en compte lors de la résolution d'une méthode surchargée en C++?

a. les arguments de l'appel.

b. le nom de la fonction.

c. le qualificatif `const` de la méthode.

d. ✓ le type de retour.

Q.16 Une fonction C++ ne peut être mise en ligne (*inlined*) si

a. ✓ elle est récursive.

b. elle fait usage de `new`.

c. elle utilise des variables globales.

d. elle renvoie une valeur (c'est-à-dire, son type de retour est différent de `void`).

Correction: On rappelle que la mise en ligne d'une fonction consiste à remplacer les appels à cette fonction par le corps de la fonction elle-même, à des fins d'optimisation. Dans le cas des fonctions récursives, la mise en ligne fait apparaître un (ou plusieurs) nouveau appels, eux-mêmes candidats à l'inlining. Or on ne peut borner à la compilation le nombre d'appels récursifs. Cette technique ne fonctionne donc pas avec les fonctions récursives.

Q.17 En C++, `std::list` est

a. un concept. b. ✓ un identifiant. c. un paramètre. d. un type.

Correction: `std::list` est un patron de classe (*class template*), et n'est pas un type valide, car il est paramétré ainsi : `std::list<T, Alloc>`, avec `T` le type des valeurs contenues dans la liste et `Alloc` le type de l'allocateur utilisé pour géré la mémoire en interne (valant par défaut `std::alloc`).

5 Langages de programmation

Q.18 Qui est l'auteur du langage C?

a. Brian Kernighan b. ✓ Dennis Ritchie c. Bjarne Stroustrup d. Ken Thompson

Q.19 Lequel de ces langages n'a pas été influencé par Simula ?

a. ✓ Algol b. C++ c. Objective C d. Smalltalk

Correction: Simula 67 est le premier langage de programmation orienté objet, et a influencé Smalltalk, C++ et Objective C. Algol a quant à lui inspiré Simula, puisque ce dernier est un sur-ensemble du premier.

Q.20 Qui est l'inventeur de la souris ?

a. ✓ Douglas Engelbart
b. Donald Knuth
c. Gordon Moore
d. Konrad Zuse

Q.21 Que signifie "BNF" ?

a. ✓ Backus-Naur Form
b. BASIC Numbering Formalism
c. Bison Normal Format
d. Bound Non-Finite (automaton)

Q.22 Classez les langages suivants par date d'apparition croissante.

a. Ada, Algol, FORTRAN, PL/I
b. ✓ FORTRAN, Algol, PL/I, Ada
c. FORTRAN, PL/I, Algol, Ada
d. PL/I, FORTRAN, Ada, Algol

Correction: Les dates d'apparition de ces langages sont :

- FORTRAN : 1957 (une première version des spécifications était disponible en 1954).
- Algol : 1958
- PL/I : 1964
- Ada : 1980

Q.23 Lequel de ces langages n'est pas normalisé ?

a. Ada b. C++ c. Fortran d. ✓ Java

Correction: Les derniers standards des langages normalisés ci-dessus sont :

- Ada 2005 (standard ISO/IEC 8652:1995/Amd 1:2007, publié en 2007)
- C++ 2003 (standard ISO/IEC 14882:2003, publié en 2003)
- Fortran 2008 (standard ISO/IEC 1539-1:2010, publié en 2010)

Quant à Java, sa dernière version est la Java Standard Edition 6 (1.6.0_26), publiée le 7 juin 2011.

6 Fonctions

Q.24 Le support des fonctions récursives nécessite

a. un tas (*heap*).

b. ✓ une pile (*stack*).

c. la liaison des fonctions dynamiques.

d. que le langage dispose de pré-déclarations (*forward declarations*).

Correction: Si un langage ne supporte par la récursion, alors on peut borner la quantité de mémoire utilisée par chaque fonction (arguments et variables locales) car chacune d'entre elles est appelée au plus une fois dans une suite d'appels de fonctions imbriqués.
Si en revanche ce langage supporte les fonctions récursives, cela implique la possibilité qu'une fonction puisse avoir plusieurs "instances" en mémoire, appelées *activations*. Pour stocker les données relatives à chacune de ces activations, ainsi que l'ordre des appels, une structure de pile est nécessaire.

Q.25 Parmi les expressions C++ ci-dessous, laquelle effectue une allocation sur le tas ?

a. `int p[42];`

b. `int p(42);`

c. ✓ `int* p = new int[42];`

d. `int* p = (int*) alloca(42);`

Correction: Parmi les réponses précédentes, seule celle qui fait usage de `new` déclenche une allocation sur le tas (dite "allocation dynamique"). Les trois autres réponses font usage de la pile : la première y crée un tableau (statique) de 42 entiers; la seconde, une variable entière initialisée à la valeur 42; quant à la dernière ligne, elle utilise la fonction `alloca` pour déplacer le pointeur de pile et y créer de la place pour un bloc mémoire de 42 octets.