Correction du partiel de Rattrapage TYLA

EPITA – Apprentis promo 2011 – **Sans documents ni machine**

Septembre 2009 (1h30)

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

1 Échauffement

1. Qu'est-ce qu'un objet fonction ?

   Correction: Un objet qui dispose d'une méthode `operator()`, i.e., qui a une méthode dont l'invocation ressemble à l'appel d'une fonction.

2. Quel(s) mot(s) clef(s) C++ servent au *masquage des données* ?

   Correction: `private` et `protected` ; et en un sens `public` aussi, qui fait partie de la même famille, même s'il permet au contraire de *montrer* les données.

3. On suppose que vous disposez du code d'une classe C++ nommée `Foo`, qui contient déjà des méthodes et des attributs. Si vous ajoutez des méthodes non virtuelles dans le code de `Foo`, est-ce que les instances (objets) de type `Foo` prendront plus de place en mémoire ? Justifiez votre réponse.

   Correction: Le fait d'ajouter des méthodes non virtuelles n'a pas d'impact sur la taille des instances. La taille d'un objet dépend de ses données (la valeur des attributs) et de la présence éventuelle d'un pointeur sur table de fonctions virtuelles, lorsque la classe comporte des méthodes polymorphes (virtuelles).

2 Portées et symboles

1. Qu'est-ce que la "portée" (*scope*) d'un identificateur ?

   Correction: La zone de programme depuis laquelle il est accessible.

2. À quoi sert un environnement/table des symboles ?

   Correction: C'est une structure de données qui permet de faire le lien entre la définition et les références (utilisations) d'un identificateur. Par exemple, lors du contrôle des types d'une déclaration de variable, c'est dans une table des symboles qu'est stockée le type de ladite variable, de façon à contrôler que les références sont conformes à ce type.

3. Quelle différence notable y a-t-il entre une table des symboles et un tableau associatif (tel que `std::map`) ?

   Correction: La gestion des portées !

4. Donnez, dans le langage de votre choix, un court programme justifiant le recourt d'un compilateur à une table des symboles plutôt qu'un tableau associatif.

Correction: La question précédente l'a souligné, la différence marquante est la gestion des portées qui autorisent l'existence simultanée, sans conflit ni confusion, de plusieurs entités portant le même nom.
Il s'agit donc de faire un programme avec deux entités (e.g., variables) portant le même nom, leur portée étant imbriquée.
En C :

```
{
 int c;    /* c1 */
 {
  char *c; /* c2 */
 }
 1 - c;    /* c1 */
}
```

En Tiger :

```
let var c := 1    /* c1 */
in
 let var c := "2" /* c2 */
 in
 end;
 1 - c          /* c1 */
end
```

Ne seront pas acceptées les réponses données dans des langages à portée dynamique : Perl (sauf avec `my`), Sh, etc.

3 Multiméthodes

1. Que sont les multiméthodes ?

Correction: Sélection à l'exécution de la bonne fonction selon le type dynamique de n'importe quel argument, et non pas seulement l'argument 0 (`this`). C'est un support à l'exécution de ce que la surcharge permet déjà de faire à la compilation.

2. Citer quelques cas où les multiméthodes offrent une facilité d'implémentation.

Correction: Un premier gain, simple, c'est de ne plus être obligé « d'être dans la classe » pour pouvoir avoir la sélection dynamique. Citons comme exemple la classique méthode `print` qui devient une fonction extérieure. En fait, les Visiteurs n'ont pas besoin des méthodes `accept` pour fonctionner. Les Visiteurs simulent les multiméthodes.
Par ailleurs, on a souvent besoin de sélection selon deux arguments : par exemple l'arithmétique entre `Numbers`, où `Number` est une classe virtuelle, ou bien encore, exemple célèbre, la recherche de l'intersection entre deux `Shapes`, où `Shape` est là encore une classe abstraite.

3. Citer un langage supportant les multiméthodes.

Correction: Common Lisp/CLOS, Dylan, Perl 6, C# 4.0 (avec le mot-clef `dynamic`).

4. En C++, à quelle technique fait-on appel pour simuler les multiméthodes ?

Correction: Le *design pattern* VISITEUR.

4 Polymorphismes

1. Qu'est-ce que le polymorphisme ? On s'attachera à donner une définition générale, et non pas liée à une forme particulière de polymorphisme, ou bien à un langage spécifique.

   Correction: C'est la possibilité d'avoir plusieurs entités (fonctions, classes) portant le même nom mais ayant des attributs différents, sans entraîner aucune ambiguïté dans le comportement du programme.
   Polymorphisme de coercition Conversion automatique de type selon le contexte d'utilisation. Exemple: en langage C, `sin(51)`, où l'entier `51` est converti en `double` avant d'être passé à `sin`.

   Polymorphisme ad hoc/de surcharge La possibilité d'avoir plusieurs fonctions ou méthodes portant le même nom, se différenciant par leur arguments (voire leur valeur de retour, dans certains langages).

   Polymorphisme d'inclusion Via les méthodes polymorphes/virtuelles.

   Polymorphisme paramétrique Les `template` du C++.

2. Soient `a` et `b` deux variables Tiger. Donner une expression Tiger dont le code effectif (c'est-à-dire celui qui sera exécuté) dépend du type de `a` et `b`.

   Correction: `a < b` donnera lieu à du code très différent selon que `a` et `b` sont des entiers, ou des chaînes de caractères (où il faut implémenter `strcmp`).
   C'est la surcharge (*overloading*).

3. Comment le compilateur détermine-t-il la version du code à générer ?

   Correction: Le contrôle de type doit avoir été effectué, de façon précisément à déterminer le type de `a` et `b` dans l'exemple précédent. En se basant sur ce typage, le générateur de code produira celui de $<_{\texttt{int}}$ ou celui de $<_{\texttt{string}}$.

4. Dans l'expression en langage C "`sin (51)` ", quel autre mécanisme de polymorphisme intervient ?

   Correction: Certains types sont automatiquement promus, comme par exemple ici, l'entier `51` sera automatiquement converti en un `double`.

5. Pourquoi le code C++ suivant ne compile pas ?

```
#include <cmath>

int main () { return std::abs (-42); }
```

   Correction: Parce que std::abs est surchargée pour plusieurs types, et qu'il n'y a pas de conversion unique de `int` vers l'un des types acceptés par ces surcharges (`double`, `float`, `long double`). Voici ce que dit `g++`:

```
abs.cc: In function 'int main()':
abs.cc:3: error: call of overloaded 'abs(int)' is ambiguous
/usr/include/c++/4.3/cmath:99: note: candidates are: double std::abs(double)
/usr/include/c++/4.3/cmath:103: note:                float std::abs(float)
/usr/include/c++/4.3/cmath:107: note:                long double std::abs(long double)
```

6. Voyez-vous en Tiger une valeur qui puisse avoir plusieurs types ?

   Correction: Il fallait bien sûr expliquer le cas de `nil`.

7. Les langages orientés objets introduisent une forme de polymorphisme. Comment l'appelleriez-vous ?

   Correction: Polymorphisme d'héritage, ou d'inclusion, voire `virtual`.

8. À quel mot-clé est-il associé en Simula et en C++?

   Correction: `virtual`.

9. Sur quelle forme de polymorphisme s'appuie largement STL ?

   Correction: Polymorphisme paramétrique, lié à la généricité. En C++, c'est lié au mot-clé `template`.