TYLA – Typologie des langages

EPITA – Promo 2012
Tous documents (notes de cours, polycopiés, livres) autorisés.
Calculatrices et ordinateurs interdits.

Juin 2010 (1h30)

Une copie synthétique, bien orthographiée, avec un affichage clair des résultats, sera toujours mieux notée qu'une autre demandant une quelconque forme d'effort de la part du correcteur. Une argumentation informelle mais convaincante, sera souvent suffisante.

Une lecture préalable du sujet est recommandée.

1 Généralités

1. Combien de nombres entiers différents peut-on représenter avec 1024 bits ?
2. En Programmation Orientée Objet (POO), qu'est-ce que l'encapsulation ?
3. En POO, qu'appelle-t-on masquage de données ?
4. Qu'appelle-t-on structure de blocs ? Citer un langage qui dispose de cette fonctionnalité.
5. Quelle différence faites-vous entre portée statique et portée dynamique ?

2 C++

1. En C++, de quels éléments dépend la sélection d'une méthode surchargée ?
2. Quelle différence faites-vous entre initialisation et affectation ? En C++, quels individus sont chargés de la première ? quels sont ceux qui sont chargés de la seconde ?
3. En C++, l'ordre d'initialisation des objets globaux situés dans des unités de compilation différentes n'est pas précisé par le standard. En quoi cela peut-il être un problème ?
4. Pourquoi est-il conseillé de définir un constructeur par copie et un opérateur d'affectation dans une classe possédant un (des) attribut(s) de type pointeur ?
5. On rappelle que la *mise en ligne* du corps d'une fonction (*inlining*) est une optimisation consistant à remplacer un appel de fonction ou de méthode par le corps de celle-ci au site d'appel (en remplaçant les arguments formels par les arguments effectifs).

   En C++, il existe plusieurs situations dans lesquelles une fonction (ou méthode) ne peut être mise en ligne ; citez-en deux. (On ne s'attache pas ici à la qualité de l'optimisation, donc une réponse telle que "le compilateur refuse l'inlining car la fonction à mettre en ligne est trop grosse" n'est pas pertinente).
6. 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.

3 Autres langages de programmation

1. Appariez (sur votre copie) chaque auteur avec son langage :

   a. Andrew Appel
   b. John Backus
   c. Ole-Johan Dahl
   d. Alan Kay
   e. Brian Kernighan
   f. Donald Knuth
   g. John McCarthy
   h. Kristen Nygaard
   i. Dennis Ritchie
   j. Bjarne Stroustrup
   k. Larry Wall
   l. Niklaus Wirth

   1. Algol W
   2. C
   3. C++
   4. FORTRAN
   5. Lisp
   6. Pascal
   7. Perl
   8. Simula
   9. Smalltalk
   10. TEX
   11. Tiger

2. En Algol 60, on peut implémenter une technique de programmation appelée *Jensen's Device* (inventée par Jørn Jensen, qui a travaillé avec Peter Naur), qui permet d'écrire ceci :

```
real procedure sum (i, lo, hi, term);
  value lo, hi;
  integer i, lo, hi;
  real term;
  comment 'term' is passed by–name, and so is 'i';
begin
  real temp;
  temp := 0;
  for i := lo step 1 until hi do
    temp := temp + term;
  sum := temp
end;
```

   Cette technique repose sur le passage d'argument par nom (*call by name*) présent dans Algol 60.

   (a) Rappelez la signification de "Algol".
   (b) Qu'appelle-t-on passage par nom ?
   (c) Réécrivez ce bout de code en C++.
   (d) Écrivez un bout de code Algol utilisant la procédure `sum` du code Algol ci-dessus, permettant d'afficher le 100e nombre harmonique noté H_{100}, défini ainsi :

$$H_{100} = \sum_{i=1}^{100} \frac{1}{i}$$

   (Vous ne serez bien entendu pas pénalisés sur des erreurs de syntaxe mineures.)

3. Considérez le code Haskell ci-dessous.

```
1 let ones = 1 : ones
2 in head ones
```

(a) Que fait la ligne 1 ?

(b) Pourquoi le programme ci-dessus ne boucle-t-il pas indéfiniment ?