

Vérifiez votre énoncé: les 5 entêtes doivent être +1/1/*xx*+... +1/5/*xx*+

Ing1 2016 – TYLA – 1h30 – Avril 2014 *Sans document ni machine*

Noircir les cases plutôt que cocher. Renseigner les champs d'identité. Les questions marquées du symbole ♣ peuvent avoir plusieurs réponses justes. Toutes les autres questions n'ont qu'une seule réponse juste; si plusieurs réponses sont valides, sélectionner la plus restrictive (par exemple s'il est demandé si 0 est *nul*, *non nul*, *positif*, ou *négatif*, sélectionner *nul*). Il n'est pas possible de corriger une erreur. Les réponses justes créditent; les incorrectes pénalisent; et les blanches et réponses multiples valent 0.

Nom et prénom :

.....

Cochez votre identifiant (de haut en bas):

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

1 Contrôle

Q.1 Avez-vous bien vérifié que les en-têtes des 5 pages de ce sujet, comme indiqué haut de cette première page ?

☒ Oui

☐ Non

2 Histoire de l'informatique

Q.2 Quelle invention ne peut-on associer à Douglas Engelbart ?

☐ *The Mother of All Demos*

☐ oN Line System (NLS)

☒ Smalltalk

☐ La souris

Q.3 À qui doit-on l'invention originelle d'Unix ?

☐ Richard Stallman

☒ Ken Thompson

☐ Brian Kernighan

☐ Bjarne Stroustrup

Q.4 Qui est le « B » de BNF ?

☒ John Backus

☐ Brian Kernighan

☐ Bjarne Stroustrup

☐ Barbara Liskov

Q.5 Quel langage introduit le concept de fonctions imbriquées (avec portée statique) ?

☐ COBOL

☐ PL/I

☐ FORTRAN

☒ ALGOL 60

Q.6 Qu'est-ce qu'un *bytecode* ?

☐ Une variable signée ou non signée sur 8 bits

☐ Un trou dans une carte perforée

☐ Un encodage de caractères mono-octet, comme ASCII

☒ Un code compilé exécutable dans une machine virtuelle

3 Passage d'arguments de fonctions

Q.7 À la fin du programme ci-dessous, avec un *Mode* de passage des arguments par valeur (copie), quelles sont les valeurs de `foo[0]`, `foo[1]` et `t` ?

```
var t      : integer
    foo    : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t      := 0;
    x      := x + 8;
end;
```

```
begin
    foo[0] := -1;
    foo[1] := 0;
    t      := 1;
    shoot_my(foo[t]);
end.
```

- | | |
|---|--|
| ☒ <code>foo[0] = 43, foo[1] = 0, t = 0</code> | ☐ <code>foo[0] = 8, foo[1] = 0, t = 0</code> |
| ☐ <code>foo[0] = 43, foo[1] = 8, t = 0</code> | ☐ <code>foo[0] = 51, foo[1] = 0, t = 0</code> |

Q.8 Même question, mais avec un *Mode* de passage d'arguments par valeur-résultat, à la Algol W. On rappelle qu'en Algol W, la l-value dans laquelle est copiée la valeur d'un argument passé par résultat ('out') est évaluée *au retour* de la fonction.

- | | |
|--|--|
| ☐ <code>foo[0] = 43, foo[1] = 0, t = 0</code> | ☒ <code>foo[0] = 8, foo[1] = 0, t = 0</code> |
| ☐ <code>foo[0] = 43, foo[1] = 8, t = 0</code> | ☐ <code>foo[0] = 51, foo[1] = 0, t = 0</code> |

Q.9 Même question, mais avec un *Mode* de passage d'arguments par valeur-résultat, à la Ada. On rappelle qu'en Ada, la l-value dans laquelle est copiée la valeur d'un argument passé par résultat ('out') est évaluée *à l'appel* de la fonction.

- | | |
|---|--|
| ☒ <code>foo[0] = 43, foo[1] = 8, t = 0</code> | ☐ <code>foo[0] = 51, foo[1] = 0, t = 0</code> |
| ☐ <code>foo[0] = 8, foo[1] = 0, t = 0</code> | ☐ <code>foo[0] = 43, foo[1] = 0, t = 0</code> |

Q.10 Même question, mais avec un *Mode* de passage d'arguments par référence.

- | | |
|--|---|
| ☐ <code>foo[0] = 8, foo[1] = 0, t = 0</code> | ☐ <code>foo[0] = 43, foo[1] = 0, t = 0</code> |
| ☐ <code>foo[0] = 51, foo[1] = 0, t = 0</code> | ☒ <code>foo[0] = 43, foo[1] = 8, t = 0</code> |

Q.11 Même question, mais avec un *Mode* de passage d'arguments par nom.

- | | |
|--|---|
| ☐ <code>foo[0] = 43, foo[1] = 8, t = 0</code> | ☒ <code>foo[0] = 51, foo[1] = 0, t = 0</code> |
| ☐ <code>foo[0] = 43, foo[1] = 0, t = 0</code> | ☐ <code>foo[0] = 8, foo[1] = 0, t = 0</code> |

Q.12 De quel type de passage d'arguments ne dispose-t-on pas, ou ne peut-on simuler, en C++ ?

- | | |
|---|---|
| ☐ Le passage par référence | ☒ Le passage par valeur-résultat à la Algol W |
| ☐ Le passage par nom | ☐ Le passage par valeur |

4 Programmation orientée objet

Q.13 En Smalltalk 76, comment instancie-t-on une classe ?

CORRECTION

- ☐ En appelant son constructeur.
- ☐ Grâce à la primitive 'to'.
- ☐ En envoyant un message 'new' à `Object`.
- ☒ En envoyant un message 'new' à `Class`.

Q.14 Qu'appelle-t-on une métaclasse en Smalltalk 76 et 80 ?

- ☐ La classe dont dérivent toutes les classes, directement ou indirectement, explicitement ou implicitement.
- ☐ Une classe paramétrée.
- ☒ Une classe dont les instances sont des classes.
- ☐ Une classe qui n'a pas d'instance.

Q.15 Dans quel langage a été introduit la notion de *classe* ?

- ☐ COBOL
- ☐ Smalltalk
- ☐ ALGOL
- ☒ Simula

Q.16 Dans quel langage a été introduit la notion d'*objet* ?

- ☒ Simula
- ☐ COBOL
- ☐ Smalltalk
- ☐ ALGOL

Q.17 Pour quelle raison peut-on souhaiter rendre un destructeur virtuel en C++ ?

- ☐ Pour rendre une classe abstraite.
- ☐ Pour rendre une classe non copiable.
- ☒ Pour que le constructeur du type dynamique de l'objet soit appelé.
- ☐ Pour que le constructeur du type statique de l'objet soit appelé.

Q.18 Templates vs méthodes virtuelles en C++ : quelle est la bonne réponse ?

- ☐ les instanciations de templates et les liaisons de méthodes virtuelles sont faites à la compilation.
- ☐ les instanciations de templates et les liaisons de méthodes virtuelles sont faites à l'exécution.
- ☐ les instanciations de templates sont faites à l'exécution tandis que les liaisons de méthodes virtuelles sont faites à la compilation.
- ☒ les instanciations de templates sont faites à la compilation tandis que les liaisons de méthodes virtuelles sont faites à l'exécution.

Q.19 Les multiméthodes permettent

- ☐ aux méthodes de retourner plusieurs résultats.
- ☒ le polymorphisme dynamique sur plusieurs arguments de fonctions.
- ☐ d'avoir des méthodes polymorphes (virtuelles) dans une hiérarchie de classe utilisant l'héritage multiple.
- ☐ à une classe d'avoir des méthodes portant le même nom (mais des arguments différents).

5 Programmation fonctionnelle

Q.20 On dit d'un langage qu'il est fonctionnel si...

- ☐ il n'effectue aucun effet de bord.
- ☒ il permet de manipuler des fonctions comme n'importe quel autre entité/objet.
- ☐ il est Turing complet.
- ☐ il supporte le concept de fonction récursive.

Q.21 Comment appelle-t-on une fonction qui capture des références à des variables libres dans l'environnement lexical ?

CORRECTION

- ☒ Une fermeture ☐ Une fonction d'ordre supérieur
☐ Une fonction récursive terminale ☐ Une lambda abstraction

Q.22 Que vaut la valeur `integer` dans le code Haskell ci-dessous ?

```
reapply f a = a : reapply f (f a)
incr x = x + 1
integers = reapply incr 0
```

- ☐ [1, 1, 1, 1, ...] ☐ [1, 2, 3, 4, ...] ☒ [0, 1, 2, 3, ...] ☐ [0, 0, 0, 0, ...]

Q.23 Quel trait du langage Haskell permet d'écrire le programme de la question précédente, produisant une liste a priori non finie ?

- ☒ L'évaluation paresseuse ☐ L'évaluation stricte
☐ La compilation séparée ☐ La séparation interface/implémentation

Q.24 Quel est le trait de langage illustré dans l'exemple Objective Caml ci-dessous exécuté dans le *toplevel* (interprète) ?

```
# let f x y = x + y;;
val f : int -> int -> int = <fun>
# let g x y = x +. y;;
val g : float -> float -> float = <fun>
```

- ☐ Les types polymorphes ☒ L'inférence de types
☐ L'application partielle ☐ La surcharge d'opérateurs

Q.25 Comment s'appelle la transformation suivante ?

$$\lambda x(\lambda y \mapsto \sin(x \cdot y)) \rightsquigarrow \lambda(x, y) \mapsto (x \cdot y)$$

- ☐ Le désucrage ☒ La dé-curryfication ☐ La curryfication ☐ La mise en ligne

6 Programmation générique

Q.26 Quel langage ne dispose pas d'une fonctionnalité conçue pour contraindre les paramètres des types paramétrés ?

- ☒ C++ 2011 ☐ Eiffel ☐ Java ☐ Ada

Q.27 Dans quel langage doit-on explicitement instancier les types paramétrés avant de pouvoir les utiliser ?

- ☐ C# ☐ Java ☒ Ada ☐ C++ 1998/2003

Q.28 Quelle différence y a-t-il entre macros et templates en C++ ?

- ☐ Les macros sont résolues à l'exécution, les templates à la compilation.
☐ Les macros figurent dans le premier standard C++ (publié en 1998), mais pas les templates.
☐ Les macros sont résolues à la compilation, les templates à l'exécution.

CORRECTION

■ Les macros ne supportent pas la récursion.

Q.29 Les concepts du C++ ISO 2011

- | | |
|--|--|
| ☐ se définissent grâce au mot clef concept . | ☐ sont vérifiés explicitement par le compilateur. |
| ☐ sont compilés automatiquement aux sites d'utilisations. | ■ expriment des contraintes sur les paramètres de templates. |

Q.30 Le code C++ 2011 ci-dessous :

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

int main()
{
    std::vector<double> v{1., 51., 4.3};
    std::sort(begin(v), end(v));
}
```

- | |
|--|
| ☐ 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 <code>std::vector<float>::iterator</code> et le concept <code>RandomAccessIterator</code> . |
| ■ compile correctement et s'exécute sans erreur. |

Fin de l'épreuve.

CORRECTION