EPITA ING1 2016 S2 PFON

Didier Verna
Documents et calculatrice interdits

Question N° 1 :

Parmi les points suivants, lequel n'est pas représentatif de la notion de fonction d'ordre supérieur ?

Réponses possibles :

a. le passage de fonctions en argument d'autres fonctions,
b. le renvoi de fonctions en retour d'autres fonctions,
c. l'absence d'effet de bord dans une fonction,
d. la création de fonctions anonymes (littérales).

Fonction ordre supérieur =
- Une ou n fonctions en entrée
- Renvoi une fonction (via la création de fonction anonyme)

Question N° 2 :

Dans un langage fonctionnel pur :

Réponses possibles :

a. les fonctions ne prennent qu'un seul argument,
b. il n'y a que des constantes,
c on ne peut pas faire d'entrées/sorties,
d. il n'y a que des fonctions.

- Pas d'effets de bord
- Pour une entrée, on a toujours la même sortie

Question N° 3 :

On parle de typage statique quand :

Réponses possibles :

a. le type des variables est connu à la compilation,
b. le type des valeurs est connu à la compilation,
c. le type des valeurs est fixe pendant l'exécution,
d. tous les types doivent être donnés explicitement dans le source.

Question N° 4 :

En Haskell, il existe un séparateur implicite qui remplace l'offside rule lors du parsing. Ce séparateur est :

Réponses possibles :

a. .
b. ;
c. ,
d. !

Offside rule = Indentation comme syntaxe (comme en python)

Question N° 5 :

En Haskell, que signifie `[a] -> Int` ?

Réponses possibles :

a. stocker le contenu de `[a]` dans la variable `Int`,

b. le type de la liste `[a]` est `Int`,

c. une fonction d'une liste de n'importe quel type a vers un entier,

d. la liste infinie de tous les entiers a.

Question N° 6 :

En LISP, les valeurs booléennes sont représentées par :

Réponses possibles :

a. `#f` et `#t` <- Utilisé dans le langage Scheme

b. `nil` ou la liste vide, et tout sauf `nil`,

c. `'false` et `'true`

d. `:false` et `:true`

Question N° 7 :

Que fait la fonction `f` suivante en haskell :

```
elt :: a -> [a] -> Bool
elt x [] = False
elt x (x:xs) = True
elt x (y:ys) = elem x ys
```

Réponses possibles :

a. elle teste si un objet est une liste,

b. elle teste si un objet est membre d'une liste,

c. une erreur de syntaxe,

d. une boucle infinie.

La 3e ligne ne fonctionne pas car dans (x:xs), on utilise un x qui est déjà défini juste avant.
GHC nous donne l'erreur suivante :

```
Conflicting definitions for 'x'
Bount at:   file.hs:3:5
            file.hs:3:8
In an equation for 'elt'
```

Question N° 8 :

En LISP, que donne l'évaluation de l'expression `(e1 e2 e3)` ?

Réponses possibles :

a. La liste composée des symboles `e1`, `e2` et `e3`,

b. la liste composée des valeurs des variables `e1`, `e2` et `e3`,

c. l'appel de la fonction nommée `e1` sur les symboles `e2` et `e3`,

d. l'appel de la fonction nommée `e1` sur les valeurs des variables `e2` et `e3`.

En LISP, l'évaluation est stricte, on évalue d'abord les arguments e2 et e3 avant d'appeler la fonction e1.
En Haskell c'est l'inverse.

	Fonct.	Évaluation	Typage	Autres
Lisp	impur*	stricte*	dynamique*	...*
Haskell	pur	lazy	statique	...

Question N° 9 :

En Haskell, que représente l'expression `(x:xs)` ?

Réponses possibles :

a. la liste ayant pour premier élément `x`, et pour reste la liste `xs`,

b. la liste ayant pour premier élément `x`, et pour deuxième élément `xs`,

c. la liste ayant pour premier élément `x`, et pour reste l'élément `xs`,

d. la liste `x` concaténée avec l'élément `xs`.

Question N° 10 :

Quelle est la valeur de l'expression suivante en Haskell :

```
[ x == 3 | x <- [2, 3, 4]]
```

Réponses possibles :

a. `[3]`

b. `[False, True, False]`

c. `3`

d. `True`

Question N° 11 :

Qu'appelle t'on « tree-accumulation » ?

Réponses possibles :

a. le remplacement des paramètres formels par leur valeur dans un appel de fonction,

b. le déploiement d'une fonction primitive-récursive sur la pile,

c. l'accumulation de bourgeons dans les arbres au printemps,

d. l'évaluation récursive de gauche à droite de toutes les sous-expressions d'une expression fonctionnelle.

Question N° 12 :

Qu'appelle t'on « opérateur spécial » en Lisp ?

Réponses possibles :

a. une fonction built-in du standard,

b. une fonction n'obéissant pas aux règles de l'évaluation stricte,

c. une fonction pouvant s'utiliser en notation infixe (par exemple `(a + b)`),

d. c'est l'autre nom des macros.

Question N° 13 :

Dans la fonction `(defun sq (x) (* x x))`, la variable `x` est :

Réponses possibles :

a. liée,

b. libre,

c. dynamique,

d. lexicale.

Variable :
- liée = définie dans le contexte (bloc) local
- libre = non définie localement

Question N° 14 :

Que vaut l'évaluation de l'expression suivante en Lisp :

Réponses possibles :

a. 3

b. 1

c. Unbound variable : b

d. Unbound variable : a

Annulée

Question N° 15 :

Qu'est-ce que le scoping lexical ?

Réponses possibles :

a. rechercher les valeurs des variables libres dans l'environnement de définition de l'expression concernée,

b. utiliser des noms différents selon le type des identifieurs (par exemple `Int` pour un type et `int` pour une variable,

c. se doter de mots réservés (par exemple `Int`, `Bool` *etc.*,

d. interdire que des fonctions et des variables portent le même nom.

Scoping :
- Lexical = Recherche dans l'environnement de définition
- dynamique = Recherche dans l'environnement d'appel

Question N° 16 :

Que signifie l'expression « Lisp-2 » ?

Réponses possibles :

a. qu'il existe des espaces de noms distincs pour les variables et les fonctions,

b. la deuxième version du standard Ansi,

c. le *deuxième* dialecte de Lisp, après Scheme,

d. elle ne veut rien dire.

Question N° 17 :

Qu'appelle t'on « lambda expression » en Lisp ?

Réponses possibles :

a. c'est l'autre nom des macros,

b. une expression utilisant un jeu de caractères non standard,

c. l'expression d'une fonction anonyme,

d. l'utilisation du symbole spécial `lambda` comme variable.

Question N° 18 :

Qu'appelle t'on « mapping » dans un langage fonctionnel ?

Réponses possibles :

a. l'application d'une fonction à tous les éléments d'une liste,

b. l'utilisation de tables de hash pures (hashmaps),

c. le remplacement des paramètres formels par leurs valeurs lors d'un appel de fonction,

d. l'association d'un nom de variable libre avec la valeur la plus « proche » selon le scoping utilisé.

Question N° 19 :

Que fait la fonction `complement` de Lisp ?

Réponses possibles :

a. elle produit de la complétion automatique dans la Repl,

b. elle prend un entier et renvoie son complémentaire,

c. elle prend une fonction booléenne et renvoie une fonction produisant le résultat inverse,

d. elle ferme toutes les parenthèses qui manquent en fin d'une expression.

Question N° 20 :

Que fait la fonction suivante en Haskell :

```
f n = \ x -> x + n
```

Réponses possibles :

a. elle prend un nombre n et renvoie une fonction qui ajoute n à son argument,

b. elle prend deux arguments et renvoie leur somme s'ils sont égaux,

c. elle redéfini la fonction = à la fonction somme,

d. syntax error.