INFO0054 – Examen Programmation Fonctionnelle

2023-01-13

Instructions : Il s'agit d'un examen à livre fermé et vous disposez de 3 heures. Ces heures comprennent le temps dont vous avez besoin pour copier vos brouillons. Utilisez une feuille par question et indiquez le numéro de la question sur la ou les feuilles de papier. Sur chaque feuille de papier, indiquez votre nom en majuscules, votre prénom et votre numéro d'étudiant.

Question 1 : Théorie (25 points)

A) Que sont les fonctions d'ordre supérieur (2p) et comment *définissons-nous* et *utilisons*-nous les fonctions d'ordre supérieur en Scala (3p) ?	5p
B) Qu'est-ce que l'évaluation non-stricte (1p) et pourquoi l'évaluation non-stricte est-elle importante dans la programmation fonctionnelle ? (2p) Comment l'utilisons-nous en Scala (2p) ? Donnez et décrivez un exemple, écrit en code, convaincant (e.g., calculer le produit de tous les nombres dans une liste) pour illustrer l'importance. (5p)	10p

C) Qu'est-ce qu'un monoïde (2p) ? Donnez un exemple concret et définissez-le en Scala (4p). Expliquez le rôle des monoïdes dans la représentation des structures algébriques et (2p) les définitions des opérations sur ces structures (2p). C'est-à-dire, comment les monoïdes sont-ils utilisés dans la programmation fonctionnelle ?
 	
`package monoids`
 	
`trait Monoid[A]:`
`  def op(o1: A, o2: A): A`
`  val id: A`
 	
`object Monoid:`
`  // MONOIDS AND FUNCTIONS TO ILLUSTRATE YOUR POINTS`
`  // TO BE DECLARED IN THIS COMPANION OBJECT`
	10p

D) BONUS: Décrivez et illustrez les lois des foncteurs.	2p

Question 2 : Récursion sur les listes (8 points)

La fonction **compute** prend une liste **l** d'éléments et une fonction **f** (qui prend deux arguments) comme arguments. La fonction **compute** renvoie une nouvelle liste contenant le résultat de l'application de **f** à chaque paire consécutive de la liste **l.** Si la longueur de **l** est < 2, vous retournez la liste vide.

- Définissez la fonction **compute** d'une manière récursive. (3p)
- Redéfinir cette fonction avec un processus itératif. (3p)
- Votre solution au problème est-elle un exemple de récursivité structurelle ou de récursivité structurelle complète ? (2p) Expliquez brièvement votre réponse. Un simple « oui » ou « non » ne justifiera pas de notes.

```
scala> compute(List(1,2,3,4), _ * _)        →  val res0: List[Int] = List(2, 6, 12)
scala> compute(List(1,2,3,4), _ + "|" + _)  →  val res1: List[String] = List(1|2, 2|3, 3|4)
```

Question 3 : Récursion sur les nombres (8 points)

- Définissez la fonction **tabulate**. Cette fonction prend en entrée deux nombres entiers (**rows** et **columns**) et une fonction **f.** La fonction **f** prend deux nombres entiers (numéro de ligne et numéro de colonne) en entrée et renvoie un nouvel objet. La fonction **tabulate** renvoie une *liste de listes* contenant les valeurs d'une fonction donnée sur des plages de valeurs de nombres entiers commençant **à partir de 1**. Assurez-vous que votre implémentation soit aussi efficace que possible. **Vous ne devez pas utiliser la fonction de tabulate de Scala !** (5p)
- Quel est le nom d'une fonction dont la récursion se porte sur plusieurs arguments ? (1p)
- Votre solution au problème est-elle un exemple de récursivité structurelle ou de récursivité structurelle complète ? (2p) Expliquez brièvement votre réponse. Un simple « oui » ou « non » ne justifiera pas de notes.
- BONUS : définissez tabulate2 qui s'appuie sur la bibliothèque standard de Scala. Vous êtes autorisé à utiliser tout ce qui est proposé par Scala, à l'exception du tabulate de Scala. (3p)

```
scala> tabulate(2, 3, _ + "" + _)
val res0: List[List[String]] = List(List(11, 12, 13), List(21, 22, 23))
scala> tabulate(2, 0, _ * _)
val res1: List[List[Int]] = List(List(), List())
```

Question 4 : ADTs, Scala et HOP (4 points)

Étant donné une longue liste de données météorologiques où chaque élément est une observation contenant la **date**, le **minimum** et le **maximum** :

```
val data = List (
    Observation("220819", 14.0, 24.0), Observation("220820", 13.0, 25.0),
    Observation("220821", 15.0, 24.0), Observation("220822", 17.0, 24.0),
    Observation("220823", 16.0, 27.0))
```

- Définissez un ADT qui vous permet de créer et de représenter des **Observation**s. (1p)
- Définissez une fonction **avg** qui renvoie la moyenne d'une observation. (1p)
- Écrivez une expression qui donne, à partir d'une liste d'observations, l'**avg** de toutes les observations avec un minimum supérieur à 13,0. **Vous devez utiliser les fonctions fournies par la bibliothèque standard Scala pour cette question.** (2p)

Question 5 : Gestion des exceptions (5 points)

Vous avez accès à une fonction **parse** qui essaie de renvoyer un nombre entier contenu dans une chaîne.Remarquez comment cette fonction renvoie des objets de type **Either.**

```
scala> parse("42") → Right(42)
scala> parse("one") → Left(Invalid string)
```

L'inverse d'un nombre entier A est $\frac{1}{A}$ puisque $A \times \frac{1}{A} = 1$. Tous les nombres entiers autres que 0 ont un inverse.

Définissez une fonction **inverse** qui prend en entrée un nombre entier **i** et renvoie un objet qui représente soit le résultat sous la forme d'un **Double**, soit l'exception sous la forme d'un **String**. (3p) Définissez ensuite une fonction **process** qui prend en entrée une chaîne et utilise à la fois les fonctions **parse** et **inverse** pour calculer un résultat. Vous n'êtes pas autorisé à utiliser le filtrage par motif (pattern matching). (2p)

```
scala> process("42") → Right(0.023809523809523808)
scala> process("one") → Left(Invalid string)
scala> process("0") → Left(Cannot take the inverse of 0.)
```