Université de Montpellier
-
Faculté des Sciences

Examen terminal écrit : Session 1	Durée de l'épreuve : 2 heures
Date : 12 mai 2016	Documents autorisés : tous
Mention Informatique	
Licence $2^{ème}$ année : Programmation applicative (HLIN403)	

Les documents et les calculatrices sont autorisés (ce qui veut dire que la réponse à une question de cours doit montrer que vous avez *compris* la notion). Lisez l'ensemble du sujet. Les différentes sous-parties peuvent être traitées indépendamment. N'oubliez pas d'inscrire les informations demandées sur vos copies et relisez-vous 5 minutes avant de rendre votre copie. N'inscrivez aucun signe distinctif sur votre copie. Bonne chance !

1 Récursivité terminale

Exercice 1. (2 points) Rappelez brièvement ce qu'est la récursivité terminale par rapport à la récursivité enveloppée. Donnez un exemple pour chacune.

Exercice 2. (2 points) Écrire en récursivité enveloppée une fonction `count` qui prend en paramètres un élément et une liste et compte le nombre d'occurences de l'élément dans la liste. Exemples :

```
> (count 'a '(a b (a a c) (c (d a) e) d))
4
> (count 'a '(b c d (e f)))
0
```

Exercice 3. (2 points) Réécrire la fonction précédente en récursivité terminale, en expliquant les changements que vous avez appliqués par rapport à l'écriture de l'exercice précédent.

2 Récursivité arborescente

On dispose de la structure de donnée d'arbre suivante : un arbre est une paire composée d'un élément et d'une liste d'arbres représentant les fils du nœud actuel. L'interface disponible est la suivante (on ne demande pas de l'implémenter) :

Constructeur

`make-arbre e l` Construit l'arbre de racine `e` et ayant pour fils les éléments de la liste `l`.

Accesseurs

`element a` Renvoie la valeur de la racine de l'arbre `a`

`fils a` Renvoie la liste des fils de la racine de l'arbre `a`.

Prédicats

`arbre-vide? a` Renvoie vrai si `a` est un arbre vide.

`feuille? a` Renvoie vrai si `a` est réduit à une feuille.

Exercice 4. (3 points) Écrire une fonction `effeuillage` qui prend en paramètre un arbre et lui enlève toutes ses feuilles. Par exemple :

```
> (effeuillage '(1.((2.((3.())(4.((5.())(6.()))))))))
(1.((2.((4.())))))
```

3 Dataflow

Exercice 5. (1 point) Expliquez les grandes lignes de la programmation dataflow. Quelles sont les deux formes spéciales qui permettent de l'implémenter en scheme ? Expliquez leurs actions.

Exercice 6. (2 points)

1. Construire le flux de la suite des carrés entiers.
2. Que fait le flux suivant :

```
(define mysterious-flow
  (stream-filter (lambda (x) (not (= 0 (modulo x 7))))
                 integers))
```

où `integers` est le flux des entiers dans l'ordre croissant à partir de 1.

4 Abstraction sur les fonctions

Exercice 7. (2 points) Étant donnée une fonction f, on veut construire une fonction qui calcule une approximation de la dérivée de f. Pour dx "petit", on définit la fonction dérivée f' par :

$$f' : x \mapsto \frac{f(x+dx) - f(x)}{dx}$$

Écrire la fonction `derive` qui prend une fonction f en paramètre et une valeur pour dx, et renvoie la fonction f'. Exemple :

```
>(define cube (lambda (x) (* x x x)))
>(cube 10)
1000
>((derive cube 0.0001) 10)
300.00300000893
```

5 Structures de données abstraites

Une rotation autour de l'origine du plan en deux dimensions est représentable par une matrice à deux lignes et deux colonnes dont le déterminant est égal à 1 (pour simplifier). Voici quelques exemples de matrice de rotation :

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} \frac{1}{2} & \frac{-\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{pmatrix} \qquad \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

FIGURE 1 – Quelques exemple de rotations dans le plan, de centre l'origine et d'angle (en radian) respectivement, de gauche à droite : 0 (rotation identité), π, $\frac{\pi}{3}$ et un angle quelconque θ.

Plus généralement, on représente ces matrices sous la forme :

$$\begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$$

Voici l'interface souhaitée pour la structure de données Rotation :

Constructeur

`make-rotation a b c d` Construit la rotation correspondant à la matrice $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$

Accesseurs

`get-x11 rot`	Renvoie la valeur de la case en haut à gauche de la matrice de rotation `rot`
`get-x12 rot`	Renvoie la valeur de la case en haut à droite de la matrice de rotation `rot`
`get-x21 rot`	Renvoie la valeur de la case en bas à gauche de la matrice de rotation `rot`
`get-x22 rot`	Renvoie la valeur de la case en bas à droite de la matrice de rotation `rot`

Prédicats

`rotation? rot`	Renvoie vrai si `rot` est une matrice de rotation

Exercice 8. (2 points) Voici l'implémentation qui a été choisie pour le constructeur :

```
(define (make-rotation a b c d)
    (cons (cons a b) (cons c d)))
```

De quel type d'objet s'agit-il ? Donnez la représentation sous forme de boîtes et pointeurs de l'objet défini par :

```
(define m (make-rotation 1 0 0 1))
```

Exercice 9. (3 points) Implémenter l'interface pour la structure de données Rotation. Pour le prédicat `rotation`, on rappelle (pour simplifier) qu'une matrice est une rotation si elle satisfait les deux conditions suivantes :

- elle a la bonne structure (en termes de paires)
- son déterminant vaut 1. Rappel : le déterminant d'une matrice $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ est donné par $ad - bc$.

Exercice 10. (1 point) Modifiez le constructeur donné ci-dessus pour qu'il vérifie si le déterminant vaut 1 avant de créer la matrice (et renvoie un message d'erreur si ce n'est pas le cas).