ISIMA 1ère année

1er février 2005
Durée : 2 heures
Documents autorisés

PROGRAMMATION FONCTIONNELLE

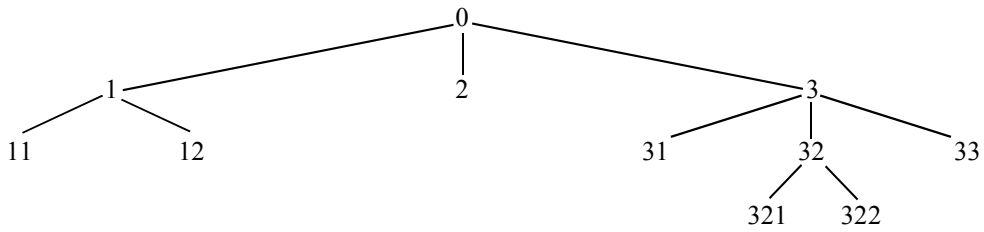
Exercice 1 :

Dans cet exercice, un arbre quelconque est représenté par une liste. Si cet arbre est vide, il est représenté par la liste vide. Si cet arbre est réduit à une feuille, il est représenté par une liste contenant l'étiquette de cette feuille. Sinon, il est représenté par une liste dont le premier élément est l'étiquette de sa racine et dont les éléments suivants sont des sous-listes représentant les sous-arbres qui sont les fils de cette racine. On convient que la profondeur de l'arbre vide est nulle, que celle d'un arbre réduit à une feuille est égale à 1 et qu'en général la profondeur d'un arbre est égale au maximum des profondeurs des sous-arbres fils de sa racine, incrémenté de 1.

Par exemple, la liste

```
(0 (1 (11) (12)) (2) (3 (31) (32 (321) (322)) (33)))
```

représente l'arbre représenté ci-dessous, dont la profondeur est égale à 4 :



Soit `SR` la fonction ci-dessous, représentant le schéma récursif sur une liste :

```
(define SR (lambda (L B C)
   (if L (C (car L) (SR (cdr L) B C)) B) ))
```

En utilisant le schéma `SR`, écrire une fonction `profondeur` ayant comme argument un arbre `A` et telle que l'évaluation de l'expression `(profondeur A)` retourne la profondeur de `A`.

On pourra utiliser la fonction prédéfinie `max`, ayant un nombre quelconque d'arguments numériques, et retournant le maximum de ses arguments.

Solution :

```
(define profondeur (lambda (A)
   (if A (+ 1 (SR (cdr A) 0 max)) 0) ))
```

Exercice 2 :

Dans cet exercice, une matrice carrée d'ordre *n* est représentée par une liste de *n* sous-listes de longueur *n*, chaque sous-liste représentant une ligne de la matrice. Ecrire une fonction `trace` ayant comme argument une matrice `M` et telle que l'évaluation de l'expression `(trace M)` retourne la trace de la matrice `M`, c'est-à-dire la somme de ses éléments diagonaux.

Solution :

```
(define trace (lambda (M)
   (if M (+ (caar M) (trace (map cdr (cdr M)))) 0) ))
```

Exercice 3 :

Soit *E* un ensemble. On appellera "suite dans *E* " toute application de $\mathbb{N}$ dans *E*. Par exemple la suite notée $(u_n)_{n\in\mathbb{N}}$ sera considérée comme l'application de $\mathbb{N}$ dans *E* qui à *n* associe u_n. On appellera "suite récurrente dans *E* " toute suite définie à partir d'une fonction *f* de *E* dans *E* et d'un élément u_0 de *E* par : $\forall\, n \in \mathbb{N}$, $u_{n+1} = f(u_n)$.

Remarque : dans les deux questions qui suivent, l'ensemble *E* est implicitement défini par la fonction `f` passée en paramètre, et il est donc inutile de le passer en paramètre.

a) Ecrire une fonction `suite` ayant comme arguments une fonction `f` de *E* dans *E* et un élément `u0` de *E* et telle que l'évaluation de l'expression `(suite f u0)` retourne la suite récurrente définie par `u0` et par "u_{n+1} = `f` (u_n)".

Indication : utiliser un `letrec`.

Réponse :

```
(define suite (lambda (f u0)
   (letrec ((u (lambda (n) (if (> n 0) (f (u (- n 1))) u0))))
   u ))
```

b) Si *E* est un ensemble fini, alors pour toute suite récurrente dans *E*, il existe un indice à partir duquel soit la suite est constante, soit elle est périodique. Ecrire une fonction `cte?` ayant comme arguments une fonction `f` et un élément `u0` de *E* et telle que l'évaluation de l'expression `(cte? f u0)` retourne `#t` si la suite récurrente définie par `u0` et par "u_{n+1} = `f` (u_n)" est constante et `#f` sinon.

Solution :

```
(define cte? (lambda (f u0) (c f (f u0) (list u0))))
(define c (lambda (f un L)
   (cond ((equal? un (car L)) #t)
         ((member? un L) #f)
         (else (c f (f un) (cons un L))) ) ))
```

Exercice 4 :

Une fonction en escalier f sur un intervalle $[a\,;\,b[$ de $\mathbb{R}$ peut être définie, pour $n \in \mathbb{N}^*$, par un $(n+1)$-uplet $(x_0, x_1, \ldots, x_n)$ et un n-uplet $(y_1, \ldots, y_n)$, de la manière suivante :

$$x_0 = a < x_1 < \ldots < x_n = b$$

et $$\forall\, k \in \{1, \ldots, n\},\ \forall\, x \in [x_{k-1}\,;\,x_k[,\ f(x) = y_k.$$

Une telle fonction sera représentée par une liste comportant $n+1$ éléments, telle que le premier élément est x_0 et les n éléments suivants sont des couples (x_k, y_k) pour $k \in \{1, \ldots, n\}$.

1°) Ecrire une fonction `fct` ayant comme argument une liste `L` représentant une fonction en escalier f sur un intervalle $[a\,;\,b[$ de $\mathbb{R}$, et telle que, pour tout symbole `x` associé à une valeur réelle, l'évaluation de l'expression `((fct L) x)` retourne "f(`x`)" si `x` $\in [a\,;\,b[$ et `()` sinon.

Par exemple si `L1` est associé à la liste `(0 (0.7 1.5) (1.5 0.8) (2 1.2))`, l'évaluation de l'expression `(fct L1)` doit retourner une fonction qui :

- à tout élément x de [0 ; 0.7 [associe 1.5 ;
- à tout élément x de [0.7 ; 1.5 [retourne 0.8 ;
- à tout élément x de [1.5 ; 2 [retourne 1.2 ;
- et à toute autre valeur réelle retourne `()`.

Solution :

```
(define fct (lambda (L)
   (lambda (x)
      (if (< x (car L)) () (f x (cdr L))) ) ))
(define f (lambda (x L)
   (if L
      (if (< x (caar L)) (cadar L) (f x (cdr L)))
      () ) ))
```

2°) Ecrire une fonction `plus` ayant comme arguments deux listes `L1` et `L2` représentant chacune une fonction en escalier définie sur un même intervalle $[a\,;\,b[$ de $\mathbb{R}$, et telle que l'évaluation de l'expression `(plus L1 L2)` retourne la fonction somme de ces deux fonctions en escalier (cette fonction devant toujours retourner `()` dans le cas où son argument n'appartient pas à $[a\,;\,b[$).

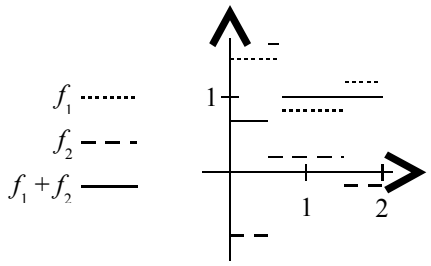
Par exemple si `L1` est associée à la liste

`(0 (0.7 1.5) (1.5 0.8) (2 1.2)`

donnée en exemple ci-dessus (et représentant la fonction f_1 dont la courbe est représentée en pointillés courts sur la figure ci-dessous), et si `L2` est associé à la liste

`(0 (0.5 -0.8) (1.5 0.2) (2 -0.2))`

(représentant la fonction f_2 dont la courbe est représentée en pointillés longs sur la figure ci-dessous), alors l'évaluation de l'expression `(plus L1 L2)` retourne la fonction $f_1 + f_2$ représentée par la liste `(0 (0.5 0.7) (0.7 1.7) (2 1))`, (et dont la courbe est représentée en traits pleins sur la figure ci-dessous).



Solution :

```
(define plus (lambda (L1 L2)
   (cons (car L1) (ed (p (cdr L1) (cdr L2)))) ))
(define p (lambda (L1  L2)
   (if L1
      (cond ((< (caar L1) (caar L2))
             (cons (list (caar L1) (+ (cadar L1) (cadar L2)))
                   (p (cdr L1) L2) ) )
            ((< (caar L2) (caar L1))
             (cons (list (caar L2) (+ (cadar L1) (cadar L2)))
                   (p L1 (cdr L2)) ) )
            (else
             (cons (list (caar L1) (+ (cadar L1) (cadar L2)))
                   (p (cdr L1) (cdr L2)) ) ) )
      () ) ))
(define ed (lambda (L)
   (if (cdr L)
      (if (equal? (cadar L) (cadar (cdr L)))
         (ed (cdr L))
         (cons (car L) (ed (cdr L))) )
      L ) ))
```