ISIMA 1ère année 2 février 2012

Durée : 2 heures

Documents autorisés

PROGRAMMATION FONCTIONNELLE

Exercice 1

1°) Ecrire une fonction `iota` ayant comme argument un nombre entier `n` et telle que l'évaluation de l'expression `(iota n)` retourne la liste `(1 … n)` si `n` est supérieur à 1 et `()` sinon.

La complexité de cette fonction (c'est-à-dire le nombre d'opérations effectuées) doit être une fonction linéaire de la taille de la liste.

2°) a) Ecrire une fonction `double` ayant comme argument une fonction `f` d'une seule variable et telle que l'évaluation de l'expression `(double f)` retourne la composée de `f` par elle-même (*i.e.* `f o f`).

Par exemple, si `inc` est une fonction qui ajoute 1 à son argument, alors `(double inc)` doit retourner une fonction qui ajoute 2 à son argument.

b) Quelle est la valeur retournée par

```
(map (lambda(x)(((double (double double)) inc) x)) (iota 3))
```

Exercice 2

Le but de cet exercice est d'écrire une fonction qui retourne la liste de toutes les permutations possibles d'une liste dont les éléments seront supposés deux à deux distincts.

1°) Ecrire une fonction `insertion` ayant comme arguments un élément `x` et une liste `L` et telle que l'évaluation de l'expression `(insertion x L)` retourne une liste contenant toutes les listes que l'on peut obtenir en insérant `x` en diverses positions dans `L`.

<u>Exemple</u> : L'évaluation de l'expression `(insertion 1 '(2 3))` doit retourner `((1 2 3) (2 1 3) (2 3 1))`.

2°) En utilisant la fonction `insertion`, écrire une fonction `permutation` ayant comme argument une liste `L` dont les éléments sont deux à deux distincts et telle que l'évaluation de l'expression `(permutation L)` retourne une liste contenant toutes les listes que l'on peut obtenir en permutant les éléments de `L`.

<u>Exemple</u> : L'évaluation de l'expression `(permutation '(1 2 3))` doit retourner `((1 2 3) (2 1 3) (2 3 1) (1 3 2) (3 1 2) (3 2 1))`.

Exercice 3

L'objectif de cet exercice est de trouver dans une liste la plus longue liste d'éléments consécutifs vérifiant une certaine propriété (ou bien l'une d'entre elles s'il y a plusieurs listes de longueur maximale).

La liste ne devra être parcourue qu'une seule fois.

Soit SR le schéma de réduction vu en cours, et défini de la manière suivante :

```
(define SR (lambda (L B C)
; L est une liste, B la valeur de SR si L est vide,
; C est une fonction de calcul.
   (if (null? L) B
       (C (car L) (SR (cdr L) B C))
    ) ))
```

Ecrire une fonction `PLSL` ayant comme arguments une liste `L` et un prédicat `P` pouvant s'appliquer aux éléments de `L` (c'est-à-dire une fonction prenant comme argument un élément de `L` et retournant `#t` ou `#f`) et telle que l'évaluation de l'expression `(PLSL L P)` retourne une plus longue liste d'éléments consécutifs de `L` vérifiant le prédicat `P` (*i.e.* pour lesquels `P` retourne `#t`).

Exemple : L'évaluation de l'expression

`(PLSL’(1 2 0 3 4 5 0 6 7 8 9 0 10 11 12) (lambda (x) (> x 0)))`

doit retourner `(6 7 8 9)`.

Décrire la sémantique des fonctions intermédiaires utilisées. Notamment, si `SR` est utilisé, décrire précisément la fonction C.

Solution :

```
(define PLSL (lambda (L P) (PG L P 0 () 0 ())))
(define PG (lambda (L P lcour SLcour lmax SLmax)
   (if (null? L) (miroir SLmax)
      (if (P (car L))
         (if (< lcour lmax)
 (PG (cdr L) P (+ lcour 1) (cons (car L) SLcour) lmax SLmax )
 (PG (cdr L) P 0 () (+ lcour 1) (cons (car L) SLmax)) )
         (PG (cdr L) P 0 () lmax SLmax) ) ) ))
(define miroir (lambda (L) (m L ())))
(define m (lambda (L LM)
   (if (null? L) LM (m (cdr L) (cons (car L) LM))) ))
```

Exercice 4

Rappels : Une fonction f définie dans un ensemble E et à valeurs dans un ensemble F est bijective si et seulement si, pour tout $y \in F$, il existe un unique $x \in E$ tel que $y = f(x)$. Si une fonction f de E dans F est bijective, on appelle fonction inverse de f, et on note f^{-1}, la fonction de F dans E définie par $y = f(x) \Leftrightarrow x = f^{-1}(y)$.

1°) Ecrire une fonction `bijective?` ayant comme arguments deux listes `E` et `F`, représentant deux ensembles E et F, et une fonction `fct` de `E` dans `F`, et telle que l'évaluation de l'expression `(bijective? fct E F)` retourne `#t` si `fct` est bijective et `#f` sinon.

<u>Solution</u> :

```
(define bijective? (lambda (fct E F)
   (egalensemble (map fct E) F) ))
(define egalensemble (lambda (E F)
   (or (and (null? E) (null? F))
       (and (membre? (car E) F)
            (egalensemble (cdr E) (enlever (car E) F)) ) ) ))
(define membre? (lambda (x E)
   (and (not (null? E))
        (or (equal? x (car E)) (membre? X (cdr E))) ) ))
(define enlever (lambda (x E)
   (if (equal? x (car E))
      (cdr E)
      (cons (car E) (enlever x (cdr E))) ) ))
```

2°) a) Ecrire une fonction `fonction` ayant comme arguments deux listes de même longueur `E` et `F`, représentant deux ensembles $E = \{e_1, \ldots, e_n\}$ et $F = \{f_1, \ldots, f_n\}$, et telle que l'évaluation de l'expression `(fonction E F)` retourne une fonction d'une seule variable qui à tout e_k de E associe l'élément f_k de F et à tout autre objet associe le symbole `indefini`.

<u>Solution</u> :

```
(define fonction (lambda (E F)
   (lambda (x)
      (if (null? E) 'indefini
         (if (equal? x (car E))
            (car F)
            ((fonction (cdr E) (cdr F)) x) ) ) ) ))
```

b) Ecrire une fonction `inverse` ayant comme arguments une liste `E`, représentant un ensemble E, et une fonction `fct` définie dans `E` et supposée bijective, et telle que l'évaluation de l'expression `(inverse fct E)` retourne une liste contenant dans l'ordre l'inverse de `fct` et son domaine de définition.

<u>Solution</u> :

```
(define inverse (lambda (fct E)
   (let ((F (map fct E)))
      (list (fonction F E) F) ) ))
```

c) Ecrire une expression Scheme dont l'évaluation permet d'obtenir l'ensemble d'arrivée `(1 4 9)` de la fonction carré, $x \mapsto x^2$, sur l'ensemble représenté par `(1 2 3)`.

<u>Solution</u> :

```
(map (lambda (x) (* x x)) '(1 2 3))
```

d) Ecrire une expression Scheme dont l'évaluation permet d'obtenir l'image de 4 par la fonction inverse de la fonction `inc` définie sur l'ensemble {1, …, 10} (cf. exercice 1).

Solution :

```
((car (inverse inc (iota 10))) 4)
```