Université Bordeaux 1 Licence Informatique Semestre 5 2003/2004

Programmation Fonctionnelle et Symbolique : Devoir surveillé

Vendredi 14 Novembre 2003

Documents personnels autorisés Durée 2 heures

Chacun des quatre exercices doit être **rédigé** sur une feuille **séparée** sur laquelle doit être inscrit votre nom. Vous devez rendre exactement **une feuille** (ou copie), éventuellement blanche, **par exercice**.

Exercice 1 *Évaluation*

Évaluer les expressions suivantes :

1. `(cons '(A B C) '(1 2 3))`
2. `(append '(A B C) '((1 2) 3))`
3. `(last '((A 1) (B 2) (C 3)))`
4. `(butlast '((A 1) (B 2) (C 3)))`
5. `(list (1+ 2) (1- 5) 6)`
6. `(remove-if #'zerop '(0 1 0 2 0 3 0 0 0))`
7. `(remove-if-not (lambda (x) (= x 3)) '(0 1 2 3 0 1 2 3))`
8. `(mapcar (lambda (x) (* 2 x)) '(1 2 3))`
9. `(funcall (lambda (x y) (+ (* 2 x) y)) 2 3)`
10. `(assoc 'bleu '((rouge . red) (vert . green) (bleu . blue) (jaune . yellow)))`

Exercice 2 *Tableaux*

1. Écrire une fonction `size-vector (v n)` non destructive qui, à partir d'un vecteur `v`, construit et retourne un vecteur de dimension `n` qui contient les `n` premiers éléments de `v` si la dimension de `v` est supérieure ou égale à `n` ou les éléments de `n` complétés par des `0` si la dimension de `v` est strictement inférieure à `n`.

   Exemples:

   ```
   * (size-vector #(1 2 3 4) 8)

   #(1 2 3 4 0 0 0 0)
   * (size-vector #(1 2 3 4) 2)

   #(1 2)
   ```

2. Écrire une fonction `list-vector (v1 v2)` non destructive qui, étant donnés deux vecteurs `v1` et `v2`, construit et retourne un nouveau vecteur `v` de dimension le maximum des dimensions de `v1` et `v2` tel que la `i`ème entrée de `v` contient la liste à deux éléments constituée du `i`ème élément de `v1` et du `i`ème élément de `v2`. Si les longueurs des vecteurs sont inégales les éléments manquants sont remplacés par des `0`.

   Exemples:

   ```
   * (list-vector #(1 2) #(3 4 5 6))

   #((1 3) (2 4) (0 5) (0 6))
   * (list-vector #(3 4 5 6) #(1 2))

   #((3 1) (4 2) (5 0) (6 0))
   ```

Exercice 3 *Listes*
Soient les définitions de `mystere (op l)` et `*l*` suivantes :

```
(defun mystere (op l)
  (cond ((endp l) l)
        ((zerop (car l)) (mystere op (cdr l)))
        (t (cons (funcall op (car l)) (mystere op (cdr l))))))

(defparameter *l* '(0.0 1.0 2.0 0.0 4.0))
```

1. Évaluer l'expression `(mystere (lambda (x) (/ 1.0 x)) *l*)`.
2. Que fait la fonction `mystere (op l)`?
3. Écrire une version itérative de la fonction `mystere-it (op l)` en utilisant la forme `do` ou `do*`.
4. En utilisant des fonctions prédéfinies mais sans utiliser ni récursion ni itération, écrire une expression donnant le même résultat que l'appel `(mystere (lambda (x) (/ 1.0 x)) *l*)`.

Note: l'expression `(zerop 0.0)` retourne `T`.

Exercice 4 *Fonctionnelles*
Supposons une fonction f à n arguments: $f : x_1, \ldots, x_n \mapsto f(x_1, \ldots, x_n)$.
On définit la fonction projection $p_{f,i,v}$, à un seul argument, qui fixe tous les arguments de f à une valeur v sauf le $i$ème: $p_{f,i,v} : x_i \mapsto p_{f,i,v}(x_i) = f(v, \ldots, x_i, \ldots, v)$.

1. Définir la fonction `projection (f n i v)` qui retourne la fonction $p_{f,i,v}$ où `n` est le nombre d'arguments de la fonction f.

   Note: On pourra utiliser la fonction `apply` dont la documentation est donnée en annexe et la fonction `place (n i e xi)` donnée ci-dessous.
   Exemples d'utilisation de `apply`: l'expression `(apply #'+ '(1 2 3))` retourne `6` et l'expression `(apply #'cons '(a b))` retourne `(A . B)`.

   ```
   (defun place (n i e xi)
     (append (make-list i :initial-element e)
             (list xi)
             (make-list (- n (1+ i)) :initial-element e)))

   * (place 6 0 1 3)

   (3 1 1 1 1 1)
   * (place 6 2 1 3)

   (1 1 3 1 1 1)
   * (place 6 5 1 3)

   (1 1 1 1 1 3)
   ```

2. En utilisant une fonction f de votre choix, écrire un exemple d'appel de fonction obtenue par un appel à `projection`.
   Indiquer le résultat de cet appel.