Université Bordeaux 1 Licence S&T Mention Informatique 2005/2006

UE INF207 - Programmation Fonctionnelle et Symbolique
Devoir surveillé

Tous documents autorisés. Vendredi 31 Mars 2006
Durée : 1h30.
Le barême est donné à titre **indicatif**.
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 *(3pts)*

Évaluer les expressions suivantes :

1. `(list '(1 2) '(3) '(4 5 6))`
2. `(append '(1 2) '(3) '(4 5 6))`
3. `(cons '(1 2) '(3 4))`
4. `(remove '(a) '((a) (b) (c)))`
5.
```
(mapcar #'(lambda (e) (if (listp e) (length e) (list e)))
          '((1 (2 3) 4) 5 (6 7 8)))
```
6. `(remove-if #'evenp '((a . 2) (b . 3) (b . 4) (c . 5)) :key #'cdr)`

Exercice 2 *(5pts)*

Soit la fonction `mystere (l)` suivante :

```
(defun mystere (l)
  (cond
    ((endp l)
     '())
    ((listp (car l)) (nconc (mystere (car l)) (mystere (cdr l))))
    (t
     (cons (car l) (mystere (cdr l))))))
```

1. Que retourne `(mystere '(1 2 3 4 5))` ?
2. Que retourne `(mystere '(1 (2 3 (4 5)) (6)))`
3. Que fait la fonction `mystere` ?
4. Donner une version itérative de `mystere`

Exercice 3 *(5pts)*

Soit la fonction `f-n (f n)` qui prend en paramètre une fonction f d'une variable et un entier $n \geq 0$ et qui retourne la fonction f^n. On rappelle que $f^n : x \mapsto f(f(...(x)...))$ (f appliquée n fois).

1. Définir cette fonction.
2. Donner un exemple d'utilisation d'une fonction retournée par `f-n`.

Exercice 4 *(7pts)*
On veut améliorer la représentation des ensembles finis vue en TD. La fonction de comparaison n'est plus un argument de `ens-member`, `ens-adjoin`, etc., mais est stockée dans une structure. On assure ainsi la cohérence de la représentation, car la même fonction d'égalité est utilisée pour toutes les opérations sur un même ensemble.
Voici un début :

```
(defstruct ens
  (test #'eql)
  (elements '()))

(defun ens-empty (&key (test #'eql))
  (make-ens :test test))

(defun ens-member (x e)
  (member x (ens-elements e) :test (ens-test e)))
```

1. Écrire une fonction `ens-adjoin (x e)` créant la représentation de { `x` } ∪ `e` .
2. Définir la fonction `ens-from-list` qui permet de créer un ensemble à partir d'une liste (en supprimant les doublons). *On justifiera le choix des paramètres de cette fonction.*
3. Modifier la définition de la structure de façon à imprimer les ensembles de façon spécifique. L'idéal serait la notation utilisée dans les exemples ci-dessous, *mais nous accepterons toute notation lisible plus facile à implanter, par exemple set(e1 e2 ... en) ou {(e1 e2 ... en)}*.

   ```
   CL-USER> (ens-adjoin 33 (ens-from-list '(4 5 3 5 2 3)))
   ```
   {*33, 4, 5, 2, 3*}
4. Écrire une version *destructive* de la fonction `ens-adjoin` :

   ```
   CL-USER> (defparameter *ens* (ens-from-list '(4 5 6 7 8 9)))
   *ENS*

   CL-USER> (ens-nadjoin 1515 *ens*)
   ```
   {*1515, 4, 5, 6, 7, 8, 9*}

   ```
   CL-USER> *ens*
   ```
   {*1515, 4, 5, 6, 7, 8, 9*}