U.F.R. Mathématiques et Informatique — Maîtrise d'informatique
Techniques et fondements de la programmation - Pratique — jeudi 21 Novembre 2002

# Devoir surveillé

Documents personnels autorisés — Durée 2 heures

Chacun des quatre exercices doit être rédigé sur une feuille séparée sur laquelle doit être marquée 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 d))`
2. `(cddar '((a b c d) (e g) (h i j)))`
3. `(append '(a b) '((e g) (h i)))`
4. `(mapcar (lambda (x) (* x 10)) '(2 4 6))`
5. `(mapcar (lambda (x) (cons x nil)) '(1 2 3))`
6. `(funcall #'+ (list (1+ 2) 4 5))`

**Exercice 2** *Listes*
Soit la fonction `mystere` suivante :

```
(defun mystere-it (l)
  (do ((l l (cdr l))
       (acc nil (if (member (car l) (cdr l) :test #'equal)
                    acc
                    (cons (car l) acc))))
      ((endp l) (nreverse acc))))
```

1. Évaluer l'expression `(mystere-it '((a b) (d e) (a b c) (d e) (a b)))`.
2. Que fait la fonction `mystere-it`?
3. Écrire une version récursive `mystere-rec` de cette fonction.
4. Écrire une expression (utilisant une fonction lisp prédéfinie) équivalente à `(mystere-it '((a b) (d e) (a b c) (d e) (a b)))`.

**Exercice 3** *Fonctionnelles*

1. Définir la fonction `trinome (a b c)` qui renvoie la fonction d'une variable numérique définie par

$$\left\{ \begin{array}{lcl} \mathbb{R} & \to & \mathbb{R} \\ x & \mapsto & ax^2 + bx + c \end{array} \right.$$

2. Écrire une expression qui utilise une fonction renvoyée par un appel à `trinome` ainsi que la valeur renvoyée par cette expression.

**Exercice 4** *Structures et tables de hachage*

1. Écrire les définitions `defstruct` définissant les structures suivantes : la structure `point2d`, représentant un point du plan; les créneaux (ou *champs*) de cette structure sont :

   - `id`, valeur par défaut `0`. C'est l'identifiant unique du point. C'est un entier positif.
   - `x`, valeur par défaut `0.0`. C'est l'abscisse du point.
   - `y`, valeur par défaut `0.0`. C'est l'ordonnée du point.

   et la structure `point3d`, représentant des points de l'espace; les créneaux de cette structure sont

   - `id`, valeur par défaut `0`. C'est l'identifiant unique du point. C'est un entier positif.
   - `x`, `y` et `z`. Valeur par défaut `0.0`, pour les trois. Ce sont les coordonnées du point.
   - `points2d`. C'est une table de hachage, qui contient des `point2d` associés au `point3d`. La clé d'un `point2d` dans la table est la valeur de son créneau `id`.

2. Modifier la définition de la structure `point2d` de sorte à y ajouter une fonction d'impression. Donner le nouveau code pour la définition de la structure `point2d` et l'éventuel code de la fonction d'impression.

3. Placer dans une variable spéciale nommée `*test-point2d*` un point2d dont l'`id` vaut 3 et situé à la position $(x, y) = (3/4, 31/5)$.

4. Écrire la fonction `make-point2d-middle (id point2d-1 point2d-2)` qui retourne un nouveau point2d, dont l'`id` est l'`id` passé en argument de la fonction et dont la position est le milieu des deux points passés.

5. Écrire la fonction `put-point2d (point3d point2d)` qui place le `point2d` passé en deuxième argument dans le `point3d` passé en premier argument.

6. Écrire la fonction `suppress-points2d (point3d min-id max-id)` qui dans le point3d supprime les références aux points2d dont l'identifiant `id` est supérieur ou égal à `min-id` et inférieur ou égal à `max-id`.

   NB: Extrait de la documentation *Hyperspec* de la fonction `maphash` :

   ```
   The consequences are unspecified if any attempt is made to add or remove an
   entry from the hash-table while a maphash is in progress, with two exceptions:
   the function can use setf of gethash to change the value part of the entry
   currently being processed, or it can use remhash to remove that entry.
   ```