U.F.R. Mathématiques et Informatique — Maîtrise d'informatique
Techniques et fondements de la programmation - Pratique — Vendredi 23 Novembre 2001

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*

1. Évaluer les expressions suivantes :
   (a) `(cons '+ (list 2 3 4))`
   (b) `(cdadr '((a b c) (d e g) (h i j)))`
   (c) `(mapcar (lambda (x) (* x x x)) '(1 2 3))`
   (d) `(funcall #'list 2 3 4)`
   (e) `(apply #'+ '(2 3 4))`

2. Soit les fonctions `mystere` et `a` suivantes :

```
(defun mystere (l1 l2)
  (cond
    ((or (endp l1) (endp l2)) l1)
    ((a (car l1) l2) (mystere (cdr l1) l2))
    (t (cons (car l1) (mystere (cdr l1) l2)))))

(defun a (x l)
  (and l
       (or (eq (car l) x)
           (a x (cdr l)))))
```

   (a) Évaluer l'expression `(mystere '(a c e g) '(a b d e))`.
   (b) Que fait la fonction `mystere`?

Exercice 2 *Fonctionnelles*

1. Définir la fonction `compose (f g)` qui s'applique à deux fonctions d'une variable et renvoie la fonction composée ($\mathtt{g} \circ \mathtt{f}$) c'est-à-dire la fonction telle que $\forall \mathtt{x}, (\mathtt{g} \circ \mathtt{f})(\mathtt{x}) = \mathtt{g}(\mathtt{f}(\mathtt{x}))$.
2. Écrire une expression qui fait appel à la fonction renvoyée par un appel à `compose`.
3. Définir la fonction `compose-n (f n)` qui renvoie la fonction $\mathtt{f}^n = \mathtt{f} \circ \ldots \circ \mathtt{f}$. On rappelle que $\mathtt{f}^0$ est la fonction identité.

Exercice 3 *Listes*

1. Écrire une fonction `compte-occ (sym l)` qui compte les occurrences d'un atome `sym` dans une liste *généralisée* `l`. Une liste généralisée est une liste pouvant contenir des sous-listes.
2. On représente des mots par ses atomes. On dispose d'un petit dictionnaire Anglais-Français implémenté sous forme de liste d'associations.
   Exemple:

   ```
   * (assoc 'flour *dico*)

   (FLOUR . FARINE)
   *
   ```

   (a) Écrire une fonction `traduire-mot (a dico)` qui traduit un mot à l'aide du dictionnaire. Si le mot est un nombre, il est traduit en lui-même sinon sa traduction est recherchée dans le dictionnaire. Si le mot n'est pas présent dans le dictionnaire alors une liste contituée du symbole `?` et du mot original est renvoyée.
   Exemple:

   ```
   * (defparameter *recette* '(recipe 2 eggs 125 grams of flour 1 spoon of salt))

   *RECETTE*
   * (mapcar (lambda (x) (traduire-atome x *dico*)) *recette*)

   (RECETTE 2 OEUFS 125 GRAMMES DE FARINE 1 CUILLERE DE (? SALT))
   *
   ```

   (b) Écrire une fonction `traduire (l dico)` (où `l` est une liste généralisée de mots anglais) qui renvoie une liste ayant la même structure que `l` et contenant les mots traduits. Exemple:

   ```
   * (defparameter *recette* '(Ingredients (2 eggs 125 grams flour 1 spoon of salt)
     Preparation (mix all)))

   *RECETTE*
   * (traduire *recette* *dico*)

   (INGREDIENTS (2 OEUFS 125 GRAMMES FARINE 1 CUILLERE DE (? SALT))
    PREPARATION (MELANGER TOUT))
   *
   ```

Exercice 4 *Structures et tables de hachage*

1. On considère un interprète d'expressions arithmétiques écrites dans une syntaxe de type Lisp, utilisant les opérateurs `+`, `-`, `*`, et `/`. Cet interprète est aussi capable d'évaluer des variables, contenant des nombres.

   Voici quatre exemples d'expressions qu'il peut évaluer :

   ```
   (+ 1 2)
   (* 3 4 5)
   (/ a 2)
   (* a b (+ 2 3 b c d) (/ a 8))
   ```

   Écrire la structure `variable` qui servira à stocker, pour chaque variable, son nom (un symbole) et sa valeur (un nombre). On n'oubliera pas la fonction d'impression de cette structure.

2. On dispose d'une table de hachage `*environment*` qui contient l'environnement de l'interprète. Il s'agit de toutes les variables qui peuvent être utilisées dans les expressions calculables par l'interprète.

   Dans cette table de hachage, les clés sont les noms de variables, et les objets, des structures de type `variable`.

   Donner le code permettant de placer une variable nommée `a` et valant 12, dans la table `*environment*`.

3. Écrire la fonction `evaluate-arithm-expr (expr)`.

   La fonction `evaluate-arithm-expr` reçoit une expression évaluable par l'interprète, et retourne sa valeur dans l'environnement courant défini par `*environment*`.

   On ne traitera pas le cas des opérateurs `+`, `-`, et `/`.

   Exemple : `(evaluate-arithm-expr '(* 3 (* 4 b)))` retourne `60` si `b` vaut `5`.

   On pourra si nécessaire s'aider de fonctions auxiliaires.