Programmation 3 : feuille 4

Listes

Exercice 4 .1

Tester les fonctions de base sur les listes : `cons, car, cdr, list, append, reverse, ....`

Les 4 exercices suivants sont tirés du livre de David Touretzky "COMMON LISP : A gentle introduction to Symbolic Computation". Dans les 3 premiers, on demande d'écrire des fonctions ***non destructives****; les guillemets utilisés dans les énoncés signalent au lecteur un évident abus de langage, consistant à spécifier une fonction "pure" dans un style impératif.*

Exercice 4 .2

1. Écrire une fonction `swap-first-last(l)` qui "échange" le premier et le dernier élément d'une liste.
2. La tester avec la liste `'(YOU CANT BUY LOVE)`.

Exercice 4 .3

1. Écrire une fonction `rotate-left(l)` qui "fait" une rotation circulaire vers la gauche des éléments d'une liste.
2. La tester.

Exercice 4 .4

1. Écrire une fonction `rotate-right(l)` qui "fait" une rotation circulaire vers la droite des éléments d'une liste.
2. La tester.

Exercice 4 .5

Un objet est décrit par une liste de propriétés. Les propriétés d'un couple d'objets sont regroupées dans une seule liste séparée en son milieu par le symbols `-vs-`.

```
(defparameter *prop*
  '(large red shiny cube -vs- small shiny red four-sided pyramid))
```

1. Écrire une fonction `right-side` qui donne la liste des propriétés du deuxième objet (à droite de `-vs-`).
2. Écrire une fonction `left-side(l)` qui donne la liste des propriétés du premier objet (à gauche de `-vs-`).
3. Écrire une fonction compare qui retourne le nombre de propriétés communes aux deux objets sous forme d'une liste `(x PROPRIETES COMMUNES)`.
   Exemple : `(compare *prop*)` retourne `(2 PROPRIETES COMMUNES)`.

Exercice 4 .6

1. Écrire une fonction `my-length (l)`, version naïve récursive de la fonction prédéfinie `length`.
2. Tester les limites de la version naïve, en temps (avec `time`) et en espace (`[Condition of type SB-KERNEL::CONTROL-STACK EXHAUSTED]`) avec des listes de plus en plus grandes fabriquées avec `make-list`.

Exercice 4 .7

1. Écrire une fonction récursive `randomize-list (l n)` où `l` est une liste quelconque, `n` $\in \mathbb{N}^*$, et qui retourne une liste de la même longueur que `l` dont les éléments sont des entiers dans $[0, \mathtt{n}[$. Exemples :

```
* (randomize-list '(hello this is a list) 100)
(33 83 2 71 18)

* (randomize-list '(hello this is a list) 100)
(12 19 53 35 22)

* (randomize-list '(hello this is a list) 2)
(1 0 0 1 1)
```

2. À l'aide de la fonction `randomize-list` (et de `make-list` de Common Lisp), écrire une fonction `random-list (length n)` qui retourne une liste de longueur `length` composée d'entiers aléatoires pris dans $[0, \mathtt{n}[$.
3. Écrire une fonction booléenne `test-randomize-list-once (m n)` où `m` $\in \mathbb{N}$, `n` $\in \mathbb{N}^*$. La fonction doit
   (a) appeler `random-list`,
   (b) vérifier que la liste l retournée par `random-list` est correcte c'est à dire que sa longueur est `m` et que chacun de ses éléments est dans $[0, \mathtt{n}[$,
   (c) provoquer une erreur si cette vérification échoue.