ENSEIRB-Matmeca
2019-2020

1ère année - Filière Info
PG104

# TD n°5 - Listes et récursivité

## Exercice 1: Grammaires

On rappelle qu'une expression symbolique peut se définir de la manière suivante :

$$\text{s-expr} ::= \text{atome} \mid \text{paire-pointée}$$

En Scheme, une liste peut se définir par :

$$\text{liste} ::= \text{nil} \mid (\text{s-expr} \ . \ \text{liste})$$

Ici, nil représente la liste vide `'()`. On décide d'appeler liste-sans-point les listes qui s'afficheront sans point à l'écran.

1. Que représente la définition suivante :

   2. liste-étendue ::= nil | paire-pointée

   Quelle différence y a t'il avec les listes de Scheme ?
3. Donner en utilisant le même langage que ci-dessus une définition de liste-sans-point
4. Écrire un prédicat `list?`.
5. Écrire un prédicat `dotless-list?`.

## Exercice 2: Listes d'entiers

1. Écrire la fonction `iota`[1] qui étant donné un entier $n$ produit la liste `(0 1 2 ... n-1 n)`.
2. Écrire une fonction `scalar-product` qui, étant données deux listes (de même taille) passées en paramètres, retourne le produit scalaire des vecteurs qu'elles représentent.
3. Écrire la fonction `divisors` qui étant donné un entier $n$ produit la liste de tous ses diviseurs

## Exercice 3: Aplatissement

1. Écrire une fonction `list-flatten` qui renvoie la liste (plate) des atomes contenus dans une expression symbolique (à l'exception des listes vides `'()`).
   *Exemple :* `(list-flatten '((1) (2 3) 4)) ; → (1 2 3 4)`
2. Pour aller plus loin... Écrire une fonction `list-flatten-tr` qui soit récursive terminale.

## Exercice 4: En attendant `map` et `apply` ...

1. La fonction `iota` est originellement une fonction standard du langage APL, mais elle apparaît aussi en C++ comme `std::iota`. Une telle fonction peut être utilisée, nous le verrons plus tard, à la place d'algorithmes utilisant des boucles for.

1. Écrire une fonction `sum-list` qui, étant donnée une liste de nombres, calcule leur somme. On pourra envisager une version `sum-list*` qui calcule récursivement cette somme dans les sous-listes. Par exemple : `(sum-list* '(10 (5 2)))` donne 17.
2. Écrire une fonction `reverse-list` qui renverse l'ordre des éléments de premier niveau d'une liste plate. Ecrire ensuite la fonction `reverse-list*` qui renverse l'ordre des éléments à tous les niveaux de la liste :

   *Exemple :*

   3. `(reverse-list '(a b (c d) e f))` ⇒ `(f e (c d) b a)`
   4. `(reverse-list* '(a b (c d) e f))` ⇒ `(f e (d c) b a)`
5. Écrire une fonction `count-list` qui prend en argument un atome et une liste et calcule le nombre d'occurrences de l'atome dans la liste

   *Exemple :* `(count-list 'a '(a (b a (c a)) d a))` ⇒ 4

### Exercice 5: Bégaiement

1. Écrire la fonction `stutter` prenant une liste de symboles en argument, une phrase, et retournant en sortie une phrase où tous les mots sont répétés.

   *Exemple :* `(stutter '(hasta la vista))` ⇒ `(hasta hasta la la vista vista)`
2. Écrire une fonction `unstutter` qui ôte d'une phrase tout bégaiement et notamment celui produit par la fonction de l'exercice précédent.

   *Exemple :* `(unstutter (stutter '(hasta la vista)))` ⇒ `(hasta la vista)`

### Pour aller plus loin... Exercice 6: The count is good

Écrire une fonction qui étant donnée une liste de nombres $l$, et un nombre objectif $x$, calcule si $x$ peut être atteint en construisant une expression arithmétique constituée uniquement d'additions et de soustractions et des valeurs contenues dans la liste $l$.

Modifier la fonction de manière à pouvoir passer la liste des opérations en paramètre (on ne considérera que des opérations binaires).