INFO0054 - Programmation fonctionnelle

Répétition 3: Récursion sur les listes

Jean-Michel Begon

25 Février 2015

Récursion sur les listes

Exercice 1.
Spécifier la fonction suivante :

```
(define xyz
    (lambda (m n)
        (if (zero? m) 0
            (if (zero? (modulo n m))
                (+ m (xyz (- m 1) n))
                (xyz (- m 1) n)))))
```

Exercice 2.
Les nombres de Gribomont sont définis comme suit :

si $n < 3$, $f(n) = n$
si $n \geq 3$, $f(n) = f(n-1) + f(n-2) + f(n-3)$.

Écrire une fonction qui permet de calculer les nombres de Gribomont.

Exercice 3.
Écrire une fonction `big` qui prend comme arguments un nombre entier `n` et une liste `l` de nombres entiers, telle que `(big n l)` est la liste des éléments de `l` plus grands que `n`.

`(big 5 '(7 -3 6 1 -2 5 6))` ⇒ `(7 6 6)`

Exercice 4.
Écrire une fonction `removeAll` à deux arguments `l` et `n` dont les valeurs sont respectivement une liste et un nombre entier, qui retourne la liste privée de toutes les occurrences de `n`.

`(removeAll '(2 7 1 7 3 1) 1)` ⇒ `(2 7 7 3)`

Exercice 5.
Que faudrait-il changer dans la fonction `removeAll` pour qu'elle renvoie la liste privée de la première occurrence de `n` ?

Exercice 6.

Définir la fonction `count` à deux arguments qui renvoie le nombre de fois que l'on a rencontré le premier argument dans la liste en second argument.

Exercice 7.
Définir la fonction `make-list` qui prend comme arguments un nombre `n` et une expression quelconque `x` et qui construit une liste composée de `n` fois l'élément `x`.

Exercice 8.
Définir la fonction `filter_` à deux arguments, un prédicat unaire `p?` et une liste d'éléments appartenant à son domaine `ls`, et renvoyant la liste des éléments de `ls` pour lesquels l'application du prédicat est `#t`.
Remarque : la fonction `filter` est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

Exercice 9.
Redéfinir les fonctions `big`, `removeAll` et `count` à l'aide de la fonction `filter`.

Exercice 10.
Définir la fonction `suffix` à deux arguments `l1` et `l2` dont les valeurs sont des listes et qui retourne `#t` si `l1` est suffixe de `l2` et `#f` sinon.

`(suffix '(1 2) '(3 x 1 2))` $\Rightarrow$ `#t`
`(suffix '(1 2) '(3 x 2))` $\Rightarrow$ `#f`

Exercice 11.
Soit la fonction f définie sur $\mathbb{N}$ par :

$$f(n) = \left(\sum_{i=0}^{n-1} ([2 + f(i)] \times [3 + f(n - i - 1)]) \right) \mod (2n + 3)$$

Implémentée par le programme suivant :

```
(define f (lambda (n) (car (fx n 0 '()))))
(define fx
  (lambda (k i u)
    (if (zero? k) u
        (let ((next
               (modulo (apply + (map (lambda (x y) (* (+ 2 x) (+ 3 y)))
                                     u
                                     (reverse u)))
                       (+ i i 3))))
          (fx (- k 1) (+ i 1) (cons next u))))))
```

Spécifier la fonction `fx`.

Exercice 12.
Écrire une fonction `frequency` prenant en argument une liste d'atomes `ls` et renvoyant une table d'apparition de chacun des atomes dans la liste `ls`. Cette table sera représentée par une liste de paires pointées, dont le car est un atome et le cdr la fréquence d'apparition de cet atome. Tous les

atomes de `ls` apparaissent une et une seule fois dans la table.

```
(frequency '(a b c b a b d a c b b))
   ⇒ ((a . 3) (b . 5) (c . 2) (d . 1))
```

Exercice 13.
Implémenter le quicksort, spécifier les fonctions auxiliaires et discuter sa complexité.

Exercice 14.
Implémenter le tri par fusion, spécifier les fonctions auxiliaires et discuter sa complexité.