Programmation applicative – L2
TD 6 : Récursivité terminale

G.Artignan {artignan@lirmm.fr} M.Lafourcade {lafourcade@lirmm.fr}
S.Daudé {sylvain.daude@univ-montp2.fr} A.Chateau {chateau@lirmm.fr}
B.Paiva Lima da Silva {bplsilva@gmail.com} C.Dony {dony@lirmm.fr}

1 Notion de récursivité enveloppée et récursivité terminale

1.1 Récursivité enveloppée

La fonction factorielle $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$ et $0! = 1$

```
(define (fact n)
    (if (= n 0) 1                ⟵ condition d'arrêt
        (* n (fact (- n 1)))))   ⟵ appel récursif
```

Exercice 1 *Écrire la séquence d'expansion de* `(fact 5)`. *Commenter.*

L'appel récursif `(fact (- n 1))` est dit "**enveloppé**". C'est à dire qu'il fait partie d'un autre calcul, ici la multiplication par `n`.

1.2 La notion d'accumulateur

Il existe un type d'appel récursif qui n'est pas enveloppé. Grâce à un accumulateur on peut sortir l'appel récursif du calcul qui l'enveloppe. L'accumulateur sert à mémoriser le résultat des calculs intermédiaires au cours des différents appels. La fonction `fact` peut être redéfinie à l'aide d'un accumulateur de cette manière :

```
(define (fact-acc n acc)
   (if (= 0 n) acc
       (fact-acc (- n 1) (* acc n))))

(define (fact-iter n) (fact-acc n 1))
```

Ce type d'appel récursif est appelé "**terminal**", car l'appel récursif n'est plus imbriqué dans aucun calcul. On parle également de **forme itérative**.

Exercice 2 *Écrire la séquence d'expansion de* `(fact-acc 5 1)`. *Comparer à l'exercice précédent et commenter.*

Exercice 3 *Soit la fonction* `sum-squares-1-n` *qui associe à un entier n la somme des carrés de* 1 *à n :*

$$\texttt{sum-squares-1-n}(n) = 1 + 2^2 + 3^2 + \ldots + n^2$$

1. *Écrire la fonction* `sum-squares-1-n` *en récursivité enveloppée.*
2. *Écrire la fonction* `sum-squares-1-n-iter` *en récursivité terminale.*

Exercice 4 *Inversion d'une liste*

1. *Définir la fonction récursive naïve qui inverse les éléments d'une liste.*
2. *Définir la fonction récursive terminale qui inverse les éléments d'une liste.*

2 Fonctions basiques sur les listes

Exercice 5 *Écrire une fonction* `sommeliste` *qui prend en paramètre une liste d'entiers et qui rend comme résultat la somme des entiers de la liste. En donner une version en récursivité terminale.*

Exercice 6 *Écrire une fonction* `sommerangimpair` *qui prend en paramètre une liste d'entiers et qui fait la somme des entiers à la première, la troisième, la cinquième, ... place dans la liste. Exemple :*

```
> (sommerangimpair  ' ( 3  5  7  4  6  5  4  2 ) )
> 20      ; car  20 =  3 + 7 + 6 + 4
```

En donner une version en récursivité terminale.

3 Fonctions de parcours de listes

Exercice 7 *Donner la fonction* `appartient?` *qui prend un élément et une liste comme arguments, qui retourne true si l'élément appartient à la liste, et false autrement. Exemples :*

```
> (appartient?  'b  '( a  b  c  d ))
> #t
> (appartient?  'e  '( a  b  c  d ))
> #f
```

Exercice 8 *Écrire une fonction* `debut` *qui prend une liste et un entier* `k` *en paramètre, et qui rend comme résultat la liste composée des* `k` *premiers éléments de la liste. Dans le cas où la liste initiale possède moins de* `k` *éléments, le résultat sera la liste ellemême. Exemples :*

```
> (debut   ' ( 3  5  7  6  4  2 )  4 )
> ( 3  5  7  6 )
> (debut   ' ( 3  5  7  6  4  2 )  8 )
> ( 3  5  7  6  4  2 )
```