IA01 : Rapport du TP1

Auteurs :
Camille Gerin-Roze
Thomas Perrin

Le 2 Octobre 2015

Table des matières

1 Exercice 1 **2**
1.1 Donner les premiers N éléments 2
1.1.1 Version récursive 2
1.1.2 Version itérative 2
1.2 Intersection de 2 listes 3
1.2.1 Version récursive 3
1.2.2 Version itérative 3

Chapitre 1

Exercice 1

1.1 Donner les premiers N éléments

1.1.1 Version récursive

```
(defun firstn (n l)
(if (> n 0) (append (list (car l)) (firstn (- n 1) (cdr l)))))
```

La fonction prend deux arguments :
— Le nombre de **n** d'éléments à prélever dans la liste
— La liste **l** dans laquelle nous allons prélever les éléments
On prend le premier argument de la liste l, et on construit une nouvelle liste avec un appel récursif de la fonction avec **n-1** éléments, jusqu'à ce que n soit égale à 0.

1.1.2 Version itérative

```
(defun firstn-ite (n l)
  (let ((listeEntiere l))
    (setq liste ())
  (dotimes (x n liste)
    (setq liste (append liste (list (car listeEntiere))))
    (setq listeEntiere (cdr listeEntiere))
    )))))
```

La fonction itérative se déroule différement. On crée une nouvelle liste dans laquelle on va copier les n premiers éléments. On doit donc utiliser deux listes différentes. La fonction dans son format itératif utilise donc plus de mémoire.

1.2 Intersection de 2 listes

1.2.1 Version récursive

```
(defun inter (l1 l2)
  (if (not (null (car l1)))
      (if (member (car l1) l2)
         (append (list (car l1)) (inter (cdr l1) l2)) ;;; Si
        (inter (cdr l1) l2)) )) ;;; Sinon
```

La fonction prend en paramètre deux listes (l1 et l2). Les appels récursif continue jusqu'à ce que le car de la première liste (celle sur laquelle nous allons itérer) Si le car de l1 est un membre de la liste l2, alors on va concaténer la liste contenant le car de l1 et le résultat de l'appel récursif sur le cdr de l1 et l2. Sinon on fera juste l'appel récursif.

1.2.2 Version itérative

```
(defun inter-iteratif (l1 l2)
    (mapcan #'(lambda (x) (if (member x l2) (list x))) l1))
```

La version itérative est plus simple car en utilisant mapcan (qui renvoie une nouvelle liste) on peut tout simplement itérer sur la première liste et vérifier que chaque élément et dans la deuxième. Si l'élément est dans la liste, alors on retourne l'élément, sinon on retourne nil. Grâce au mapcan qui utilise **cons** pour concaténer les résultats, les nil ne seront pas membre de la liste retourner par la fonction.