

RAPPORT TP N°2 IA01

Damien MARIÉ et Antoine POUILLAUDE

4 novembre 2013

Table des matières

1	Les cannibales et les missionnaires	3
1.1	Question n°1	3
1.2	Question n°2	3
1.3	Retourner la concaténation de deux listes	4
1.4	Intersection de deux listes	5
1.5	Inclusion de deux listes	5
2	Exercice 2	8
3	Exercice 3	9
3.1	Algorithmique	9
3.2	Code source	9
4	Exercice 4	10
4.1	Petites fonctions pour commencer	10
4.2	Personnes contenues dans la base de données.	11
4.3	Les personnes qui s'appellent Pierrot.	11
4.4	Les personnes avec le nom donné en argument.	12
4.5	Les personnes avec X années.	12
4.6	Les personnes ayant moins de X livres.	13
4.7	Moyenne des ages de la famille X.	13

Remarques générales

Chapitre 1

Les cannibales et les missionnaires

1.1 Question n°1

Suivons le formalisme donné dans l'énoncé. Un état est ainsi représenté : (État rive gauche, État rive droite) sachant qu'un état d'une rive est donné par (nombre de Missionnaire, nombre de Cannibale, 1 si le bateau est amarré à la rive 0 sinon). Ainsi nous obtenons l'ensemble des états :

$$\text{Ensemble d'état} = \{((3\ 3\ 1)(0\ 0\ 0)), ((2\ 2\ 0)(1\ 1\ 1)), ((3\ 2\ 1)(0\ 1\ 0)), ((0\ 1\ 0)(3\ 2\ 1)), \dots\}$$

Cela dit on remarque très vite que la notation est lourde et futile puisque dans tous les cas on doit avoir :

$$\left\{ \begin{array}{l} \sum_{\{Rivegauche, Rivedroite\}} Cannibale = 3 \\ et \\ \sum_{\{Rivegauche, Rivedroite\}} Missionnaire = 3 \end{array} \right. \quad \forall \text{ etat} \in \text{Ensemble d'état}$$

Alors on pourrait simplifier la liste des états. Du moment que l'on connaît l'état d'une on peut alors connaître l'état de l'autre rive par déduction. Par conséquent, nous simplifierons en affichant l'état que d'un seul côté. Ainsi on aura l'ensemble des états suivant :

$$\text{Ensemble d'état} = \{(3\ 3\ 1), (2\ 2\ 0), (3\ 2\ 1), (3\ 2\ 0), (0\ 1\ 0), (2\ 3\ 1), (2\ 3\ 0), (1\ 1\ 1), \dots\}$$

1.2 Question n°2

Listing 1.1– F2 (N L) version recursive

```
(defun F2rec (N L)
  (if (> N 0)
      (cons (first L) (F2 (- N 1) (rest L)))
      ()))
```

Listing 1.2– F2 (N L) version itérative

```
(defun F2Iter (N L)
  (if (< N (length L))
      (let ((out NIL))
        (dotimes
          (x (+ N 1)
```

```

                                out)
                                (cons out (nth x L))
                                )
                                )
                                )
                                (print "The_number_you_chose_is_not_correct")
                                )
                                )
                                )

```

F2rec et F2iter retourne la tête de la liste L formée par les N premiers éléments de la liste L.

1.3 Retourner la concaténation de deux listes

Listing 1.3– F3 (L M) version recursive

```

(defun F3rec (L M)
  (if (null L) M
      (F3rec (rest L) (cons (first L) M))
  )
)

```

Listing 1.4– F3 (L M) version itérative

```

(defun F3iter (L M)
  (append L M)
)

```

F3rec et F3iter retournent la concaténation des deux listes L et M.

1.4 Intersection de deux listes

Algorithm 1 Algorithme pour l'intersection de deux listes

```
if  $L = NIL$  then
    retourner  $T$ 
else if  $L[1] \in M$  then
    retourner  $(L[1]F4(L/L[1]))$ 
else
    retourner  $F4(L L[1])$ 
end if
```

Listing 1.5– F4 (L M) version recursive

```
(defun F4rec (L M)
  (if (NULL L) L
      (if (member (first L) M)
          (cons (first L) (F4rec (rest L) M))
          (F4rec (rest L) M))
      ))
)
```

Listing 1.6– F4 (L M) version it  rative

```
(defun F4iter (L M)
  (let ((out '()))
    (if (< (length L) (length M))
        (dolist (x L out)
          (if (member x M)
              (cons out x)
              ))
        (dolist (x M out)
          (if (member x L)
              (cons out x)
              ))
        ))
  )
)
```

F4rec et F4iter retournent l'intersection des deux listes. Attention, cependant, apr  s test la fonction F4iter ne retourne pas ce que l'on souhaite.

1.5 Inclusion de deux listes

La premi  re fonction a   t   produite avant la r  ception des sp  cifications que Mme Abel nous a envoy   en plus. Elle v  rifie simplement que tous les   l  ments de la premi  re liste sont dans la seconde, et ce, ind  pendamment de l'ordre dans lequel les   l  ments sont.

Algorithm 2 Algorithme pour l'inclusion de deux listes version 1

```
if  $L = NIL$  then
  retourner  $T$ 
else if  $L[1] \in M$  then
  retourner  $F4(L\ L[1])$ 
else
  retourner  $NIL$ 
end if
```

Listing 1.7– F5 (N L) version recursive 1

```
(defun F5 (L M)
  (if (NULL L) T
      (if (member (first L) M) (F5 (rest L) M)
          NIL)
      )
  )
)
```

Cet seconde fonction tient, cette fois-ci, compte de l'ordre des éléments de la première liste. Par exemple si la liste L est définie comme L=(1 2) et M comme M=(1 3 5 2) alors la fonction retournera NIL alors qu'avec M=(1 2 5 3) elle renverra T. La composition de ces deux fonctions, en revanche, nous donne une complexité plutôt très mauvaise puisque dans le pire des cas on a une complexité en $O(card(L) * card(M))$ c'est à dire, en simplifiant, $O(n^2)$.

Algorithm 3 Algorithme pour l'inclusion de deux listes version 2

```
if  $L = NIL$  then
  retourner  $T$ 
else if  $M = NIL$  then
  retourner  $T$ 
else if  $verifF5(L, M)$  then
  retourner  $T$ 
else
  retourner  $NIL$ 
end if
```

Listing 1.8– F5 (L M) version récursive 2

```
(defun F5 (L M)
  (cond
    ((NULL L) T)
    ((NULL M) NIL)
    ((verifF5 L M) T)
    (T (F5 L (rest M)))
  )
)
```

La fonction suivante vérifie les termes consécutifs.

Algorithm 4 Algorithme de vérification

```
if  $L = NIL$  then
  retourner  $T$ 
else if  $M = NIL$  then
  retourner  $T$ 
else if  $L[1] = M[1]$  then
  retourner  $verifF5(rest(L), rest(M))$ 
else
  retourner  $NIL$ 
end if
```

Listing 1.9– verifF5 (L M) version récursive 2

```
(defun verifF5 (L M)
  (cond
    ((NULL L) T)
    ((NULL M) NIL)
    ((eq (first L) (first M)) (verifF5 (rest L) (rest M)))
    (T NIL)
  )
)
```


Chapitre 2

Éxercice 2

Cet exercice consistant purement à la compréhension de la fonction `mapcar` dans un exemple plutôt simple, il n'y pas de commentaires dessus.

Listing 2.1– `list-carres (L)` avec fonction `lambda`

```
(defun list-carres (L)
  (mapcar (lambda (x)(* x x)) L)
)
```

Chapitre 3

Exercice 3

My-assoc est une fonction qui recherche si la clé donner en argument se trouve dans la liste.

3.1 Algorithmique

Ci-dessous se trouve l’algorithme fait en pseudo-code de la fonction.

Algorithm 5 Algorithme de My-Assoc

```
if cle! = cledeL[1] then
    retourner my - assoc(cle, rest(L))
else
    retourner la paire
end if
```

3.2 Code source

Et voici le code source de ladite fonction.

Listing 3.1– My_Assoc (cle a-liste) version récursive

```
(defun my-assoc (cle a-liste)
  (if (null a-liste)
      NIL
      (if (equal (first (first a-liste)) cle)
          (first a-liste)
          (my-assoc cle (rest a-liste)))
  )
)
```

Chapitre 4

Exercice 4

4.1 Petites fonctions pour commencer

Pour retourner le nom il suffit de logiquement renvoyer le premier élément de la liste.

Listing 4.1– nom(Personne)

```
(defun nom (pers) (first pers))
```

Pour retourner le prénom il suffit de logiquement renvoyer le second élément de la liste.

Listing 4.2– prenom(Personne)

```
(defun prenom (pers) (cadr pers))
```

Pour retourner la ville il suffit de logiquement renvoyer le troisième élément de la liste.

Listing 4.3– ville(Personne)

```
(defun ville (pers) (caddr pers))
```

Pour retourner l'âge il suffit de logiquement renvoyer l'avant avant dernier élément de la liste.

Listing 4.4– age(Personne)

```
(defun age (pers) (caddr pers))
```

Pour retourner le nombre de livre il suffit de logiquement retourner le dernier élément de la liste.

Listing 4.5– nom(Personne)

```
(defun nom (pers) (first pers))
```

Pour retourner le nom il suffit de logiquement renvoyer le premier élément de la liste.

Listing 4.6– nombre-livre(Personne)

```
(defun nombre-livre (pers) (car (last pers)))
```

Pour les questions les premières fonctions sont triviales cependant la dernière fonction (FB6) à été difficilement réalisable en version récursive de par la nature même de la question (en récursif pur, cela reviendrait à faire une moyenne progressive).

4.2 Personnes contenues dans la base de données.

Je vais tout simplement retourner toute la base données.

Listing 4.7– FB1(data)

```
(defun FB1 (data)
  (if (null (rest data))
      data
      (cons (first data) (FB1 (rest data)))
  )
)
```

4.3 Les personnes qui s'appellent Pierrot.

Ci-dessous se trouve l'algorithme fait en pseudo-code de la fonction.

Algorithm 6 Algorithme de Pierrot

```
if data = NIL then
  retourner data
else if nom(data[1]) = "Pierrot" then
  retourner (data[1]FB2(rest(data)))
else
  retourner FB2(rest(data))
end if
```

Listing 4.8– FB2(data)

```
(defun FB2 (data)
  (if (null data)
      data
      (if (eq (nom (first data)) 'Pierrot)
          (cons (first data) (FB2 (rest data)))
          (FB2 (rest data))
      )
  )
)
```

4.4 Les personnes avec le nom donné en argument.

Ci-dessous se trouve l'algorithme fait en pseudo-code de la fonction.

Algorithm 7 Algorithme de nom

```
if data = NIL then
  retourner data
else if nom(data[1]) = name then
  retourner (data[1]FB2(rest(data), name))
else
  retourner FB2(rest(data), name)
end if
```

Listing 4.9– FB3(data name)

```
(defun FB3 (data name)
  (if (null data)
      data
      (if (eq (nom (first data)) name)
          (cons (first data) (FB2 (rest data) name))
          (FB2 (rest data) name)
      )
  )
)
```

4.5 Les personnes avec X années.

Ci-dessous se trouve l'algorithme fait en pseudo-code de la fonction.

Algorithm 8 Algorithme de l'âge

```
if data = NIL then
  retourner NIL
else if age(data[1]) = X then
  retourner data[1]
else
  retourner FB4(rest(data))
end if
```

Listing 4.10– FB4(data X)

```
(defun FB4 (data x)
  (if (null data)
      NIL
      (if (= (age (first data)) x)
          (first data)
          (FB4 (rest data) x)
      )
  )
)
```

4.6 Les personnes ayant moins de X livres.

Ci-dessous se trouve l'algorithme fait en pseudo-code de la fonction.

Algorithm 9 Algorithme de l'âge

```
if data = NIL then  
    retourner NIL  
else if nombre – livre(data[1]) ≤ X then  
    retourner data[1]  
else  
    retourner FB5(rest(data))  
end if
```

Listing 4.11– FB5(data X)

```
(defun FB5 (data x)  
  (if (null data)  
      NIL  
      (if (< (nombre–livre (first data)) x)  
          (first data)  
          (FB5 (rest data) x)  
      )  
  )  
)
```

4.7 Moyenne des ages de la famille X.

Listing 4.12– FB6(data X)

```
(defun FB6 (data x) (  
  (setq c 0)  
  (setq sum 0)  
  (dolist (el data) (  
    (if (eq (name el) x) (  
      (setq sum (+ sum (age el)))  
      (setq c (+ c 1))  
    ))  
  ))  
  (if (eq c 0) 0 (  
    (/ sum c)  
  ))  
)
```