Programmation Fonctionnelle : TD4

Université de Tours

`Département informatique de Blois`

Tris de listes

*
* *

Appropriation du cours

1. On reprend le code suivant qui permet la réalisation du tri par insertion sur des listes d'entiers.

```
let rec insere x l = match l with
  [] -> [x]
| h::t -> if (h > x) then x::l
          else h::(insere x t);;

 let rec tri_insertion l = match l with
  [] -> []
| [x] -> [x]
| h::t -> insere h (tri_insertion t);;
```

   Exécuter cet algorithme pour la liste $l = [0; -1; 1; -2; 2; -3; 3]$ et $l' = [13; 11; 7; 5; 3; 2]$.

   Suivez l'exécution à l'aide de la commande `#trace`.

2. On reprend le code suivant qui permet la réalisation du tri par insertion sur des listes d'entiers.

```
let rec separe l = match l with
  [] -> ([], [])
| [x] -> ([x], [])
| a::b::c -> let (l1, l2) = separe c in
                 (a::l1, b::l2);;

let rec fusionne l1 l2 = match (l1, l2) with
 ([], l2) -> l2
|(l1, []) -> l1
|(a::b, h::t) -> if (a <= b) then a::(fusionne b l2)
                 else h::(fusionne l1 t);;

let rec tri_fusion l = match l with
  [] -> []
| [x] -> [x]
| h::t -> let (l1,l2) = (separe l) in
              fusionne (tri_fusion l1) (tri_fusion l2);;
```

   Exécuter cet algorithme sur les précédents exemples et avec la commande `#trace`.

Problème 1

1. Écrire la spécification et l'algorithme d'une fonction `partitionne l p` qui prend en paramètre une liste d'entiers l et un entier p et retourne pour résultat un couple de listes (l_1, l_2) tel que $l = l_1 \cup l_2$ où :

$$l_1 = \bigcup_{e \in l | e \leq p} e \quad \text{et} \quad l_2 = \bigcup_{e \in l | e > p} e = l \backslash l_1$$

Plus simplement, l_1 est la liste avec tous les éléments e de l plus petits ou égaux à p, et l_2 , la liste de tous les éléments $e$de l strictement plus grands que p.

$$partitionne : \begin{cases} \texttt{List} < \texttt{int} >, \texttt{int} & \rightarrow \texttt{List} < \texttt{int} > \times \texttt{List} < \texttt{int} > \\ l, p & \mapsto (l_1, l_2) \end{cases}$$

avec l_1 et l_2 définies ci-dessus.

```
let rec partitionne l p = match l with
  [] -> ([], [])
| h::t -> let (l1,l2) = (partitionne t p) in
               if(h <= p) then (h::l1, l2) else (l1, h::l2);;
```

2. Écrire la fonction `tri_rapide l` du cours qui prend une liste l et la tri selon la relation $\leq$.

```
let rec tri_rapide l = match l with
  [] -> []
| h::t -> let (l1, l2) = partitionne t h in
              (tri_rapide l1)@[h]@(tri_rapide l2);;
```

Problème 2

1. Écrire la spécification et l'algorithme d'une fonction `extract_min l` qui prend en paramètre d'entrée une liste d'entiers l et retourne pour résultat un couple (min, l') où :

$$min = \min_{x \in l}\{x\} \quad \text{et} \quad l' = l \backslash \{min\}$$

Plus simplement, min est le plus petit élément de l et l' est la liste l sans l'élément min.

On pourra créer plusieurs fonctions intermédiaires afin de simplifier l'algorithme principal.

$$extract\_min : \begin{cases} \texttt{List} < \texttt{int} > & \rightarrow \texttt{List} < \texttt{int} >, \texttt{int} \\ l & \mapsto (min, l') \end{cases}$$

avec min et l' définis ci-dessus.

```
(* Calcul du minimum d'une liste min = min_{x∈l}{x}
   List<int> -> int *)
let rec min_liste l = match l with
  [] -> failwith "Liste vide"
| [x] -> x
| h::t -> min h (min_liste t);;
```

```
(* Liste l sans l'élément n  l' = l\{n}
   List<int> -> List<int> *)
let rec retire l n =
  match l with
  [] -> []
| h::t -> if(h = n) then t else h::(retire t n);;

let extract_min l =
  let m = min_liste l in (m, retire l m);;
```

2. Écrire la fonction `tri_selection l` du cours qui prend une liste l et la tri selon la relation $\leq$.

```
let rec tri_selection l = match l with
  [] -> []
| h::t -> let (m, _l) = extract_min l in
                m::(tri_selection _l);;
```