Algorithmique
Correction Contrôle n° 1

Info-sup S1 – Epita

Solution 1 **(Type abstrait : liste itérative (modifier) – *2 points*)**

1. On peut mettre la précondition suivante, ce qui "allègera" les axiomes. *Ne pas la préciser ici n'est pas grave si le postulat existe au niveau des axiomes.*

   PRÉCONDITIONS

   modifier(l, i, e) **est-défini-ssi** $l \neq$ *liste-vide* & $1 \leqslant$ i $\leqslant$ *longueur(l)*

2. Les axiomes sont les suivants :

   AXIOMES

   longueur(modifier(l, i, e)) = *longueur(l)*

   $k = i \Rightarrow$ *ième(modifier(l, i, e), k)* = *e*

   $1 \leqslant k \leqslant$ *longueur(l)* & $k \neq i \Rightarrow$ *ième(modifier(l, i, e), k)* = *ième(l, k)*

   AVEC

liste	l
entier	i, k
élément	e

Solution 2 **(Un peu de cours... – *4 points*)**

1. Un observateur retourne un type prédéfini.
2. Une opération qui n'est pas définie partout est une opération partielle.
3. Les problèmes qui se posent lors de la conception de l'ensemble des axiomes sont la complétude et la consistance.
4. Les zones qui ne composent pas la signature d'un type abstrait sont les zones PRECONDITIONS, AXIOMES et AVEC.
5. Nous écrivons des axiomes en appliquant des observateurs aux opérations internes.

***Solution 2* (Suppression intervalle – *6 points*)**

Spécifications : La fonction `remove_interval` *list i j* supprime toutes les valeurs comprises entre les places i et j dans la liste *list*.

```
let remove_interval list i j =
  if i <= 0 || j <= 0 || j < i then
    invalid_arg "remove_interval: invalid ranks"
  else
    let rec remove = function    (* (list, n), n the current position *)
    ([], n) -> if n <= j then failwith "remove_internal: list too short"
                         else []
      | (list, n) when n > j -> list
      | (e::q, n) when n < i -> e :: remove (q, n+1)
      | (e::q, n) (* i in [x, y] *) -> remove (q, n+1)
    in
      remove (list, 1) ;;

let remove_interval list i j =
  if i <= 0 || j <= 0 || i > j then
    invalid_arg "remove_internal: invalid ranks"
  else
    let rec aux = function
       (1,1,e::l) -> l
     | (_,_,[])   -> failwith "remove_internal: list too short"
     | (1,j,_::l) -> aux (1,j-1,l)
     | (i,j,e::l) -> e :: aux (i-1,j-1,l)
    in aux (i,j,list)
;;

(* Evaluation *)
val remove_interval : 'a list -> int -> int -> 'a list = <fun>
```

***Solution 3* (Listes alternées – *5 points*)**

Spécifications : La fonction `alternate_list` $l1$ $l2$ construit la liste alternée de $l1$ et $l2$.

```
let alternate_list list1 list2 =
  let rec aux = function
     ([],l2)          -> aux (list1,l2)
   | (l1,[])          -> []
   | (e1::l1,e2::l2) -> e1 :: e2 :: aux (l1,l2)
  in
    match list1 with
        []    -> list2
      | list1 -> aux (list1,list2)
;;

(* Evaluation *)
val alternate_list : 'a list -> 'a list -> 'a list = <fun>
```

Solution 4 **(Mystery –** ***4 points*****)**

```
let aux p e (r1, r2) =
  if p e then (e::r1, r2) else (r1, e::r2);;
(*Evaluation:*)
val aux : ('a -> bool) -> 'a -> 'a list * 'a list -> 'a list * 'a list =
<fun>

aux (function x -> x mod 2 = 0) 10 ([0; 2], [1; 3]);;
(*Evaluation:*)
- : int list * int list = ([10; 0; 2], [1; 3])

let rec mystery p = function
  ([], [])                    -> ([], [])
| (e1::l1, []) | ([], e1::l1) -> aux p e1 (mystery p (l1, []))
| (e1::l1, e2::l2)            -> aux p e1 (aux p e2 (mystery p (l1, l2)));;
(*Evaluation:*)
val mystery : ('a -> bool) -> 'a list * 'a list -> 'a list * 'a list = <fun>

let f x y = x - y = 0 in mystery f ([1; 2; 3; 4; 5; 6], [1; 2; 3; 4; 5; 6])
;;
(*Error*)

mystery (function x -> x mod 2 = 0) ([1; 2; 3; 4; 5; 6; 7; 8], [10; 11; 12;
13; 14]);;
(*Evaluation:*)
- : int list * int list = ([10; 2; 12; 4; 14; 6; 8], [1; 11; 3; 13; 5; 7])
```