Algorithmique
Correction Contrôle n° 1

Info-sup S1 – Epita

Solution 1 **(Une pincée de listes... – *5 points*)**

1. Opérations *tête*, *premier*, *fin* et *cons* :

   — *tête(l)* <==> *accès(l,1)*

   — *premier(l)* <==> *ième(l,1)*

   — *fin(l)* <==> *supprime(l,1)*

   — *cons(e,l)* <==> *insérer(l,1,e)*

2. Opération *insérer* :

```
l2 <- listevide

l2 <- cons(premier(l),l2)  /* boucler (i-1) fois */
l  <- fin(l)

l  <- cons(e,l)

l  <- cons(premier(l2),l)  /* boucler (i-1) fois */
l2 <- fin(l2)
```

   Comme on peut le constater, dans ce cas la correspondance ne se résume pas à une simple opération, mais l'opération *insérer* à bien été retranscrite en terme des opérations de la *liste récursive*.

Solution 2 (Vérifier la présence – *4 points*)

Spécifications :

Écrire la fonction `check_occ` *list* x n qui vérifie si l'élément x est présent au moins n fois dans la liste *list*.

```
#let rec check_occ x n = function
    [] -> false
  | e :: l when x = e ->
    if n = 1 then
      true
    else
      check_occ x (n-1) l
  | _ :: l -> check_occ x n l ;;

#let check_occ x n list =
  let rec aux list cpt =
    match list with
      [] -> false
     |e::reste -> if e = x then
                    if cpt = (n-1) then
                      true
                    else
                      aux reste (cpt+1)
                  else aux reste cpt
  in aux list 0;;
```

Solution 3 Avant dernier – *4 points*

Spécifications :

Ecrire la fonction `insert_penultimate` x *list* qui insère l'élément x en avant-dernière position de la liste *list*. La fonction déclenche une exception `Invalid_argument` si la liste donnée en paramètre est vide.

```
#let insert_penultimate l el =
  match l with
    [] -> invalid_arg "insert_penultimate: empty list"
   |_ -> let rec aux = function
             |[e] -> el::[e]
             |e::l -> e::aux l
         in aux l;;

#let rec insert_penultimate x = function
    [] -> invalid_arg "insert_penultimate: empty list"
  | e :: [] -> x :: e :: []
  | e :: l -> e :: insert_penultimate x l ;;
```

Solution 4 find2 – *5 points*

Spécifications :

La fonction `find2` retourne le premier couple d'éléments qui vérifie le prédicat p : la première paire d'éléments (a_i, b_i) tel que $p\ a_i\ b_i$ est vrai. Elle déclenche une exception `Failure` si aucun couple (a_i, b_i) tel que $p\ a_i\ b_i$ est vrai n'a été trouvé ou si les deux listes sont de longueurs différentes.

```
let rec find2 p l1 l2 = match (l1,l2) with
  ([],_) | (_,[]) -> failwith "find2: not found or different lengths"
 |(e::l,f::m) -> if p e f then (e,f) else find2 p l m;;
```

Spécifications :

La fonction `first_shared l1 l2` retourne le premier élément présent dans les deux listes à la même position.

```
let first_shared l1 l2 = let a,_ = find2 (function x -> function y ->
x=y) l1 l2 in a;;
```

Solution 5 **(Mystery –** ***3 points*****)**

1. **Spécifications :**

   Donner les résultats des évaluations des phrases suivantes.

```
# let rec mystery a b c = match a with
    []    -> b
   |d::e -> let r = (mystery e b c) in c d r
;;
val mystery : 'a list -> 'b -> ('a -> 'b -> 'b) -> 'b = <fun>

# mystery [] 12 (let c x y = x+y in c) ;;
- : int = 12
# mystery [1; 2; 3; 4; 5] 0 (function x -> function y -> x+y) ;;
- : int = 15
```