Algorithmique
Correction Contrôle n° 1

INFO-SUP S1 – EPITA

Solution 1 **(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 deux manières de définir une liste sont la récursive et l'itérative.
4. Les zones qui composent la signature d'un type abstrait sont les zones TYPES, UTILISE et OPERATIONS.
5. Les opérations qui définissent le type liste récursive sont : listevide, cons, fin, premier.

Solution 2 **(Insertion après –** ***3,5 points*****)**

Spécifications : La fonction `insert_post x f lst` insère l'élément `x` dans la liste `lst` juste après le premier élément `y` tel que `f y` est `true`.

```
let rec insert_post x f = function
  []    -> failwith "insert_post: x cannot be inserted"
 |e::l -> if f e then
             e::x::l
          else
             e::insert_post x f l
;;

val insert_post : 'a -> ('a -> bool) -> 'a list -> 'a list = <fun>
```

Solution 3 **(Position du maximum – *4,5 points*)**

Spécifications : La fonction `pos_max lst` qui retourne la position de l'élément maximum de la liste `lst`. On suppose que la liste `lst` ne contient aucun doublon et la position du premier élément de la liste est 1.

```
let pos_max = function
  []    -> invalid_arg "pos_max: empty list"
 |e::l  -> let rec aux_max vmax vpos cur_pos = function
                []   -> vpos
               |e::l -> if e > vmax then
                           aux_max e cur_pos (cur_pos + 1) l
                        else
                           aux_max vmax vpos (cur_pos + 1) l
           in
             aux_max e 1 2 l;;

let pos_max = function
  []    -> invalid_arg "pos_max: empty list"
 |l     -> let rec aux_max = function
                [e]  -> (e, 1)
               |e::l -> let vmax, vpos = aux_max l
                        in
                          if e > vmax then
                            (e, 1)
                          else
                            (vmax, vpos + 1)
           in
             let _, vpos = aux_max l in vpos;;

val pos_max : 'a list -> int = <fun>
```

Solution 4 **(Less – *5 points*)**

Spécifications : La fonction `less2 p k l1 l2` retourne `true` si le nombre de paires d'éléments (a_i, b_i) tels que $p\ a_i\ b_i$ est `true` est strictement plus petit que `k`, et `false` sinon. Elle déclenche une exception `Invalid_argument` si le paramètre `k` est invalide.

```
let less2 p k l1 l2 =
  if k <= 0 then
    invalid_arg "less: k <= 0"
  else
  let rec aux_less k = function
    ([], [])          -> true
   |(e1::l1, e2::l2)  -> if p e1 e2 then
                           k <> 1 && aux_less (k - 1) (l1, l2)
                         else
                           aux_less k (l1, l2)
  in
    aux_less k (l1, l2)
;;

val less2 : ('a -> 'b -> bool) -> int -> 'a list -> 'b list -> bool = <fun>
```

Spécifications : La fonction `common k l1 l2` vérifie si les listes `l1` et `l2` ont strictement plus de `k` éléments de mêmes valeurs aux mêmes positions. Les deux listes sont supposées de même longueur et le paramètre `k` est supposé positif ou nul.

```
let common k l1 l2 = not (less2 (=) (k+1) l1 l2);;

val common : int -> 'a list -> 'a list -> bool = <fun>
```

Solution 5 **(Mystery – *3 points*)**

```
# let mystery1 l1 l2 =
  let rec aux l3 = function
    (l1, []) -> [l1]
  | (l1, 0::l2) -> l3::aux [] (l1, l2)
  | (e::l1, i::l2) -> aux (e::l3) (l1, (i-1)::l2)
  in
    aux [] (l1, l2);;
val mystery1 : 'a list -> int list -> 'a list list = <fun>

# mystery1 [1; 2; 3; 4; 5; 6; 7; 8; 9; 10] [4; 1; 2];;
- : int list list = [[4; 3; 2; 1]; [5]; [7; 6]; [8; 9; 10]]

# mystery1 [1; 2; 3; 4; 5; 6; 7; 8; 9; 10] [7; 3];;
- : int list list = [[7; 6; 5; 4; 3; 2; 1]; [10; 9; 8]; []]

# mystery1 [5; 15; 5; 21; 5; 1; 4] [3; 1];;
- : int list list = [[5; 15; 5]; [21]; [5; 1; 4]]
```