Algorithmics
Correction Midterm Exam #1

Undergraduate 1^{st} year S1 – Epita

Solution 1 **(Abstract Types: Recursive lists –** ***5 points*****)**

1. The operation `search` is defined only when the searched element exists. Therefore, it is a precondition. Then we have the three axioms applying the observer `ispresent` to the internal operations `emptylist` and `cons`. In order: the element e does not exist in an empty list, the element e exists in a list in which it is equal to the first element and otherwise... try again (it may exist in the tail of the list). Then the axiom explaining that the box returned by `search(e,`λ`)` is the one which contains e.

   PRECONDITIONS

   search(e, λ) **is-defined-iaoi** *ispresent*(e,λ)= true

   AXIOMS

   $ispresent\ (e, emptylist) = \text{false}$
   $e = e' \Rightarrow ispresent\ (e, cons(e', \lambda)) = \text{true}$
   $e \neq e' \Rightarrow ispresent\ (e, cons(e', \lambda)) = ispresent\ (e, \lambda)$
   $contents(search(e, \lambda)) = e$

2. Two axioms suffice. The first says that the concatenation result of an empty list and a list λ is the list λ, which means that the elements of the second list are retained in order and number. The second axiom explains that we also keep in order and number the elements of the first list. How? Showing that if the concatenation is done before or after building (cons) the list, the result is the same, which means that the concatenation modifies neither the order nor the elements.

   AXIOMS

   $concatenate\ (emptylist, \lambda 2) = \lambda 2$
   $concatenate\ (cons(e, \lambda), \lambda 2) = cons(e,\ concatenate\ (\lambda, \lambda 2))$

Solution 2 **(Deletion) –** ***4 points***

Specifications:

Write the function `delete` x *list* that removes the first appearance of the value x (if it is present) from the sorted (in increasing order) list *list*.

```
♯ let rec delete x = function
      [] -> []
    | h::q when h > x -> h :: q
    | h::q when h = x -> q
    | h::q -> h::delete x q ;;
 val delete : 'a -> 'a list -> 'a list = <fun>

♯ delete 4 [1; 2; 2; 3; 4; 4; 4; 5];;
 - : int list = [1; 2; 2; 3; 4; 4; 5]
```

Solution 3 **(Insertion at the rank** i **–** ***5 points*****)**

Spécifications : Write the function `insert_nth` x i *list* that inserts the value x at the rank i in the list *list*.
The function has to raise an exception `Invalid_argument` if i is negative or zero, an exception `Failure` if the list is too short.

```
♯ let insert_nth  x i list =
     if i < 1 then
       invalid_arg "negative rank"
     else
       let rec insert = function
           (1, list)    -> x :: list
         | (_, [])   -> failwith "out of bound"
         | (i, e::q) -> e :: insert(i-1, q)
       in
         insert (i, list);;

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

Solution 4 **(Search)** – ***4 points***

Specifications:
Write the function `search_both` *list* a b that tests whether the two distinct values a and b are in the list *list*.

```
♯ let search v1 v2 l =
    let rec aux1 v l = match l with
        [] -> false
       |e::l -> v = e || aux1 v l
    in
    let rec aux2 l = match l with
        [] -> false
       |e::l -> if e = v1 then
                    aux1 v2 l
                else
                if e = v2 then
                    aux1 v1 l
                else
                    aux2 l
    in aux2 l
;;
 val search : 'a -> 'a -> 'a list -> bool = <fun>
```

Another version:

```
♯ let search v1 v2 l =
    let rec aux l found1 found2 = match l with
        [] -> false
       |e::l -> if e = v1 then
                    found2 || aux l true false
                else
                if e = v2 then
                    found1 || aux l false true
```

```
                else
                    aux l found1 found2
    in aux l false false
;;
 val search : 'a -> 'a -> 'a list -> bool = <fun>
```

Solution 5 **(Mystery –** ***2 points*****)**

1. **Specifications:**
   Give the results of the successive evaluations of the following phrases.

```
♯ let go = function
        [] -> []
      | e::list ->
        let rec what x = function
            [] -> []
          | e::list -> (e * x)::(what e list)
        in
        what e list;;
 val go : int list -> int list = <fun>

♯ go [1; 1; 1; 1; 1] ;;
 - : int list = [1; 1; 1; 1]

♯ go [42] ;;
 - : int list = []

♯ go [1; 2; 3; 4; 5] ;;
 - : int list = [2; 6; 12; 20]

♯ go [2; 21; 2; 21; 2; 21] ;;
 - : int list = [42; 42; 42; 42; 42]
```