Algorithmics
Correction Midterm Exam #1
(Version profs)

UNDERGRADUATE 1^{st} YEAR S1# – EPITA

Solution 1 **(Abstract types: Vector (errors and extension) – *6 points*)**

1. There are two types of problems:
   - a completeness problem (lack of axioms) with a missing axiom applying `isinit` to `vect`.
   - a consistency problem (ambiguity between axioms) with i≠j on one of the axioms applying `isinit` to `modify`.

   The corrected declaration of the type `vector` should be:

   TYPES
   vector
   USES
   integer, element, boolean
   OPERATIONS

vect	:	integer × integer → vector
modify	:	vector × integer × element → vector
nth	:	vector × integer → element
isinit	:	vector × integer → boolean
lowerlimit	:	vector → integer
upperlimit	:	vector → integer

   PRECONDITIONS
   nth(v,i) if-and-only-if lowerlimit(v) ⩽ i ⩽ upperlimit(v) & isinit(v,i)=vrai
   AXIOMS
   lowerlimit(v) ⩽ i ⩽ upperlimit(~~v~~) nth(modify(v,i,e),i) = e
   lowerlimit(v) ⩽ i ⩽ upperlimit(v) & lowerlimit(v) ⩽ j ⩽ upperlimit(v) & i≠j
   ⇒ nth(modify(v,i,e),j) = nth(v,j)
   isinit(vect(i,j),k)=Faux
   lowerlimit(v) ⩽ i ⩽ upperlimit(~~v~~) isinit(modify(v,i,e),i)=vrai
   lowerlimit(v) ⩽ i ⩽ upperlimit(v) & lowerlimit(v) ⩽ j ⩽ upperlimit(v) & i≠j
   ⇒ isinit(modify(v,i,e),j)=isinit(v, j)
   lowerlimit(vect(i,j))=i
   lowerlimit(v) ⩽ i ⩽ upperlimit(~~v~~) lowerlimit(modify(v,i,e))=lowerlimit(v)
   upperlimit(vect(i,j))=j
   lowerlimit(v) ⩽ i ⩽ upperlimit(~~v~~) upperlimit(modify(v,i,e))=upperlimit(v)
   WITH

vector	v
integer	i, j, k
element	e

2. Entension of the type *vector*

(a) There is no precondition. It is an internal operation defined on the bounds of the vector. Its limits will be precised if necessary.

(b) The axioms are the following:

AXIOMS

lowerlimit(v) $\leqslant$ i $\leqslant$ upperlimit(v) isinit(reinitialize(v,i),i)=faux
lowerlimit(v) $\leqslant$ i $\leqslant$ upperlimit(v) & lowerlimit(v) $\leqslant$ j $\leqslant$ upperlimit(v) & i$\neq$j
$\Rightarrow$ isinit(reinitialize(v,i),j) = isinit(v,j)
lowerlimit(v) $\leqslant$ i $\leqslant$ upperlimit(v) & lowerlimit(v) $\leqslant$ j $\leqslant$ upperlimit(v) & i$\neq$j
$\Rightarrow$ nth(reinitialize(v,i),j) = nth(v,j)
lowerlimit(reinitialize(v,i))=lowerlimit(v)
upperlimit(reinitialize(v,i))=upperlimit(v)

WITH

vector v
integer i, j

Solution 2 (Insertion Sort – *7 points*)

1. **Specifications:**
   The function `insert` x l *comp* adds the element x at its place in the list l sorted according to the comparison function *comp*.

```
# let rec insert x list comp =
    match list with
      | [] -> x::[]
      | e::l when comp x e -> x::e::l
      | e::l -> e :: insert x l comp ;;
val insert : 'a -> 'a list -> ('a -> 'a -> bool) -> 'a list = <fun>
```

2. **Specifications:**
   The function `insertion_sort` *comp list* sorts the list *list* in order according to the function *comp*.

```
# let rec insertion_sort comp = function
      [] -> []
    | e::l -> insert e (insertion_sort comp l) comp ;;
val insertion_sort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Tail-recursive version:

```
# let insertion_sort_term comp l =
    let rec sort accu = function
        [] -> accu
      | e::l -> sort (insert e accu comp) l
    in
    sort [] l ;;
val insertion_sort_term : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Solution 3 (Association – *5 points*)

Specifications:
The function `assoc k list` returns the value corresponding to the key `k` in `list`: a list of couples (*key*, *value*) (with *key* > 0) sorted in increasing order with respect to keys. It raises an exception if `k` is not a valid key or if it does not correspond to any couple.

```
# let rec assoc k list =
      if k <= 0 then
        invalid_arg "k not a natural"
      else
        let rec search = function
            []        -> failwith "not found"
          | (key, value)::l -> if key = k then
                                 value
                               else
                                 if k < key then
                                   failwith "not found"
                                 else
                                   search l
        in
        search list ;;

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

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

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

```
# let mystery = function
      [] -> failwith "..."
      |e::f::l -> (let rec aux_mystery m1 m2 = function
                       [] -> m2
                       |e::l -> if e < m1 then aux_mystery e m1 l
                                else if e < m2 then aux_mystery m1 e l
                                else aux_mystery m1 m2 l
                   in if e < f then aux_mystery e f l else aux_mystery f e l);;
val mystery : ('a list) -> 'a = <fun>

  mystery [1;3;4;2];;
   - : int = 2

  mystery [3.5;8.2;9.5;4.0];;
   - : float = 4.0

  mystery ['a'];;
Une erreur (Exception : match_failure)
```

2. **Specifications:**
   What is the return value of `mystery` ?

   The function mystery returns the second smallest element of the list if it exists