Algorithmics
Correction Midterm Exam #1

UNDERGRADUATE 1^{st} YEAR S1 – EPITA

Solution 1 **(A little coursework... –** ***4 points*****)**

1. An internal operation returns a defined type
2. An operation used to specify the domain of definition of another one is an auxiliary operation
3. The problems that arise during the making of the set of axioms are the completeness and the consistency.
4. The areas that make up the signature of an abstract type are the TYPES, USES and OPERATIONS areas.
5. We write axioms by application of observers to internal operations.

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

```
# let rec is_dominoes = function
    | [] | _::[] -> true
    | (_, b)::(c, d)::l -> (b = c) && is_dominoes ((c, d)::l) ;;

# let rec is_dominoes = function
    | [] | [_] -> true
    | (_, b)::(c, _)::_ when b <> c -> false
    | _::(c, d)::l -> is_dominoes ((c, d)::l) ;;

# let rec is_dominoes = function
    | [] | _::[] -> true
    | (_, b)::(c, d)::l -> if b <> c then false
                           else is_dominoes ((c,d)::l) ;;

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

Solution 3 **(Deletion at rank i – *5 points*)**

Spécifications :

The function `remove_nth` *i list* deletes the value at the rank i in the list *list*.
It raises an exception `Invalid_argument` if i is negative or zero, or an exception `Failure` if the list is too short.

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

# let remove_nth i list =
    let rec del i = function
        []   -> failwith "out of bound"
      | e::q -> if i = 1 then
                  q
                else
             e :: del (i-1) q
    in
      if i < 1 then
        failwith "negative rank"
      else
        del i list ;;

# let remove_nth i list =
    if i < 1 then
      failwith "negative rank"
    else
      let rec del i list = match list with
       []   -> failwith "out of bound"
        | _::q when i = 1 -> q
        | e::q -> e::del (i-1) q
      in
        del i list ;;

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

Solution 4 **(for_all2 – *5 points*)**

1. **Specifications:**
   The function `for_all2`:
   - takes a two-argument predicate (a boolean function), p, and two lists, $[a_1; a_2; \cdots; a_n]$ and $[b_1; b_2; \cdots; b_n]$, as parameters.
   - returns $p\ a_1\ b_1$ && $p\ a_2\ b_2$ && $\cdots$ && $p\ a_n\ b_n$.
   - raises `Invalid_argument` if the two lists have different lengths.

```
# let rec for_all2 p list1 list2 =
    match (list1, list2) with
        ([],[]) -> true
      | (_,[]) | ([],_) -> invalid_arg "for_all2: different lengths"
      | (a::l1,b::l2) -> p a b && for_all2 p l1 l2 ;;

# let rec for_all2 p list1 list2 =
    match (list1, list2) with
        ([], []) -> true
      | (_, []) | ([], _) -> invalid_arg "for_all2: different lengths"
      | (a::l1, b::l2) when p a b -> for_all2 p l1 l2
      | _ -> false ;;

# let rec for_all2 p list1 list2 =
    match (list1, list2) with
        ([], []) -> true
      | (_, []) | ([], _) -> invalid_arg "for_all2: different lengths"
      | (a::l1, b::l2) -> if p a b then
                            for_all2 p l1 l2
                          else
                            false ;;
val for_all2 : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool = <fun>
```

2. **Specifications:**
   The function `equal` takes two lists as parameters. It returns *true* if they are identical, *false* otherwise. It raises an exception if the two lists have different lengths.

```
# let almost list1 list2 =
    let near x y = (y - 2 < x) && (x < y + 2)
    in
      for_all2 near list1 list2 ;;

val almost : int list -> int list -> bool = <fun>
```

Solution 5 **(Mystery – 2 points)**

```
# let mystery a b =
    let rec what = function
        ([], _) -> true
      | (_, []) -> false
      | (e::l1, f::l2) -> (e = f) && what (l1, l2)
    in
    let rec is_that x y = match y with
        [] -> 0
      | e::q -> (if what (x, y) then 1 else 0) + (is_that x q)
    in
      is_that a b ;;

val mystery : 'a list -> 'a list -> int = <fun>

  # mystery [1; 2] [1; 2];;
- : int = 1

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

  # mystery [1; 2] [2; 1];;
- : int = 0
```