Last name	
First name	
Group	

Grade	/ 5

Algorithmics
Midterm Exam 1 - Part. 1

Undergraduate 1^{st} year S1

EPITA

29 Oct. 2018 - 8 : 30

□ **This is the part 1 of the subject - You have to give back the two parts!**

□ You must answer on **this subject.**

– Answer within the provided space. **Answers outside will not be marked.**
– Penciled answers will not be marked.

□ The presentation is marked.

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

Consider the algebraic abstract type *recursive list* seen in class and recalled below.

TYPES

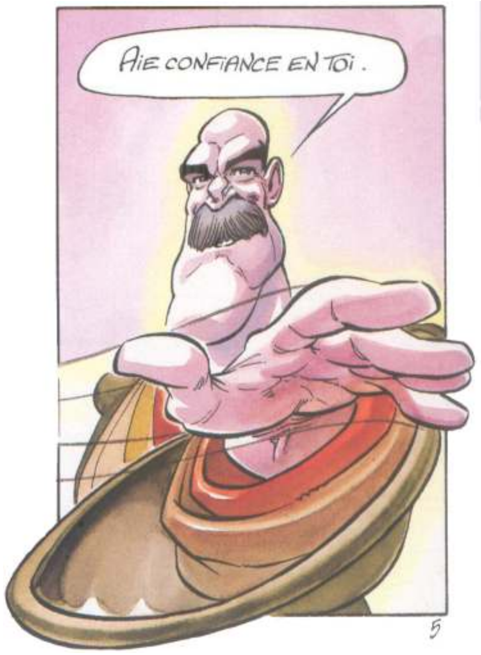
list, box

USES

element

OPERATIONS

emptylist	:	→ list
head	:	list → box
contents	:	box → element
first	:	list → element
cons	:	element × list → list
tail	:	list → list
next	:	box → box



PRECONDITIONS

head(λ) **is-defined-iaoi** $\lambda \neq$ *emtylist*
tail(λ) **is-defined-iaoi** $\lambda \neq$ *emtylist*
first(λ) **is-defined-iaoi** $\lambda \neq$ *emtylist*

AXIOMS

$$first(cons(e, \lambda)) = e$$
$$tail(cons(e, \lambda)) = \lambda$$
$$contents(head(\lambda)) = first(\lambda)$$
$$next(head(\lambda)) = head(tail(\lambda))$$

WITH

list	λ
element	e

We propose to extend the properties of this type allowing it:

- to search for an element in a list
- to concatenate two lists.

The search for an item in a list will return the corresponding box to the element only if it exists. Then we have two operations for the search, the one which determines the existence of the element and the other one which determines the box for the latter, if it exists. As for the concatenation, it requires no auxiliary operation. We then consider the three following operations:

OPERATIONS
ispresent : element × list → boolean
search : element × list → box.

concatenate : liste × list → list

1. Give the axioms allowing one to deduce a value for the search for an element e in a *recursive list* λ. Specify the PRECONDITIONS if there are any.

2. Give the axioms allowing one to deduce a value for the concatenation of two *recursive lists* λ and $\lambda 2$. Specify the PRECONDITIONS if there are any.

Last name	
First name	
Group	

Grade	/ 16

Algorithmics
Midterm Exam 1 - Part. 2

Undergraduate 1st year S1

Epita

29 Oct. 2018 - 8 : 30

□ **This is the part 1 of the subject - You have to give back the two parts!**

□ You must answer on **this subject.**

- Answer within the provided space. **Answers outside will not be marked.**
- Penciled answers will not be marked.

□ **Caml :**

- All Caml code not indented will not be marked.
- In the absence of any indication in the document, the only functions that you can use are `failwith` and `invalid_arg` (no other predefined function of Caml).

□ The presentation is marked.

Exercise 2 (is_image – *4 points*)

Write the Caml function `for_all2` with the following specifications:

- It takes a one-argument function, f, and two lists, $[a_1; a_2; \cdots; a_n]$ and $[b_1; b_2; \cdots; b_n]$, as parameters.
- It checks whether for all pairs of elements $(a_i,\, b_i)$ that b_i is the image of a_i under f.
- If a pair such that $f\ a_i \neq b_i$ is found, it returns false. Otherwise, it raises `Invalid_argument` if the two lists have different lengths.

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

 ♯ is_image (function x -> x * 2) [1; 2; 3; 4; 5] [2; 4; 6; 8; 10] ;;
  - : bool = true
 ♯ is_image (function x -> string_of_int x) [1; 2; 3; 4; 5] ["1"; "2"; "0"; "4"; "5"] ;;
  - : bool = false
```

Exercise 3 (How many? – *4 points*)

1. Write the CAML function `how_many` with the following specifications:
   - it takes a boolean function f and a list $[a_1; a_2; \cdots; a_n]$ as parameters.
   - It returns the number of values a_i such that $f(a_i)$ is true.

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

2. Use the function `how_many` to define the function `count_multiples` n l that returns the number of multiples of n in the list l.

```
val count_multiples : int -> int list -> int = <fun>
```

Exercise 4 (Insertion at the rank i – *5 points*)

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 an exception `Failure` if the list is too short.

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

Application examples:

```
♯ insert_nth 0 5 [1; 2; 3; 4; 5; 6; 7; 8; 9];;
 - : int list = [1; 2; 3; 4; 0; 5; 6; 7; 8; 9]

♯ insert_nth 0 10 [1; 2; 3; 4; 5; 6; 7; 8; 9];;
 - : int list = [1; 2; 3; 4; 5; 6; 7; 8; 9; 0]

♯ insert_nth 0 12 [1; 2; 3; 4; 5; 6; 7; 8; 9];;
 Exception: Failure "out of bound".

♯ insert_nth 0 (-2) [1; 2; 3; 4; 5; 6; 7; 8; 9];;
 Exception: Invalid_arg "negative rank".
```

Exercise 5 (Evaluations – *3 points*)

Give the results of the successive evaluations of the following phrases.

```
♯ let rec decode = function
      [] -> []
    | (1, e)::list -> e::decode list
    | (nb, e)::list -> e::decode ((nb-1, e)::list) ;;
```

```
♯ decode [(6, "grr")] ;;
```

```
♯ decode [(1, ’a’); (3, ’b’); (1, ’c’); (1, ’d’); (4, ’e’)] ;;
```

```
♯ let encode list =
    let rec encode_rec (nb, cur) = function
        []   -> [(nb, cur)]
      | e::list -> if e = cur then
                     encode_rec (nb+1, cur) list
                   else
                     (nb, cur)::encode-rec (1, e) list
    in
      match list with
          []   -> []
        | e::l -> encode_rec (1, e) list
```

```
♯ encode [0; 0; 0; 0; 0; 0; 0; 0; 0; 0] ;;
```

```
♯ encode [’b’;’b’;’b’; ’c’; ’a’;’a’; ’e’;’e’;’e’;’e’; ’d’;’d’] ;;
```