Nom	
Prénom	
Groupe	

Note	/ 5

Algorithmique
Contrôle 1 - Part. 1

INFO-SUP S1

EPITA

29 Oct. 2018 - 8 : 30

- ☐ **Ceci est la partie 1 de l'épreuve - Vous devez rendre les deux parties !**
- ☐ Vous devez répondre directement **sur ce sujet**.
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées**.
  - Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée.

Exercice 1 (Types Abstraits : Listes récursives – *5 points*)

Supposons le type abstrait algébrique *Liste récursive* vu en cours et rappelé ci-dessous.

TYPES

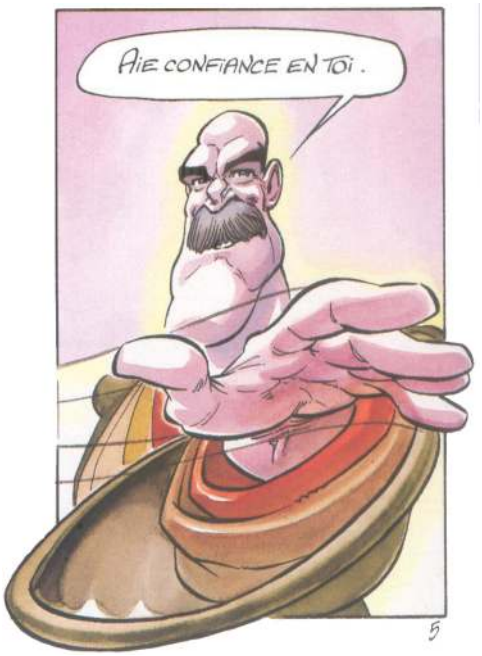
liste, place

UTILISE

élément

OPÉRATIONS

listevide	:	$\rightarrow$ liste
tête	:	liste $\rightarrow$ place
contenu	:	place $\rightarrow$ élément
premier	:	liste $\rightarrow$ élément
cons	:	élément $\times$ liste $\rightarrow$ liste
fin	:	liste $\rightarrow$ liste
succ	:	place $\rightarrow$ place



PRÉCONDITIONS

tête(λ) **est-défini-ssi** $\lambda \neq$ *listevide*
fin(λ) **est-défini-ssi** $\lambda \neq$ *listevide*
premier(λ) **est-défini-ssi** $\lambda \neq$ *listevide*

AXIOMES

$premier(cons(e, \lambda)) = e$
$fin(cons(e, \lambda)) = \lambda$
$contenu(tête(\lambda)) = premier(\lambda)$
$succ(tête(\lambda)) = tête(fin(\lambda))$

AVEC

liste	λ
élément	e

On se propose d'étendre les propriétés de ce type en lui permettant :
- de rechercher un élément dans une liste
- de concaténer deux listes.

La recherche d'un élément parmi ceux d'une liste ne retournera la place correspondante à celui-ci que s'il existe. Dès lors nous avons deux opérations pour la recherche, celle qui détermine la présence effective de l'élément et celle qui détermine la place de ce dernier s'il est présent. La concaténation n'a quant à elle besoin d'aucune opération auxiliaire. Nous considérerons donc les trois opérations suivantes :

OPÉRATIONS

est-présent : élément × liste → booléen
rechercher : élément × liste → place.

concaténer : liste × liste → liste

1. Donner les axiomes déduisant une valeur pour la recherche d'un élément e parmi ceux d'une *liste récursive* λ. Vous préciserez les PRÉCONDITIONS s'il y en a.

2. Donner les axiomes déduisant une valeur pour l'opération de concaténation de deux *listes récursives* λ et $\lambda 2$. Vous préciserez les PRÉCONDITIONS s'il y en a.

Nom	
Prénom	
Groupe	

Note	/ 16

Algorithmique
Contrôle 1 - Part. 2

Info-sup S1

Epita

29 Oct. 2018 - 8 : 30

- ☐ **Ceci est la partie 2 de l'épreuve - Vous devez rendre les deux parties !**
- ☐ Vous devez répondre directement **sur ce sujet**.
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées**.
  - Aucune réponse au crayon de papier ne sera corrigée.
- ☐ **Caml :**
  - Tout code Caml non indenté ne sera pas corrigé.
  - En l'absence d'indication dans l'énoncé, les seules fonctions que vous pouvez utiliser sont `failwith` et `invalid_arg` (aucune autre fonction prédéfinie de Caml).
- ☐ La présentation est notée.

Exercice 2 (is_image – *4 points*)

Écrire la fonction Caml `is_image` dont les spécifications sont les suivantes :
- Elle prend en paramètre une fonction à un paramètre : `f` ainsi que deux listes : $[a_1; a_2; \cdots; a_n]$ et $[b_1; b_2; \cdots; b_n]$.
- Elle vérifie pour toutes les paires (a_i, b_i) que b_i est l'image de a_i par f.
- Si elle trouve une paire telle que $f\ a_i \neq b_i$, elle retourne faux. Sinon, elle déclenche une exception `Invalid_argument` si les deux listes sont de longueurs différentes.

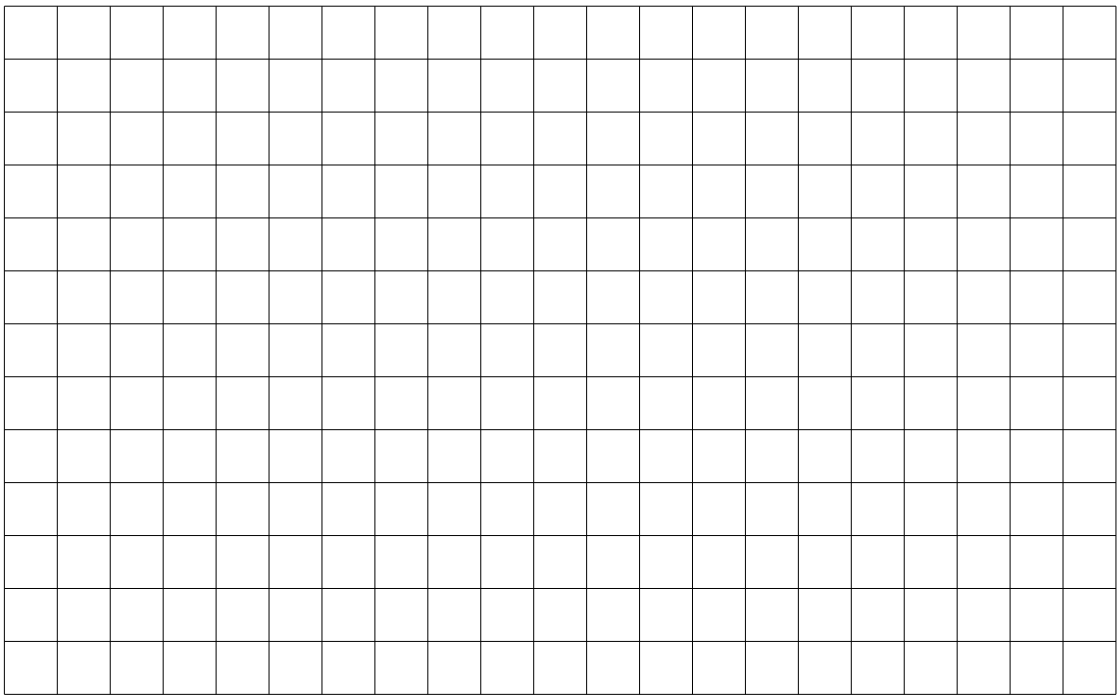
```
val is_image : ('a -> 'b) -> 'a list -> 'b list -> bool = <fun>

# 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
```

Exercice 3 (Combien ? – *4 points*)

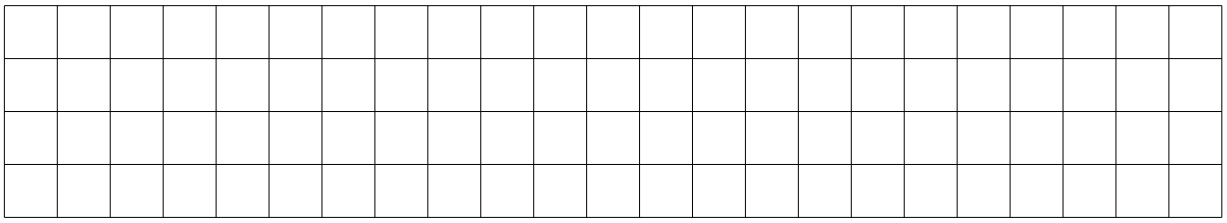
1. Écrire la fonction CAML `how_many` dont les spécifications sont les suivantes :
   - Elle prend en paramètre une fonction booléenne f ainsi qu'une liste : $[a_1; a_2; \cdots; a_n]$.
   - Elle recherche dans la liste les valeurs a_i telle que $f(a_i)$ soit vrai et retourne le nombre de valeurs trouvées.

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



2. Utiliser la fonction `how_many` pour définir la fonction `count_multiples` n l qui retourne le nombre de multiples de n dans la liste l.

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



Exercice 4 (Insertion à la $i^{ème}$ place – *5 points*)

Écrire la fonction `insert_nth` x i *list* qui insère la valeur x à la $i^{ème}$ place dans la liste *list*.
La fonction devra déclencher une exception `Invalid_argument` si i est négatif ou nul, ou une exception `Failure` si la liste est trop courte.

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

Exemples d'application :

```
# 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".
```

Exercice 5 (Évaluations – *3 points*)

Donner les résultats des évaluations successives des phrases suivantes.

```
# 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'] ;;
```
