

Nom	
Prénom	
Groupe	E1

Note	4 / 4
------	-------

Algorithmique Contrôle 1 - Partie 1

INFO-SUP S1

EPITA

31 Oct. 2022 - 08 : 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 (Un peu de cours... - 4 points)

1. Quel type de données retourne un observateur ?

Un observateur retourne un argument de type prédéfini @1, ✓

2. Comment appelle-t-on une opération qui n'est pas définie partout ?

Une opération partielle @1, ✓

3. Quelles sont les deux façons de définir une liste ?

- récursive ✓ sont les deux façons de définir une
- itérative ✓ liste. 1

4. Quelles zones constituent la signature d'un type abstrait ?

Les zones qui constituent la signature d'un
type abstrait sont: OPERATIONS, TYPES et
UTILISE. ✓ 1

5. Donnez trois opérations définissant une liste récursive (sans places).

cons: élément x liste → liste
fin: liste → liste et tête: liste → élément

325

Nom	PELOU
Prénom	Lucile
Groupe	E1
Prof TD	

Note	14.5 / 16
------	-----------

Algorithmique
Contrôle 1 - Partie 2

INFO-SUP S1

EPITA

31 Oct. 2022 - 08 : 30

Remarques (à lire!) :

-
- ☐ 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 ou au stylo rouge 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).
 - Tout code CAML doit être suivi du résultat son évaluation : la réponse de CAML.
 - ☐ La présentation est notée.
-

Exercice 2 (Insertion après - 3,5 points)

Écrire la fonction `insert_post x f lst` qui prend en paramètres :

- un élément `x`
- une fonction à un paramètre `f` renvoyant une valeur booléenne
- une liste `lst`

et qui insère l'élément `x` dans la liste `lst` juste après le premier élément `y` tel que `f y` est `true`.

La fonction devra déclencher une exception `Failure` s'il n'existe aucun élément `y` après lequel `x` peut être inséré.

```
# insert_post 42 (function y -> y mod 2 = 0) [5; -7; 2; 4; 1];;
- : int list = [5; -7; 2; 42; 4; 1]

# insert_post "toto" (function y -> y = "one") ["one"; "two"; "three"; "four"];;
- : string list = ["one"; "toto"; "two"; "three"; "four"]

# insert_post 4.1 (function y -> y < 0.) [1.1; 2.4; 2.];;
Exception: Failure "insert_post: x cannot be inserted".
```

Fonction CAML :

```
#let rec insert_post x f lst =
  match lst with
  | [] -> failwith "insert_post: x cannot be inserted"
  | y::l when f y = true -> y::(x::l)
  | y::l -> y::insert_post x f l;;

val insert_post : 'a -> ('a -> bool) -> 'a list -> 'a list = fun
```


Exercice 3 (Position du maximum - 4,5 points)

Écrire la fonction posmax lst qui retourne la position de l'élément maximum de la liste lst. On suppose que la liste lst ne contient aucun doublon et la position du premier élément de la liste est 1.
 La fonction devra déclencher une exception Invalid_argument si la liste lst est initialement vide.

```
# pos_max [];;
Exception: Invalid_argument "pos_max: empty list".
# pos_max [1; 2; 3];;
- : int = 3
# pos_max [3; 2; 1];;
- : int = 1
# pos_max [8.5; 9.; -4.5];;
- : int = 2
```

Fonction CAML :

```
#let posmax lst =
  match lst with
  | [] -> invalid_arg "pos_max: empty list"
  | _ :: list ->
    let rec aux lst max i b =
      match lst with
      | [] -> b
      | e :: l -> if e > max then aux l e (i+1) b
                  else aux l max (i+1) b
    in aux list max 1 1;;

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

Exercice 4 (Less - 5 points)

4

1. Écrire la fonction `less2` p k l1 l2 dont les spécifications sont les suivantes :

- Elle prend en paramètre une fonction à deux paramètres `p`, un entier strictement positif `k` ainsi que deux listes : $[a_1; a_2; \dots; a_n]$ et $[b_1; b_2; \dots; b_n]$. Les deux listes sont supposées de même longueur.
- Elle retourne `true` si le nombre de paires d'éléments (a_i, b_i) tels que `p a_i b_i` est `true` est strictement plus petit que `k`, et `false` sinon.
- Elle déclenche une exception `Invalid_argument` si le paramètre `k` est invalide.

```
# less2 (function x -> function y -> x = y) 2 [] [];;
- : bool = true

# less2 (function x -> function y -> x = y) 0 [5; 1; 1; 2] [1; 5; 1; 4];;
Exception: Invalid_argument "less: k <= 0".

# less2 (function x -> function y -> x > y) 3 [5.; 1.1; 1.8; 2.5] [1.; 5.7; 1.9; 5.];;
- : bool = true

# less2 (function x -> function y -> x mod y = 0) 2 [5; 15; 5; 2] [7; 5; 4; 4];;
- : bool = true
```

3,4

Fonction CAML :

```
let less2 p k l1 l2 =
  if k <= 0 then invalid_arg "less: k <= 0"
  else
    let rec aux2 k l1 l2 =
      match (l1, l2, k) with
      | (l1, l2, 0) -> false
      | ([], [], _) -> true
      | (e::l, n::p, k) when p e n = true ->
        aux2 (k-1) l p
      | (e::l, n::p, k) -> aux2 k l p
    in aux2 k l1 l2;;

val less2: ('a -> 'b -> bool) -> int -> 'a list -> 'b list
-> bool = <fun>
```

0,1,4

2. Utiliser la fonction précédente `less2` (qu'elle soit écrite ou pas) pour écrire la fonction `common k l1 l2` qui vérifie si les listes `l1` et `l2` ont strictement plus de `k` éléments de mêmes valeurs aux mêmes positions. Les deux listes sont supposées de même longueur et le paramètre `k` est supposé positif ou nul.

```
# common 0 ['a'; 'y'; 'c'] ['c'; 'a'; 'y'];;
- : bool = false
# common 2 [1; 2; 3; 4] [1; 2; 3; 5];;
- : bool = true
# common 1 ['a'; 'y'; 'c'] ['b'; 'y'; 'c'];;
- : bool = true
```

Fonction CAML :

~~let~~ common k l₁ l₂ =
 less2 (function x → function y → x=y) k l₁ l₂;;
 val common : int → 'a list → 'a list → bool = <fun>

Exercice 5 (Mystery - 3 points)

1. Donner le type de la fonction suivante.

```
# let mystery1 l1 l2 =
  let rec aux l3 = function
    (l1, []) → [l1]
  | (l1, 0::l2) → l3::aux [] (l1, l2)
  | (e::l1, i::l2) → aux (e::l3) (l1, (i-1)::l2)
  in
    aux [] (l1, l2);;
```

val mystery1 : int list → int list → int list list = <fun>

2. On suppose la fonction `mystery1` ci-dessus correcte et dans l'environnement Caml. Donner les résultats des évaluations successives des phrases suivantes si elles sont correctes. Si elles sont incorrectes indiquez "Erreur".

```
# mystery1 [1; 2; 3; 4; 5; 6; 7; 8; 9; 10] [4; 1; 2];;
```

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

```
# mystery1 [1; 2; 3; 4; 5; 6; 7; 8; 9; 10] [7; 3];;
```

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

3. Donner une application de la fonction `mystery1` renvoyant le résultat ci-dessous.

```
# mystery1 ???;;
- : int list list = [[5; 15; 5]; [21]; [5; 1; 4]]
```

#mystery1 [5;15;5;21;5;1;4] [3;1];;