Nom	Hyperion
Prénom	Rémi
Groupe	A2 - Alizée

Note	15,5

Algorithmique
S1
Contrôle n° 1 (C1)
6 Nov. 2017
Feuilles de réponses

I	1
II	1,5
III	5,5
IV	4,5
IV	5

Réponses 1 **(Types Abstraits : liste itérative (modifier)** – *2 points*)

1. Précondition éventuelle.
   $i \geq 0$ && $i <$ longueur(l) et liste non vide ?

2. Axiomes de l'opération *modifier*.
   $0 \leq i$ && $i <$ longueur(l) => ième(modifier(l,i,e),i) = e ✓
   $i <$ longueur(l) && (j < longueur(l) && $i \neq j$) => ième(modifier(l,i,e),j) = ième(l,j) ✓
   1

Réponses 2 **(Types Abstraits : liste récursive arrière** – *4 points*)

1. Préconditions éventuelles.
   queue(l) : l ≠ liste_vide est définissable
   debut(l) : l ≠ liste_vide
   dernier(l) : l ≠ liste_vide
   ~~pred(i) : i > 1~~
   1,5

2. Axiomes du type *liste récursive arrière*.
   l = cons(e, liste_vide) => tête(l) = queue(l)
   l ≠ liste_vide => ~~contenu(queue(l)) = e~~
   l = cons(e, liste_vide) => ~~debut(l) = liste_vide~~
   l ≠ liste_vide => contenu(~~queue~~(debut(l))) = contenu(queue(l))
   l = cons(e, liste_vide) => ~~dernier(l) = e~~
   l ≠ liste_vide => contenu(queue(l)) = dernier(l) ✓
   i = queue(l) => ~~contenu(queue(debut(l)))~~ = contenu(pred(i))
   i > 1 => ~~contenu(pred(i))~~ = ~~contenu(i-1)~~

Réponses 3 (**Association** – *4 points*)

La fonction `assoc` :

```
let assoc k list =
  if k > 0 then
    let rec assoc_rec k list =
      match list with
        [(n,s)::l when n = k -> s
        |(n,_)::l when n > k -> fail with "not found"
        |(_,_)::l -> assoc_rec k l
    in
    assoc_rec k list
  then
    invalid_arg "k not a natural";;

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

liste vide ?

3,5

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

1. La fonction `for_all2` :

```
let rec for_all2 p list1 list2 =
    match (list1, list2) with
      |([], []) -> true
      |(_, []) | ([], _) -> invalid_arg "les listes ont des longueurs
différentes"
      |(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>
```

5

2. La fonction `almost` :

```
let almost list1 list2 =                                  (a & b-a>0)
  let p a b = if (a-b<2) || (b-a<2) then
                true                    (&& a-b>0)
              else
                false
  in for_all2 p list1 list2 ;;
val almost : 'a list -> 'b list -> bool = <fun>
```

0,5

Réponses 5 **(Combine it – *5 points*)**

La fonction `combfilter` :

```
let rec combfilter f list1 list2 =
    match (list1, list2) with
      |([],[]) -> []
      |(_,[]) |([],_) -> invalid_arg "les listes n'ont pas la
même longueur"
      |(e1::l1, e2::l2) -> if f e1 e2 then
                             (e1,e2)::combfilter f l1 l2
                           else
                             combfilter f l1 l2;;

val combfilter ('a -> 'b -> bool) -> 'a list -> 'b list -> ('a*'b)
list = <fun>
```

5