Correction du devoir surveillé n°1

MPSI Lycée Clemenceau : option informatique

Lundi 24 mars 2025

Vous avez 3 heures.

Il sera tenu compte de la présentation et de la rédaction. Les programmes (ou fonctions) devront être commentés et le plus souvent justifiés (terminaison et correction).
Lire l'annexe pour les fonctions prédéfinies et autorisées pour chaque exercice.

Exercice 1

Une application φ de l'ensemble $[\![0,99]\!]$ dans lui-même est représentée par un tableau `t` à cent cases : pour tout entier i dans $[\![0,99]\!]$, la i-ième case du tableau `t` contient $\varphi(i)$.
Ecrire un programme `inverse` qui prend en entrée le tableau `t` et retourne le tableau `u` représentant l'application réciproque φ^{-1} lorsque φ est bijective ; dans le cas contraire le programme affiche le message « non bijectif ».

Correction : soit `tab` un tableau représentant une application φ de $[\![0,n-1]\!]$ dans lui-même. On a donc par définition, pour $i \in [\![0,n-1]\!]$, `tab.(i)`$= \varphi(i)$. Si `tab2` est le tabeau représentant, dans le cas où φ est bijective, de φ^{-1}, alors, pour $j \in [\![0,n-1]\!]$, `tab2.(j)`$= \varphi^{-1}(j)$. Si on pose $i = \varphi^{-1}(j)$ alors `tab2.(`φ`(i))=i`. Ce qui s'écrit encore `tab2.(tab.(i))=i`.

Une application de $[\![0,n-1]\!]$ dans lui même est bijective si et seulement si elle est injective. On va donc parcourir le tableau `tab` pour remplir le tableau `tab2`. Si une cas a déjà été remplis alors φ n'est pas injective. On initialise `tab2` avec (-1) pour voir si la case est occupée ou non.

```
let inverse tab = let n = Array.length tab in
      let tab2 = Array.make n (-1) in
      let i = ref 0 in
      while !i<n do
          if tab2.(tab.(!i)) <> -1 then failwith "application non bijective"
             else begin
               tab2.(tab.(!i))<- !i;
               incr i
               end
           done;
      tab2;;
```

Exercice 2 : Arithmétique

Un entier naturel différent de 1 et de 0 est premier si ses seuls diviseurs positifs sont 1 et lui-même.

1) Test de primalité.

a) Proposer une fonction `premier : int-> bool` qui renvoie `true` si un entier naturel différent de 1 et de 0 est premier et `false` sinon.

Correction : on va parcourir tous les entiers entre 2 et $n-1$ en s'arrêtant si on trouve un diviseur de n.

Version irérative :

```
let premier n = let d = ref 2 and res = ref true in
       while !d <n && !res do
          if n mod !d =0 then res:= false else incr d
       done;
       !res;;
```

Version récursive :

```
let premier n = let rec aux d = match d with
                    | d when d = n -> true
                    | d when n mod d =0 -> false
                    | _ -> aux (d+1)
                in aux 2;;
```

b) Montrer que si p, un entier naturel différent de 1 et de 0, n'est pas premier alors il existe $d \in \mathbb{N} \setminus \{0,1\}$ tel que d divise p et $d^2 \leqslant p$.

Correction : soit n un entier non nul et différent de 1. On suppose que n n'est pas un nombre premier. Il existe alors $(p,q) \in (\mathbb{N} \setminus \{0,1\})^2$ tel que $n = pq$.

On suppose que $p = \min\{p,q\}$. On a alors $p^2 \leqslant pq$, or $pq = n$, d'où $p^2 \leqslant n$.

c) En utilisant la question précédente, proposer un algorithme `premier2 : int-> bool` qui renvoie `true` si un entier naturel différent de 1 et de 0 est premier et `false` sinon.

Correction : on adapte la première fonction

Version itérative :

```
let premier2 n = let d = ref 2 and res = ref true in
       while !d * !d <n && !res do
          if n mod !d =0 then res:= false else incr d
       done;
       !res;;
```

Version récursive :

```
let premier2 n = let rec aux d = match d with
                    | d when d * d > n -> true
                    | d when n mod d =0 -> false
                    | _ -> aux (d+1)
                in aux 2;;
```

d) Dénombrer le nombre d'opérations pour `premier` et `premier2`.

Correction : dans le pire des cas (le nombre est premier) la première version fait $O(n)$ opérations (comparaisons, divisions euclidiennes ...), alors que la seconde n'en fait que $O(\sqrt{n})$.

2) Calcul de PGCD. Soit a_1, a_2, ..., a_n des entiers naturels non nuls. Le PGCD de ces n entiers est leur plus grand commun diviseur positif.

a) Proposer une fonction `pgcd : int -> int -> int` qui renvoie le PGCD de deux entiers naturels non nuls. *(sans l'algorithme d'Euclide qui intervient après.)*

Correction : on va parcourir les entiers entre 1 et $\min\{a,b\}$ et garder le plus grand diviseur commun.

```
let pgcd a b = let res = ref 1 in
     for d = 2 to min a b do
        if a  mod d =0 && b  mod d = 0 then res:=d
     done;
     !res;;
```

b) Soit $(a,b,q,r) \in (\mathbb{N}^*)^2 \times \mathbb{N}^2$ tels que $a = bq + r$. Montrer que si $r = 0$ alors $PGCD(a,b) = b$ et que sinon $PGCD(a,b) = PGCD(b,r)$

Correction : cf cours de maths

c) En utilisant la question précédente, proposer une fonction itérative (donc non récursive) `pgcd2 : int-> int -> int` qui renvoie le PGCD de deux entiers naturels non nuls.

Correction : on applique la question précédente

```
let pgcd2 n m = let a = ref n and b = ref m in
           let r = ref (!a mod !b) in
   while !r <> 0 do
       a:= !b;
       b:= !r;
       r:= !a mod !b
   done;
   !b;;
```

d) Proposer une fonction récursive sans fitrage `pgcd3 : int-> int -> int` qui renvoie le PGCD de deux entiers naturels non nuls.

```
let rec pgcd3 a b = if a mod b = 0 then b else pgcd3 b (a mod b);;
```

e) En sachant que $PGCD(a1, a2, ..., an) = PGCD(PGCD(a1, a2, ..., an-1), an)$, proposer une fonction `pgcd4 : int array -> int` qui renvoie le PGCD d'un ensemble d'entiers naturels non nuls contenus dans un tableau.

Correction : on applique la propriété de l'énoncé pour écrire une fonction auxiliaire récursive

```
let pgcd4 tab = let n = Array.length tab in
   let rec aux k = if k = 0 then tab.(0) else pgcd3 (aux (k-1)) tab.(k) in
   aux (n-1);;
```

Exercice 3 : Cryptographie de substitution

Une variable de type string et de longueur $\ell \in \mathbb{N}^*$ peut être vue comme un tableau v indexé de 0 à $\ell - 1$. Le terme en i^e position est noté $v.[i-1]$. L'instruction `String_length` donne la longueur d'une chaîne de caractères.

Les chaines de caractères n'étant plus modifiables l'exercice utilisera des tableaux de caractères pour les valeurs obtenues par les fonctions

1) On rappelle le théorème de la division euclidienne pour les entiers :

Soit $a \in \mathbb{Z}$ et $b \in I\!N^*$. Il existe un unique couple $(q, r) \in \mathbb{Z} \times I\!N$ tel que $a = bq + r$ et $0 \leqslant r < b$.

Soit a et b des entiers avec $b > 0$, en `Ocaml` l'instruction `a mod b` renvoie le reste de la division euclidienne de a par b lorsque a est positif, sinon elle renvoie le reste moins b.

Proposer une fonction `modulo : int*int -> int` qui renvoie le reste de la division euclidienne de a par b quelque soit le signe de a.

Correction : Remarque : *en fait l'énoncé est faux car, lorsque a est négatif et est un multiple de b alors CAML donne le bon reste.*

Voici la version attendue :

```
# let modulo (a,b) =
    if a >= 0 then a mod b else a mod b + b;;
```

Voici la bonne version (qui tient compte du texte dans la définition) :

```
# let modulo (a,b) =
    if (a >= 0) || (a mod b =0) then a mod b else a mod b + b;;
```

2) La fonction `int_of_char` permet d'associer à une variable de type `char` une variable de type `int` (codes ASCII). Par exemple `int_of_char 'a' ;;` renvoie la valeur 97, `int_of_char 'b' ;;` renvoie 98 et ainsi de suite jusque `int_of_char 'z' ;;` qui donne 122 en respectant l'ordre alphabétique.

La fonction `char_of_int` est la fonction inverse.

À chaque lettre de l'alphabet, on souhaite associer un nombre comme ci-dessous

a	b	c	d	e	f	...	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	...	17	18	19	20	21	22	23	24	25

(a) Donner une fonction `i_of_c : char -> int` qui à toute lettre associe l'entier compris entre 0 et 25 qui lui correspond.

Correction : sans faire compliqué et en suivant le texte :

```
# let i_of_c a = int_of_char a - 97;;
```

(b) Donner une fonction `c_of_i : int -> char` qui à tout entier compris entre 0 et 25 lui associe la lettre qui lui correspond.

Correction : sans commentaires

```
# let c_of_i k = char_of_int (k + 97);;
```

3) Jules César codait ses messages écrits en opérant un décalage de trois lettres vers la "droite" dans l'alphabet usuel avec un retour en "a" dans le cas d'un dépassement. Ainsi "Gaule" devenait "Jdxoh". Le "z" devenait "c".

Proposer une fonction de type `code : string -> char array` qui en entrée reçoit une chaîne de caractères et qui renvoie un tableau contenant les caractères décalés de trois rangs vers la "droite" dans l'alphabet usuel. On prendra en compte les décalages des dernières lettres de l'alphabet.

Correction : il suffit d'utiliser les fonctions des questions précédentes en utilisant le mod pour l'effet de bord.

```
let code message = let n = String.length message in
         let mes_code = Array.make n 'a' in
         for i = 0 to n-1 do
```

```
        mes_code.(i)<-c_of_i((i_of_c message.[i] + 3) mod 26)
    done;
    mes_code;;
```

4) Proposer une fonction de type `decode : string -> char array` qui en entrée reçoit une chaîne de caractères et qui renvoie le tableau avec les caractères décalés de trois rangs vers la "gauche" dans l'alphabet usuel. On prendra en compte les décalages des premières lettres.

Correction : il suffit d'utiliser les fonctions des questions précédentes en utilisant le modulo cette fois pour l'effet de bord.

```
let decode message = let n = String.length message in
    let mes_decode = Array.make n 'a' in
    for i = 0 to n-1 do
        mes_decode.(i)<-c_of_i(modulo(i_of_c message.[i] - 3), 26))
    done;
    mes_decode;;
```

5) Afin de brouiller les pistes, on peut changer le nombre de décalages.

Proposer une fonction de type `code2 : char -> string -> char array` qui en entrée reçoit un caractère (dont le rang est noté $k \in [\![0;25]\!]$), une chaîne de caractères et qui renvoie le tableau avec les caractères décalés de k rangs vers la "droite" dans l'alphabet usuel.

Correction : on reprend le principe du codage de la question 3)

```
let code2 cle message = let k = i_of_c cle in
    let n = String.length message in
    let mes_code = Array.make n 'a' in
    for i = 0 to n-1 do
        mes_code.(i)<-c_of_i((i_of_c message.[i] + k) mod 26)
    done;
    mes_code;;
```

6) Ce cryptage de données a un inconvénient. Si le texte codé est assez long, on peut calculer la fréquence d'apparition des lettres. Or dans la langue française la fréquence du e est supérieure à toutes les autres lettres. On peut donc savoir comment est codé le e puis en déduire le décalage.

Proposer une fonction de type `frequence : string -> int array` qui en entrée reçoit une chaîne de caractères et qui renvoie un tableau de 26 entiers comportant les 26 nombres d'apparitions des lettres dans la chaîne de caractères.

Correction : il suffit de remplir le tableau demandé en parcourant le message

```
let frequence message =
    let tab = Array.make 26 0 and n = String.length message in
    for i = 0 to n-1 do
       tab.(i_of_c message.[i]) <- tab.(i_of_c message.[i])+1
  done;
  tab;;
```

7) Pour résoudre partiellement, la difficulté on peut utiliser un codage avec une chaîne de caractères appelée clé de cryptage. Le principe est de coller sous le texte à coder une répétition de la clé et d'effectuer le décalage donner par la lettre de la clé. Par exemple, on prend la clé `taf` et on veut coder `reussir`.

message	r	e	u	s	s	i	r
clé répétée	t	a	f	t	a	f	t
décalages	+19	+0	+5	+19	+0	+5	+19
message codé	k	e	z	l	s	n	k

a) Quel est l'intérêt de ce type de cryptage ?

Correction : Une même lettre peut être codé de plusieurs façons différentes ce qui limite l'efficacité de la recherche des fréquences de lettres surtout si la clé est longue voire aussi longue que le texte lui-même.

b) Proposer une fonction de type `code3 : string -> string -> char array` qui en entrée reçoit deux chaînes de caractères (la clé et le message) et qui renvoie le message codé dans un tableau, avec ce principe. On prendra en compte les effets de bords comme sur l'exemple où il a fallu ajouter un t pour compléter.

Correction : il faut faire attention à la longueur de la clé pour pouvoir utiliser la répétition :

```
let code3 message cle = let n = String.length message and p = String.length cle in
     let tab = Array.make n 'a' in
     for i = 0 to (n-1) do
        tab.(i)<- c_of_i ((i_of_c message.[i] + i_of_c(cle.[i mod p])) mod 26)
      done;
      tab;;
```

Exercice 4

Le but de l'exercice est d'étudier diverses méthodes de calcul du maximum des sommes d'éléments consécutifs d'un tableau donné (non trié) de nombres entiers relatifs.

t et n étant donnés, t de type tableau d'entiers et n de type entier (avec n plus petit que la longueur du tableau supposé suffisamment grande), on s'intéresse aux n premières valeurs contenues dans t et l'on cherche la plus grande des sommes de la forme $\sum_{k=g}^{d} t.(k)$, où les entiers g et d vérifient $0 \leqslant g \leqslant d \leqslant n-1$.

Par exemple, pour $n = 6$:
— si les 6 premières valeurs sont $-4, 8, -1, -3, 5, -2$, le résultat est 9, obtenu pour $g = 2$ et $d = 5$;
— si les 6 premières valeurs sont $-4, -1, -3, -2, -7, -5$, le résultat est -1, obtenu pour $g = d = 2$.

Cette valeur sera notée $S(t, n)$; elle est bien définie, en tant que plus grand élément d'un ensemble fini totalement ordonné.

Pour cet exercice on pourra utiliser entre autre, la fonction **`max`** *qui donne le plus grand de deux entiers. Cette fonction est currifiée.*

1) Algorithme naïf

La fonction ci-dessous (que l'on ne demande pas de justifier) détermine $S(t, n)$ en calculant successivement toutes les sommes décrites ci-dessus :

```
# let SomMax1 t n=
    let smax= ref t.(0) in
      for g=0 to n-1 do
        for d=g to n-1 do
          let som= ref t.(g) in
            for k=g+1 to d do som:=!som+t.(k) done;
            smax:=max !smax !som;
        done;
      done;
    !smax;;
```

Montrer que le nombre $T_1(n)$ d'additions d'entiers effectuées au cours de l'appel de `SomMax1tn` est un $\Theta\left(n^3\right)$.

Correction : La boucle interne effectue $d - g$ additions, d'où

$$T_1(n) = \sum_{g=0}^{n-1}\sum_{d=g}^{n-1}(d-g) = \sum_{g=0}^{n-1}\sum_{k=0}^{n-g} k = \sum_{j=0}^{n-1}\sum_{k=0}^{j} k = \sum_{j=0}^{n-1}\frac{j(j+1)}{2} = \frac{1}{2}\frac{(n-1)n(2n-1)}{6} + \frac{1}{2}\frac{(n-1)n}{2}$$

soit $\boxed{T_1(n) = \dfrac{n(n-1)(n+1)}{6} \sim \dfrac{n^3}{6} = \Theta\left(n^3\right)}$

2) Première amélioration

On peut supprimer l'une des boucles `for` imbriquées de la fonction `SomMax1`, en mettant à jour la variable `smax` au fur et à mesure du calcul des $\sum_{k=g}^{d} t.(k)$, pour g fixé et d variant de g à $n-1$.

Justifier cela en écrivant une fonction d'en-tête `SomMax2 t n` renvoyant $S(t, n)$, au prix d'un nombre $T_2(n)$ d'additions d'entiers, qui devra être un $\Theta\left(n^2\right)$ et que l'on calculera.

Correction : L'idée est d'essayer d'éviter les calculs redondants dans les deux boucles internes de `SomMax1` : on va mettre à jour la variable `smax` **à l'intérieur** de la boucle qui calcule $\sum_{k=g}^{d} t\,[k]$:

```
# let SomMax2 t n=
    let smax= ref t.(0) in
```

```
  for g=0 to n-1 do
    let som:=ref t.(g) in
    smax:=max !smax !som;
    for d=g+1 to n-1 do
      som := !som+t.(d);
      smax:=max !smax !som;
    done;
  done;
!smax;;
```

La terminaison est banale puisque les boucles sont inconditionnelles.

`smax` est initialisé avec la valeur `t.(1)`, à la suite de quoi je calcule toutes les sommes à considérer en mettant à jour `smax` à chaque nouveau calcul : à la sortie `smax` contient bien $S(t,n)$.

Attention ! Ne pas initialiser `smax` par 0, pour le cas où tous les éléments du tableau seraient négatifs.

Pour g fixé dans $\mathbb{N}_n$, la boucle interne du programme précédent effectue $n-g$ additions, d'où : $T_2(n) = \sum_{g=0}^{n-1}(n-g) = \sum_{k=1}^{n-1} k$ soit $\boxed{T_2(n) = \frac{n(n-1)}{2} = \Theta\left(n^2\right)}$

3) Une version récursive

On remarque que, pour $n \geq 2$, $S(t,n)$ est le plus grand des $n+1$ nombres suivants : $S(t,n-1)$ et les $\sum_{k=g}^{n-1} t.(k)$, $g \in [\![0, n-1]\!]$.

En déduire une fonction récursive ou utilisant une fonction auxiliaire récursive, d'en-tête `SomMax3 t n` renvoyant $S(t,n)$.

Préciser le nombre $T_3(n)$ d'additions d'entiers effectuées au cours de l'appel de `SomMax3 t n` .

Correction :

```
let somMax3b t n =
          let rec aux som smax g =
               if g = -1 then smax
               else
                 let s = som + t.(g) in aux s (max s smax) (g-1)
          in
          let rec aux1 n =
          if n= 1 then t.(0) else begin
               let s = aux t.(n-1) t.(n-1) (n-2) in
               max s (aux1 (n-1)) end
          in aux1 n;;
```

⋆ La terminaison est justifiée par la décroissance stricte de l'entier naturel n à chaque appel récursif et l'arrêt lorsque $n = 1$.

⋆ La correction est justifiée par la remarque de l'énoncé, accompagnée d'une récurrence : la boucle `FOR` place dans `smax` la plus grande des sommes $\sum_{k=g}^{n} t[k]$, $g \in [\![1,n]\!]$. Il n'y a plus qu'à renvoyer `Max smax (SomMax3 t (n-1))`.

⋆ Le nombre d'additions de réels $T_3(n)$ vérifie ici

$$T_3(1) = 0 \quad \text{et} \quad \forall n \geq 2 \quad T_3(n) = n - 1 + T_3(n-1),$$

d'où — par une récurrence immédiate — $T_3(n) = \sum_{k=1}^{n}(k-1)$, soit $\boxed{T_3(n) = \frac{n(n-1)}{2}}$

Exercice 5 : manipulation de listes

Les questions de cet exercice sont indépendantes.

1. Écrire une fonction `sommeliste` qui prend en entrée une liste d'entiers et renvoie la somme de ses termes.
   Correction : cf TD 2
2. Écrire une fonction `premierneg` qui prend en entrée une liste d'entiers et renvoie son premier terme strictement négatif (si un tel terme existe).
   Correction :

```
let rec premierneg lst = match lst with
            | [] -> failwith "liste vide ou pas de nombres négatifs"
            | t::q when t < 0 -> t
            | t::q -> premierneg q;;
```

3. Écrire une fonction `enumdecr` qui prend en entrée un entier `n` et renvoie la liste `[n ; n-1 ; n-2 ; ... ; 1 ; 0]`.
   Correction :

```
let enumdecr n = let rec aux k  = match k with
                        | 0 -> [0]
                        | k -> k::aux (k-1)
                      in aux n;;
```

   Autre version

```
let enumdecr n = let rec aux k lst = match k with
                        | k when k = n -> n::lst
                        | k -> aux (k+1) (k::lst)
                       in aux 0 [];;
```

4. Écrire une fonction `enumcroi` qui prend en entrée un entier `n` et renvoie la liste `[0 ; 1 ; 2 ; ... ; n-1 ; n]`. Cette fonction devra s'exécuter en $O(n)$ opérations sans appeler la fonction `List.rev`.
   Correction :

```
let enumcroi n = let rec aux k  = match k with
                  | 0 -> [n]
                  | k -> (n-k)::aux (k-1)
                  in aux n;;
```

   Autre version

```
let enumcroi n = let rec aux k lst = match k with
                  | 0 -> 0::lst
                  | k -> aux (k-1) (k::lst)
                  in aux n [];;
```

Exercice 6

1) À une liste `a=[`a_0`; ...;` a_n`]` d'entiers entre 0 et 9 on associe l'entier $\sum_{i=0}^{n} a_i 10^i$ et réciproquement à un entier positif on associe la liste de ses chiffres en base 10 à l'envers sans 0 à la fin, en convenant que la liste associée à 0 est `[]`, et que réciproquement à `[]` on associe 0.
Ainsi à `[1; 5; 7; 2]` est associé l'entier 2751, et réciproquement à 2751 est associée la liste `[1; 5; 7; 2]`.
Dans la suite toutes les listes sont formées d'entiers entre 0 et 9, et éventuellement vides.

a) Écrire une fonction récursive `valeur : int list -> int` telle que `valeur a` renvoie l'entier associé à `a`.
Correction : on utilise le fait que $a_n a_{n-1}..a_2 a_1 a_0 = a_0 + 10(a_1 + 10(a_2 + 10(...)))$.

```
# let rec valeur lst = match lst with
        | [] -> 0
        | a::q -> a+10*(valeur q);;
```

b) Écrire une fonction récursive `liste : int -> int list` telle que `liste n` renvoie la liste associée à `n`, `n` étant supposé positif.

Correction : on utilise le reste et le quotient de la division euclidienne par 10 pour écrire la récursivité.

```
# let rec liste n = match n with
        | 0 -> []
        | _ -> (n mod 10)::(liste (n/10));;
```

2) Ici, à une liste non vide d'entiers `[`a_0`; ...; `a_n`]` on associe le polynôme $a_0 + a_1X + ... + a_nX^n$, et à un polynôme $a_0 + a_1X + ... + a_nX^n$ avec $a_n \neq 0$, on associe la liste `[`a_0`; ...; `a_n`]`. On associe au polynôme nul la liste `[]` et inversement à `[]` on associe le polynôme nul.

a) Écrire une fonction `val0 : int list -> int` de sorte que `val0 a` renvoie la valeur en 0 du polynôme associé à `a`. C'est-à-dire $P(0)$ si `a` est associée à $P(X)$.

Correction : la valeur en 0 correspond à a_0 donc :

```
# let val0 lst = match lst with
          | [] -> 0
          | a::q -> a;;
```

b) Écrire une fonction `val1 : int list -> int` de sorte que `val1 a` renvoie la valeur en 1 du polynôme associé à `a`. C'est-à-dire $P(1)$ si `a` est associé à $P(X)$.

Correction : on a $P(1) = \sum_{k=0}^{n} a_k$ d'où

```
# let rec val1 lst = match lst with
          | [] -> 0
          | a::q -> a + val1 q;;
```

c) Écrire une fonction `derivation : int list -> int list` de sorte que `derivation a` renvoie la liste associée à la dérivée du polynôme associé à `a`.

Correction : on utilise une fonction auxiliaire qui permet de gérer le degré. On fait de plus attention au fait que la dérivée d'un polynôme constant est nulle.

```
# let derivation lst = match lst with
          | [] -> []
          | a -> let rec aux b k =
                   if b = [] then []
                   else (k*(hd b))::(aux (tl b) (k+1))
                 in aux (tl a) 1;;
```

d) Écrire une fonction `integration : int list -> int list` de sorte que `integration a` renvoie la liste associée à la primitive nulle en 0 du polynôme associé à `a`. *On supposera que la primitive est encore à coefficients entiers.*

Correction : il faut gérer le fait que la primitive de la fonction nulle cherchée est la fonction nulle. De plus il y a une erreur d'énoncé car on ne peut pas garder des coefficients entiers du fait de la division.

```
# let integration lst = match lst with
          | [] -> []
          | a -> let rec aux b k =
                   if b = [] then []
                   else ((hd b)/(k+1))::(aux (tl b) (k+1))
                 in 0::(aux a 0);;
```

e) Écrire une fonction récursive `somme : int list -> int list -> int list` de sorte que `somme a b` renvoie la liste associée à la somme des polynômes associés à `a` et `b`.

(remarque : les listes `a` et `b` ne sont pas supposées de mêmes longueurs)

Correction : il suffit de sommer les termes des deux listes tant qu'il y a des éléments au même niveau

```
# let rec somme l1 l2 = match (l1,l2) with
          | lst,[] -> lst
          | [],lst -> lst
          | a::p,b::q -> (a+b)::(somme p q);;
```

f) Écrire une fonction récursive `produit : int list -> int list -> int list` de sorte que `produit a b` renvoie la liste associée au produit des polynômes associés à `a` et `b`.

(même remarque)

Correction : on va utiliser une fonction auxiliaire qui multiplie un polynôme par un coefficient et on gère l'augmentation du degré.

```
# let rec produit l1 l2 = match (l1,l2) with
           | lst,[]-> []
           | [],lst -> []
           | a::q,lst -> let rec aux b l = match b,l with
                         | 0,_ -> []
                         | _,[] -> []
                         | b,t::r -> (b*t)::(aux b r) in
                 somme (aux a lst) (0::(produit q lst));;
```

Annexes : fonctions utilisables

Pour les différents exercices :
— `Array.make n a` : créer un tableau de taille n dont tous les éléments sont égaux à `a`
— `Array.length` : donne la longueur d'un tableau
— `p.(i)` : donne l'élément du tableau d'indice `i` (l'indice commence à 0)
— `p.(i)<- a` : pour mettre la valeur de `a` dans le tableau à l'indice `i`

Exercice 1

cf ce qui précède

Exercice 2

— / pour le quotient de la division euclidienne de deux entiers, `mod` pour le reste de la division euclidienne
— `print_int` : affiche un entier
— `incr` : fonction qui incrémente d'un une référence entière

Exercice 3

cf début d'annexe

Exercice 4

cf début d'annexe

Exercice 5

Aucune fonction sur les listes prédéfinie n'est autorisée à par, si besoin, `List.hd`, `List.tl`, le constructeur `::`.

Exercice 6

idem que pour le 5