Exercices dirigés no. 1 en Ocaml
Conception et développement OO (19357)

Année 2003-2004

Premiers pas

1. Pour chacune des expressions suivantes expliquez les messages affichés par Ocaml. Expliquez pourquoi le typage échoue, ou réussit, selon le cas.

```
#  1 + 2;;
- : int = 3

#  1 + 2.0;;
       ^^^
This expression has type float but is here used with type int

#  [1] = [1;2];;
- : bool = false

#  1 = [1];;
       ^^^^
This expression has type 'a list but is here used with type int

# let x = 1 in x + 3;;
- : int = 4

#  let x = 3 in x^x;;
                ^^
This expression has type int but is here used with type string

# let x = [1] in 2::x;;
- : int list = [2; 1]

# ["bonjour"; "adieu"];;
- : string list = ["bonjour"; "adieu"]

# let f(x) = x*2;;
val f : int -> int = <fun>

# f(3);;
- : int = 6

# let f(x) = x+1 in f(3);;
- : int = 4

# f(3);;
```

```
- : int = 6
```

2. Donnez les messages affichés par Ocaml, et en cas d'erreur, expliquez.

```
# [1] = [[1]];;

# [1;[2]];;

# let x = 3 in let y = 4 in x+y;;

# let f(x) = x + 1 in f(3);;

# let f(x) = x + 1 in f(true);;
```

3. (Liaison statique). Expliquez les messages affichés par Ocaml, et signalez, pour chaque utilisation de variable, ou d'appel de fonction, quelle est la définition (liaison) qui lui correspond:

   (a) Liaison de constantes:

```
#   let x = 2;;
- : int = 2

# let x = 5 in x+3;;
- : int = 8

# x+3;;
- : int = 5

# let f(y) = y+x;;
val f : int -> int = <fun>

# f(2);;
- : int = 4

# let x = 7;;
val x : int = 7

# x+3;;
- : int = 10

# f(2);;
- : int = 4
```

   (b) Liaison de fonctions:

```
# let f(x) = x+3;;
val f : int -> int = <fun>

# f(3);;
- : int = 6

# let g(y) = 2*f(y);;
val g : int -> int = <fun>

# g(5);;
- : int = 16
```

```
# let f(x) = x*x;;
val f : int -> int = <fun>

# f(3);;
- : int = 9

# g(5);;
- : int = 16
```

Fonctions en argument

Expliquez les messages affichés:

```
# let double_du_succ(f,y) = 2*f(y+1);;
val double_du_succ : (int -> int) * int -> int = <fun>

# let f1 (x) = x-2;;
val f1 : int -> int = <fun>

#  double_du_succ(f1,3);;
- : int = 4

# let g1(z) = z*z;;
val g1 : int -> int = <fun>

#  double_du_succ(g1,3);;
- : int = 32

# let compare_res(f,g,x) = f(x+2) > g(x)+2;;
val compare_res : (int -> int) * (int -> int) * int -> bool = <fun>

# compare_res(f1,g1,2);;
- : bool = false
```

Utilisation de tableaux

On souhaite manipuler des polynômes à une variable à l'aide des tableaux. Un polynôme de degré N, sur une variable X, et avec coefficients entiers peut être représenté par un tableau T(0..N) d'entiers avec indices entiers: chaque case du T(i) contient le coefficient d'exposant i. Par exemple, le polynôme: $3-4x^2+x^3$ peut être représenté par le vecteur:

T =

3	0	-4	1
0	1	2	3

Avec cette modélisation des données:

1. Définissez une constante avec le polynôme de l'exemple plus haut.
2. Écrire une fonction qui calcule la valeur d'un polynôme pour une valeur de X donnée.
3. Écrire un sous-programme qui affiche le contenu d'un polynôme sous forme de terme mathématique.
4. Écrire un sous-programme qui calcule la somme de deux polynômes. Lorsque les dégrés sont différents, on considère que tout coefficient au délà du dégré du polynôme est égal à zéro.

Utilisation de listes

1. Écrire une fonction qui additionne les éléments d'une liste d'entiers.

2. Écrire une fonction qui prend en arguments une liste `l` d'entiers et un nombre entier `x`, et renvoie en résultat la liste où tous les éléments de `l` sont augmentés de `x`.
3. On souhaite modéliser les polynômes creux à l'aide des listes. Un polynôme creux est un polynôme dont une grande partie des coefficients sont nuls. Dès lors, plutôt que d'employer des tableaux avec beaucoup de cases à zéro, on prefère les modéliser à l'aide des listes, où seuls les coefficients non nuls, ainsi que leurs dégrés sont représentés dans la liste.

   (a) Faites un choix de représetation pour polynômes creux, et utilise-la pour définir le polynoôme $1 + 2X^{34} - 7X^{1000}$.

   (b) Donnez l'équivalent des fonctions qui manipulent les polynômes représentés avec tableaux (polynômes pleins), pour les polynômes creux.

Polymorphisme

1. Considérez les définitions Ocaml suivantes:

```
# let ppe(x,y) = x <= y;;
val ppe : 'a * 'a -> bool = <fun>

#  let compare (f,g,x)  = f(x) > g(x);;
val compare : ('a -> 'b) * ('a -> 'b) * 'a -> bool = <fun>

# let rec list_lenght l =
  match l
  with  []      -> 0
   |   a::reste -> 1 + list_lenght reste;;
val list_lenght : 'a list -> int = <fun>

# let rec list_member (x, l) =
  match l
  with  []      -> false
   |   a::reste -> a=x or list_member(x, reste);;
val list_member : 'a * 'a list -> bool = <fun>
```

   (a) Expliquez le typage donné par Ocaml à chacune de ces définitions. Donnez deux exemples d'utilisation. Pour chaque exemple, expliquez quelle instantiation est faite pour les variables de type de la fonction.

   (b) Les appels suivants sont-ils correctement typés? Expliquez pourquoi.

```
# list_member (1, [3;4]);;

# list_member (1, ["Bonjour"]);;

# list_lenght ["Bonjour"];;
```

   (c) La fonction suivante, permet de trier (par insertion) une liste de valeurs comparables avec l'opérateur de comparaison `>`. Donnez une version polymorphe adaptée aux liste de n'importe quel type d'éléments. Donnez ensuite deux exemples d'appels de manière à trier des listes d'entiers en ordre croissant, et des listes de monômes, par l'ordre des coefficients.

```
# let rec insert_1 (x,l) =
  match l
  with  []     -> [x]
   |  a::reste -> if x < a then x::a::reste
                  else a::(insert_1 (x,reste));;
val insert_1 : 'a * 'a list -> 'a list = <fun>
```

```
# let rec tri_insertion_1 l =
  match l
  with   []     -> []
   |   a::reste -> insert_1 (a, (tri_insertion_1 reste));;
val tri_insertion_1 : 'a list -> 'a list = <fun>
```

Types sommes

On souhaite travailler dans un mêmes programme, avec des polynômes à une variable, creux ou pleins. Il est possible de réunir les deux representations dans un mm̂e type somme, défini (par exemple) par:

```
type monome = {Coeff: int; Degre: int};;

type polynome = Plein of int vect
              | Creux of monome list;;
```

Donnez les définitions qui permettront de calculer la valeur d'un polynôme, de l'afficher, et d'additionnner deux polynômes, indépendamment de leur representation.