Université Paris 6
Licence d'informatique
Maîtrise de Sciences et Techniques
(Experts en Systèmes Informatiques)

Module de Programmation
ANNÉE 2000-2001

Travaux dirigés 6

Exercice 1 Typer la fonction :

```
let rec foo = function n ->
  if (n = 0)
  then 0
  else
   if (n = 1)
   then 1
   else (2 * foo(n-1)) - foo(n-2)
```

Que calcule-t-elle ?

Exercice 2 Typer la fonction :

```
let bar = function test -> function si -> function sinon ->
  if test
  then hd(si)
  else tl(sinon)
```

On rappelle que :

- `hd :` $\forall\alpha\ \alpha$ `list ->` α
- `tl :` $\forall\alpha\ \alpha$ `list ->` α `list`

Exercice 3 Soit la déclaration :

```
let comp = function f -> function g -> function x -> f(g x)
let double_applic = function f -> function x -> f(f x) ;;
```

1. faire le typage de `comp` et de `double_applic`.
2. Quel est le résultat de l'application de `double_applic` à `(function x -> char_of_int (1 + int_of_char x))` ?
3. Typer cette application.
4. Redéfinir `double_applic` à partir de `comp`.
5. En déduire une fonction qui prend en argument une fonction qui prend en argument une fonction F, un entier positif n,une valeur x et retourne

$$F^n(x) = F(\ldots F(x)) \quad n \text{ fois}$$

Exercice 4 Soient les types suivants :

```
#open "float" ;;
type cube = {arete : float} ;;
type  parallelepipede =
        {longueur : float ; largeur : float ; hauteur : float} ;;
type prisme = {aret_p : float; long_p : float} ;;
let volume_prisme = function  p ->
 p.aret_p * p.aret_p * p.long_p / 2 ;;
type cylindre = {rayon : float; haut : float} ;;
```

- Écrire une fonction `volume_cube` qui calcule le volume d'un `cube`.

- Écrire une fonction `volume_par` qui calcule le volume d'un `parallelepipede`.
- Écrire une fonction `volume_prisme` qui calcule le volume d'un `prisme`.
- Écrire une fonction `volume_cyl` qui calcule le volume d'un `cylindre`.

On ajoute le type :

```
type solide =
        | Cyl of cylindre
        | Pri of prisme
        | Cub of cube
        | Par of parallelepipede;;
```

Écrire une fonction `volume_solide` qui calcule le volume d'un `solide`.

Exercice 5 Les listes d'associations

Une *association* est un couple composé d'une *clé* et d'une *valeur*. Notre objectif est de définir une fonction `assoc` recherchant la valeur associée à une clé dans une liste d'associations.

1. Sans utiliser le type prédéfini `list`, définir le type paramétré `list_assoc`. Écrire une valeur de ce type.
2. Écrire la fonction `assoc` prenant une clé et une liste d'associations et retournant “une” valeur associée à la clé dans la liste. La fonction lèvera l'exception prédéfinie `Not_found` si la clé est absente. Faire l'arbre de typage de cette fonction.
3. Écrire la fonction `assoc_all` retournant toutes les valeurs associées à une clé donnée. En cas d'absence, on continuera à lever l'exception `Not_found` et on prendra soin de retourner les valeurs dans l'ordre où elles apparaissent dans la liste d'association.
4. Une liste d'association permet de représenter une fonction qui n'est définie que pour un nombre fini de valeurs. Écrire une fonction `function_of_assoc_list` qui prenne en argument une liste d'association l et qui retourne une fonction F telle que $F(a) = b$ si a est la prenmière clé associée à la valeur b dans la liste l.

Exercice 6 Typer les déclarations suivantes :

```
let curry = function f -> function a -> function b -> f (a,b);;
let uncurry = function f -> function a -> f (fst a) (snd a);;
```

Écrire les fonctions `curry_3` et `uncurry_3` analogues à `curry` et `uncurry` pour 3 arguments. Peut-on écrire des fonctions `curry_n` et `uncurry_n` générales pour un nombre quelconque d'arguments ?

Exercice 7 Typer les fonctions suivantes :

```
 let rec nulle1 = function x -> (fst x)+(nulle1 (snd x));;
 let rec nulle2 = function x -> (fst x, nulle2 (snd x));;
```

Exercice 8 La fonction définie ci-dessous est-elle typable ? Pourquoi ?

```
 let f = function g1 -> function g2 -> function x ->
   ((g1 (fst x))&(snd x), (fst x), (g2 (snd x)));;
```