TD 3 (Types élaborés, paramétrages)

Exercice 1. Pour chacune des expressions cartésiennes suivantes, préciser l'arité de chaque fonction impliquée, puis définir ensuite en Caml une forme curryfiée de `f`.

- `f(h(x,y),g(z,t))`
- `f(h,g(x(h),y),z)`
- `f(g,h(x,y),z(x),t)`

Exercice 2.

1. Créer un type somme appelé `chiffre` (il y a dix chiffres, de `zéro` a `neuf`).

2. Créer un type de données pour un `panier` contenant un certain nombres de fruits identiques : des pommes, des poires, ou bien des pêches. Définir un panier de 10 pommes, un panier de 6 poires, un panier vide.

3. Créer un type somme appelé `operateur` pour les quatre opérateurs entiers plus, moins, multiplie, divisepar. Définir une fonction `calculer` qui prend en paramètre un opérateur et deux entiers et calcule l'opération correspondante.

4. Créer un type somme appelé `complexePur` pour lequel un nombre est soit un réel (avec une valeur réelle), soit un imaginaire pur (avec une valeur imaginaire).

Exercice 3.

1. Définir un type `dble_ent` produit cartésien de deux entiers, puis un type `dbl_reel` produit cartésien de deux réels. Définir enfin un type `dbl_ec` produit cartésien des deux types précédents. Définir une fonction qui transforme un `dbl_ec` en un `dbl_reel` en multipliant membre a membre entier et réel de la paire.

2. Créer un type produit `point2` pour les points du plan. En utilisant les fonctions `fst` et `snd`, créer deux fonctions qui retournent l'abscisse et l'ordonnée d'un point du plan. On s'assurera que ces fonctions ne s'appliquent qu'au type `point2`.

3. Créer un type produit `point3` pour les points de l'espace. Peut-on utiliser la fonction `fst` pour obtenir l'abscisse d'un point ? Définir une fonction qui associe a un point P=(x,y,z) sa projection P′=(x,y+z) qui est un point du plan.

4. Créer un type produit `vecteur` pour les vecteurs de l'espace a trois dimensions. Créer une fonction qui à deux points A et B de l'espace associe le vecteur $\overrightarrow{AB}$.

Exercice 4. *Une personne simple.*
Créer un type produit cartésien `individu` regroupant un prénom, un nom et un age. Créer un individu `d` de ce type : Tsing Tao agé de 98 ans. Comment Caml distinguera le nom du prénom ?

Exercice 5. *Une personne plus élaborée.*

1. Créer un type `nom` comme un type somme (a une composante) contenant une valeur chaîne de caractères. De même pour un type `prenom`.

2. Créer un type `age` comme un type somme contenant une valeur entière.

3. Créer, avec les types précédents, un type `personne` regroupant un prénom, un nom et un age.

4. Créer une personne `p` : Alice DURAN, 20 ans.

5. Lorsque l'on déclare une nouvelle personne, comment Caml distingue le prénom, le nom et l'age ?

6. Créer deux fonctions qui associent à une `personne` respectivement son `nom` et son `age`.

Exercice 6. Ecrire une plus petite* fonction curryfiée ayant le typage suivant :

• `'a->int`	• `'a->'b->'a`	• `'a->('a->'b)->'b`
• `'a->'a`	• `'a->'b->'b`	• `'a->('a->int)->int`
• `'a->unit`	• `'a->int->int`	• `int->(int->'a)->'a`
• `int->int`	• `'a->int->'a`	• `('a->'b)->('b->'c)->'a->'c`

* ***Important*** *: ces fonctions ne devront pas avoir de paramètres constants ni de forçage de type.*

Exercice 7. Typer (càd donner le type de) les fonctions suivantes :

- `function x -> x+1;;`
- `fun x -> fun y -> fun f -> f x y;;`
- `fun x -> fun y -> fun f -> f (x y);;`
- `fun x -> fun y -> fun f -> f(x,y);;`
- `fun f g x y -> f(g(x,y),y);;`
- `fun f g x y -> f(x,g(y))+g(y);;`

Exercice 8.

1. Ecrire la fonction `curry` qui curryfie une fonction f, sachant que f prend un couple d'arguments. Typez (cad *donner le type de*) la fonction `curry`.
2. Ecrire la fonction `uncurry` qui décurryfie une fonction f, sachant que f prend deux arguments. Typez (cad donner le type de) la fonction `uncurry`.

Exercice 9.

1. Que retourne Caml pour les commandes suivantes (où `Plus` est de type operateur) :

```
let prefixe x y z = if x=Plus then y+z else y-z;;
let addition = prefixe Plus;;
addition 5 7;;
let infixe x y z = if y=Plus then x+z else x-z;;
let suffixe x y z = if z=Plus then x+y else x-y;;
```

2. Quelle différence y a-t-il entre les fonctions prefixe, infixe et suffixe ?
3. Compléter de deux façons différente la fonction `prefixe` afin qu'elle prenne en compte les autres valeurs du type `operateur` : d'une part en utilisant seulement `if then else` et d'autre part en utilisant uniquement `match`.

Exercice 10. Définir au moyen d'un `match`, les fonctions suivantes.

- `nullite` : retourne 0 si l'entier n est nul, 1 sinon.
- `unoudeux` : retourne 1 si l'entier n est égal a 1 et 2 si l'entier n est égal a 2.
- `doublenul` : retourne 0 si l'entier x est nul, 1 si le réel y est nul, et 2 sinon.
- `minuscule` : retourne 'v' si le premier caractère de la chaîne s est une voyelle minuscule, '-' sinon.
- `position` : retourne "origine" si le point p du plan est (0,0), "abscisse" si p est sur l'axe des abscisses, ou "ordonnee" si p est sur l'axe des ordonnées.

Exercice 11.

1. La fonction logique XOR (ou exclusif) renvoie faux si le couple de booléens en entrée a la même valeur et vrai sinon. écrivez quatre versions d'une fonction XOR en Caml :
– en utilisant un `if`,
– en utilisant un `match` sans choix générique,
– en utilisant un `match` avec choix générique _,
– sans utiliser ni match ni if.
2. Mêmes questions pour les fonctions logiques NOR (not-or), puis NAND (not-and).

Exercice 12. Définir une fonction qui donne le nombre de fruit contenu dans un objet de type `panier` (type défini précédemment).

Exercice 13. Peut-on toujours remplacer un `if` par un `match` ? Et réciproquement ?