Université Pierre et Marie Curie	option Informatique 2002−2003 feuille de TP numéro 4	**Langage *Objective Caml***

Exercice 1 Introduire le type suivant, qui représente une "case mémoire" contenant un entier :

```
type case_mem = { mutable contenu: int }
```

Faire les définitions suivantes :

```
let c1={contenu = 5};;
let c2=c1;;
let c3={contenu = c1.contenu};;
```

A ce stade, on aura du obtenir :

```
val c1 : case_mem = {contenu=5}
val c2 : case_mem = {contenu=5}
val c3 : case_mem = {contenu=5}
```

Affecter l'entier 6 à `c1`. Quels sont les contenus de `c2` et `c3` ? Affecter l'entier 7 à `c2`. Quels sont les contenus de `c1` et `c3` ? Affecter l'entier 8 à `c3`. Quels sont les contenus de `c1` et `c2` ?

Exercice 2 A partir du type `case_mem` précédent. Définir la fonction `incr_affiche` qui prend une case mémoire, incrémente son contenu, puis retourne son contenu.

```
incr_affiche : case_mem -> int
```

Vérifier votre fonction en appliquant quatre fois de suite votre fonction sur `c1`. Quels sont les contenus de `c2` et `c3` ?

Exercice 3 A partir de la fonction précédente, créer la fonction `compteur` qui retourne la suite successive des entiers à chaque application (à partir de 1) :

```
val compteur : unit -> int = <fun>
# compteur ();;
- : int = 1
# compteur ();;
- : int = 2
# compteur ();;
- : int = 3
```

Exercice 4 Reprendre les 3 exercices précédents en utilisant un tableau d'entier à 1 élément, à la place du type `case_mem`. On écrira par exemple :

```
let c1=[| 5 |];;
let c2=c1;;
let c3=[| c1.(0) |];;
```

Exercice 5 Ecrire une fonction qui prend un tableau d'entier et incrémente tous ses éléments de leur indice dans le tableau. L'élément d'indice i dans le tableau sera incrémenté de i. On utilisera une boucle "`for`".

```
incr_tab : int array -> unit
```

Tester votre fonction sur un tableau de 10 éléments initialisés à 0. Que se passe-t-il si on applique cette fonction sur le tableau vide ?

Exercice 6 Ecrire une fonction qui prend un tableau d'entier A et retourne qui retourne un tableau d'entier B de même longueur que A, vérifiant, pour tout i, $B[i] = i + A[i]$. Le tableau A ne devra pas être modifié au cours de cette opération.

```
incr_tab2 : int array -> int array
```

Tester votre fonction sur un tableau de 10 éléments initialisé à 0. Que se passe-t-il si on applique cette fonction sur le tableau vide ?

Exercice 7 Faire les 2 déclaration suivantes :

```
let m1 = Array.make_matrix 2 5 0;;
let m2 = Array.make 2 m1.(1);;
```

On doit avoir :

```
val m1 : int array array = [|[|0; 0; 0; 0; 0|]; [|0; 0; 0; 0; 0|]|]
val m2 : int array array = [|[|0; 0; 0; 0; 0|]; [|0; 0; 0; 0; 0|]|]
```

Appliquer `incr_tab` à `m1.(1)`. Quels sont les éléments de `m1` ? Quels sont les éléments de `m2`. Exécuter :

```
m2.(1) <- m1.(0);;
incr_tab m1.(1);;
incr_tab m1.(0);;
```

Quels sont les éléments de `m1`, et quels sont ceux de `m2` ?

Exercice 8 Créer un tableau `t` de 10 éléments de type `case_mem`, initialisé à 0. Affecter 5 comme contenu au premier élément de ce tableau. Quel est le contenu des éléments de `t` ? Ecrire une fonction qui prend deux entiers `n` et `m`, et qui crée un tableau de `n` éléments de type `case_mem`, dont le contenu aura été affecté à `m`, mais tels que chacune de ces "cases mémoires" soit distincte (ou indépendante des autres).

```
cree_tab : int -> int -> case_mem array
```

Tester cette fonction.

Exercice 9 Exécuter les lignes suivantes :

```
let x = ref 5;;
x.contents <- x.contents + 2;;
x;;
```

Écrire la déclaration du type `ref`. Écrire les fonctions correspondant aux opérateurs "`ref`", "`!`" et "`:=`" :

```
ref : 'a -> 'a ref
deref : 'a ref -> 'a
assign : 'a ref -> 'a -> unit
```

Exercice 10 Ecrire la fonction `min_maj` qui transforme toutes les minuscules d'une chaine de caractères en majuscules et réciproquement. Les autres caractères resteront inchangés. On profitera du fait que les chaines sont codées comme des sortes de tableaux de caractères, et des fonctions `Char.uppercase` et `Char.lowercase` de la bibliothèque.

```
min_maj : string -> unit
```

Écrire la fonction qui effectue la transformation précédente, mais en retournant une nouvelle chaine, et en laissant celle prise en argument inchangée.

```
min_maj2 : string -> string
```