Université Pierre et Marie Curie	option Informatique 2002−2003 feuille de TP numéro 1	**Langage *Objective Caml***

Il est normal de ne pas pouvoir finir une feuille de TP entièrement lors des séances prévues pour cette feuille. Vous êtes encouragés à terminer les feuilles chez vous et à poser éventuellement des questions au cours ou au TP suivant.

Exercice 1 Premier pas

Le but de cet exercice est de vous familiariser avec les différents modes d'utilisation de l'environnement d'*Objective Caml*. Vous allez écrire et exécuter le programme suivant dans plusieurs contextes différents :

```
let nom = "Curie"
let prenom = "Pierre" ;;
print_string ("je suis " ^ prenom ^ " " ^ nom ^ "\n") ;;
```

Question 1.1 La boucle d'interaction

La boucle d'interaction peut être lancée depuis un *shell* en utilisant la commande `ocaml`. Vous entrez ensuite le programme ci-dessus; remarquez que les *phrases* sont exécutées une par une après chaque retour chariot (on rappelle qu'une *phrase* en *Objective Caml* est terminée par le double point-virgule).

Question 1.2 La boucle d'interaction sous Emacs

En utilisant `Emacs` et le mode adapté à *Objective Caml* vous pouvez exécuter un *toplevel* directement dans un *buffer* Emacs. Pour activer le mode *Objective Caml* sous Emacs, il vous suffit d'ouvrir un fichier dont l'extension est `.ml` (qui est l'extension obligatoire des fichiers *Objective Caml*).

Question 1.3 Compilation en *"byte-code"*

Un programme *Objective Caml* peut être écrit dans un fichier portant l'extension `.ml` (par exemple `exo1.ml`). Puis être compilé (c'est à dire transformé dans un autre langage) par la commande `ocamlc` suivie du nom du fichier.

Le programme compilé (par défaut, il s'agit du fichier `a.out`) peut alors être exécuté par la commande `ocamlrun` ou directement par le nom du fichier.

Écrivez le programme dans un fichier `exo1.ml`, puis compiler le par :
`ocamlc exo1.ml`
ou en spécifiant le nom du fichier généré (ici `exo1`) par : `ocamlc exo1.ml` -o exo1
Exécutez ce programme par `ocamlrun a.out` (respectivement `ocamlrun exo1`) ou directement par `a.out` (respectivement `exo1`).

Question 1.4 Compilation en *code natif*

Il existe un second compiulateur *Objective Caml* qui génère des fichiers directements exécutables par la machine que vous utilisez. Il est appelé par la commande `ocamlopt`. Testez-le avec le fichier `exo1.ml` et vérifier que l'exécution du programme produit bien le même résultat.

Note : Le compilateur en code natif est utilisé lorsque l'on souhaite obtenir un programme s'exécutant plus rapidement (entre 1.5 et 10 fois plus vite selon les cas). Sur l'exemple précédent, vous ne pourez pas constater de différence).

Question 1.5 L'interface de développement `ocamlbrowser`

Le langage *Objective Caml* dispose d'une plateforme de développement "intégrée" qui se lance en exécutant la commande `ocamlbrowser`. Vous pouvez la tester en ouvrant depuis le menu `File` un éditeur et un `shell` contenant un `toplevel` . Vous pourrez alors écrire le programme dans l'éditeur et l'envoyer dans la boucle d'interaction.

Pour la suite du TP, il vous est conseillé d'écrire vos programmes dans un éditeur pour pouvoir les sauvegarder dans un fichier et ainsi en conserver la trace. Par contre vous devez les exécuter dans une boucle d'interaction (par exemple en utilisant le copier-coller depuis l'éditeur) afin de vérifier qu'ils sont corrects et produisent bien les résultats escomptés.

Exercice 2 Expressions et déclarations

Question 2.1 Les entiers en *Objective Caml* sont codés sur une certaine taille, il y a donc un entier maximal qui n'a pas de successeur ainsi qu'un entier minimal qui n'a pas de prédécesseur. Trouvez par tatonnements successifs ces deux entiers.

Question 2.2 Consultez le manuel de référence d'*Objective Caml* [1] pour y trouver les deux entiers précédents qui sont en fait des valeurs prédéfinies.

Question 2.3 Testez les divisions sur les flottants (nombres à virgule flottante). Avec les cas limites vous devez obtenir 3 valeurs particulières. Déclarer ces valeurs respectivement avec les identificateurs `plus_infini`, `moins_infini` et `pas_un_nombre`. Tester les opérations usuelles sur ces "nombres".

Exercice 3 Fonctions

Dans ces exercices, il est demandé d'écrire des fonctions avec un certain nom; il s'agit en fait d'écrire les expressions définnissant ces fonctions et de les déclarer avec le nom comme identificateur. Par exemple, si on vous demande d'écrire la fonction `succ` qui prend un entier et retourne son successeur; vous écrirez :

```
let succ = function x -> x + 1 ;;
```

ou avec la syntaxe alternative des fonctions :

```
let succ x = x + 1 ;;
```

Question 3.1 Écrivez la fonction `even` et qui prend un entier et rend le booléen `true` si l'entier est pair. Faîtes le en utilisant la forme conditionnelle (`if---then---else`) et sans l'utiliser.

Question 3.2 En utilisant la fonction précédente et un opérateur booléen, écrivez la fonction `odd` qui retourne `true` si son argument est un entier impair.

Question 3.3 Écrire la fonction `odd2` qui est équivalente à `odd` mais en utilisant les fonctions `succ` et `even`.

Question 3.4 Écrire une fonction `affiche_parite` qui prend un entier et affiche à l'écran sa parité.

Exercice 4 Fonctions *polymorphes*

Question 4.1 Écrire la fonction `identite` qui prend un argument et retourne exactement cet argument. Quel est le type de cette fonction ?

Question 4.2 Appliquez la fonction `identite` à des valeurs de différents types. Appliquez la aussi à d'autres fonctions.

Exercice 5 Fonctionnelles

On appelle une *fonctionnelle*, une fonction dont la valeur de retour est une fonction. *Objective Caml* permet d'écrire des *fonctionnelles* de la même manière que des fonctions classiques.

Question 5.1 Écrivez la fonction `plus_x` qui prend un entier n et retourne une fonction qui prend comme argument un entier et retourne cet entier ajouté à n. Appliquer la fonction `plus_x` à un entier et appliqué le résultat à un autre entier.

Question 5.2 Comparez la fonction précédente avec une fonction `plus` qui prendrait deux entiers et rendrait leur somme. Ecrivez une telle fonction.

Question 5.3 Écrivez la fonction `itere` qui prend en argument une fonction f et qui retourne l'itéré deux fois de cette fonction, i.e. la fonction qui à une valeur x retourne la valeur $(f \circ f)(x)$.

Question 5.4 Écrivez la fonction `double` qui prend un entier et rend son double en utilisant les fonctions `plus_x` et `itere`.

[1] **http://caml.inria.fr/ocaml/htmlman/index.html**