Feuille de travaux pratiques
Premiers pas et fonctions

Vous pouvez utiliser OCaml de deux façons :
— en compilant un fichier avec la commande `ocamlc`, par exemple :
`ocamlc -o hello hello.ml`
— dans un mode interactif en lançant un interpréteur dans le terminal, avec la commande `ocaml`

Il est conseillé dans un premier temps de travailler dans le mode interactif (à terme, vous aurez aussi besoin de compiler). Vous trouverez dans la rubrique TP/Documentations et aides-mémoires sur madoc un document `ocaml-tools` qui donne, entre autres, la liste des options du compilateur et les commandes utiles dans le mode interactif.

Il n'y a pas d'éditeur conseillé pour ces TPs, vous pouvez utiliser n'importe quel éditeur de votre choix. Si vous souhaitez utiliser emacs avec le mode tuareg, qui permet d'exécuter du code OCaml dans l'éditeur, vous trouverez ici une aide pour l'installation : `http://people.rennes.inria.fr/Alan.Schmitt/teaching/caml/tp1.html`.

Exercice 1 (Prise en main)

Dans un premier temps, essayez de pratiquer la syntaxe et jouer avec le système de types en écrivant quelques expressions usuelles :
— une expression arithmétique sur les entiers avec au moins 3 opérateurs,
— une expression arithmétique sur les entiers avec au moins 3 opérateurs et deux déclarations locales,
— une expression booléenne avec au moins 3 opérateurs et deux déclarations globales,
— une expression contenant au top level un `if`,
— une expression contenant au top level un `match`,
— une fonction qui prend en argument une fonction,
— une expression provoquant les erreurs de typage suivantes (remarquez que OCaml identifie précisément la sous-expression mal typée) :
  — `Error: This expression has type float but an expression was expected of type int`
  — `Error: This function has type int -> int It is applied to too many arguments; maybe you forgot a ';'.`
  — `Error: This expression has type int This is not a function; it cannot be applied..`

Exercice 2 (Exercices de TD)

1. Reprenez l'exercice "inférence de type" vu en TD en essayant de choisir à chaque fois une expression différente de celles vues en TD :

   (a) `val x : int = 4`

(b) `- : char -> int = <fun>`

(c) `val f : int -> int = <fun>`

(d) `val f : float -> int = <fun>`

(e) `val f : float -> float -> float = <fun>`

(f) `val g : (int -> int) -> int = <fun>`

(g) `val g : int -> int -> int = <fun>`

2. programmez l'exercice sur la fonction d'Ackermann en réfléchissant aux affichages : écrivez dans un premier temps une fonction `print_indent` qui affiche la chaine "Ackermann" puis deux entiers passés en argument, avec un nombre d'indentations fixé qui sera indiqué par un argument supplémentaire `nbindent`. Utilisez ensuite `print_indent` pour écrire la fonction `pretty_Ackermann`.
3. de la même façon, implémentez l'exercice sur la suite de Syracuse en particulier les fonctions calculant l'altitude maximum, le temps de vol et le temps de vol en altitude.

Exercice 3 (Dés)

On veut simuler une suite de lancers de dés aléatoires, réalisés de la façon suivante : on stocke un score qu'on augmente en fonction de chaque lancer. On commence avec un score de 0. On lance un dé et on additionne le résultat obtenu au score. Si on fait un 5 ou un 6, on relance le dé pour augmenter son score, sinon on s'arrête.

1. Écrire une fonction `lancer_de` de type type unit -> int qui renvoie un entier aléatoire entre 1 et 6. On pourra utiliser la bibliothèque Random [1] de la façon suivante : on initialise la graine avec `Random.self_init ()`, on tire un entier aléatoire entre 0 et n avec `Random.int` n.
2. Écrire une fonction récursive `jouer`, de type unit -> int qui permet de jouer au jeu décrit ci-dessus et renvoie le score atteient. On prendra soin d'afficher après cahque lancer de dé le résultat du dé et le score courant.
3. Écrire un programme permettant de jouer au jeu. On initialisera le générateur aléatoire une seule fois au début du programme.
4. Tester ce programme en s'assurant d'avoir couvert tous les cas possibles.

1. Voir `https://caml.inria.fr/pub/docs/manual-ocaml/libref/Random.html`