Commandes OCaml à connaître

Déclarations et instructions

commentaires	`(* commenter le code *)`
définition d'une constante	`let v = ...`
définition récursive	`let rec f x = ...`
définition locale	`let a = ... in ...`
définitions parallèles	`let a = ... and b = ...`
définitions successives	`let a = ... in let b = ...`
variable modifiable	`let v = ref ...`
valeur d'une référence	`!v`
modification d'une référence	`v := ...`
fonction sans argument	`let f () = ...`
fonction à un argument	`let f x = ...`
fonction à plusieurs arguments	`let f x1 x2 x3 = ...`
expression conditionnelle	`if ... then ... else ...`

choix multiple	`match ... with`
	`| motif1 -> ...`
	`| motif2 -> ...`
	`| _ -> ...`

ne rien faire	`()`
boucle croissante	`for i = d to d do ... done`
boucle décroissante	`for i = d downto f do ... done`
boucle conditionnelle	`while ... do ... done`
déclencher une erreur	`failwith "message"`

Expressions booléennes

vrai, faux	`true, false`
ou, et, non	`||`, `&`, `not`

Enregistrement

définition d'un type « enregistrement » (les champs x_i sont immuables et de type t_i)	`type record = {x1: t1; x2: t2; ... }`
pour r de type `record`, valeur du champ `x1`	`r.x1`
on peut rendre un champ mutable	`type record = {mutable x1: t1; x2: t2; ... }`
modification (quand le champ est mutable)	`r.x1 <- a`

Expressions entières

opérations arithmétiques	`+ - * /`
modulo	`mod`
valeur absolue	`abs`
entier précédent, entier suivant	`prod, succès`
min, max	`min a b, max a b`
entier aléatoire entre 0 et $n-1$	`Random.int n`

Expressions réelles

opérations arithmétiques	`+. -. *. /.`
puissance	`**`
min, max	`min a b, max a b`
fonctions usuelles	`abs_float, exp, log, sqrt, sin, cos, tan` `sinh, cosh, tanh, asin, tacos, atan`
réel → entier	`int_of_float`
entier → réel	`float`
réel → chaine	`string_of_float`
chaine → réel	`float_of_string`
réel aléatoire entre 0 et a	`Random.float a`

Chaines de caractères

caractère	`'e'`
chaine de caractères	`"Ceci est une chaine de longueur 34"`
k-ième caractère	`chaine.[k]`
modification	`chaine.[k] <- 'e'`
longueur	`String.length`
sous-chaine	`String.sub chaine deb long`
concaténation	`chaine1^chaine2`

Listes

liste composée de $a, b, c, ...$	`[a;b;c;...]`
liste vide	`[]`
ajouter a en tête de l	`a::l`
tête et queue de l	`List.hd l, List.tl l`
longueur de l	`List.length l` (linéaire en $lg(l)$)
concaténation de $l1$ et $l2$	`l1@l2` (linéaire en $lg(l1)$
image miroir de l	`List.rev l` (linéaire en $lg(l)$)
a appartient-il à l ?	`List.mem a l`
existe-t-il a dans l vérifiant $f(a) = true$?	`List.exists f l`
tous les a de l vérifient-ils $f(a) = true$?	`List.for_all f l`
filtrer l par le biais d'un fonction f	`List.filter f l`
appliquer f à tous les éléments de l	`List.map f l`
itérer f avec tous les éléments de l	`List.iter f l`

Tableaux

tableau composée de $a, b, c, ...$	`[|a;b;c;...|]`
tableau vide	`[||]`
k-ième élément	`tab.(k)`
modification du k-ième élément	`tab.(k) <- x`
longueur de tab	`Array.length tab`
création d'un tableau de taille n	`Array.make n x`
création de $[\|f(0); f(1); ...; f(n-1)\|]$	`Array.init n f`
création d'une matrice de taille (n, p)	`Array.make_matrix n p x`
créer une copie de tab	`Array.copy tab`
extraction d'un sous-tableau	`Array.sub tab i long`
a appartient-il à tab ?	`Array.mem a tab`
existe-t-il a dans tab vérifiant $f(a) = true$?	`Array.exists f l`
tous les a de l vérifient-ils $f(a) = true$?	`Array.for_all f l`
appliquer f à tous les éléments de tab	`Array.map f l`
itérer f avec tous les éléments de tab	`Array.iter f l`

Le module Stack

Le programme encourage à utiliser le module `Stack` pour créer et manipuler des piles (structure linéaire LIFO)). Les éléments d'une pile doivent être tous du même type.

créer une pile vide p	`let p = Stack.create()`
tester si la pile p est vide	`Stack.is_empty p`
ajouter un élément x au sommet de p	`Stack.push x p`
renvoyer le sommet de p et le supprimer de p	`Stack.pop p`

Le module Queue

De même, on utilisera le module `Queue` pour créer et manipuler des files (structure linéaire FIFO). Les éléments d'une file doivent être tous du même type.

créer une file vide f	`let f = Queue.create()`
tester si la file f est vide	`Queue.is_empty f`
ajouter un élément x à la queue de f	`Queue.push x f`
renvoyer l'élément qui est tête de f et le supprimer de f	`Queue.pop f`

Le module Hashtbl

Le module Hashtbl permet de créer et manipuler des dictionnaires efficaces.

créer une table vide	`Hashtbl.create n`
ajouter l'entrée (c, v) à t	`Hashtbl.add t c v`
supprimer le couple de clé c	`Hashtbl.remove t c`
la clé c est-elle présente ?	`Hashtbl.mem t c`
valeur associée à la clé c	`Hashtbl.find t c`
option sur la valeur associée à c	`Hashtbl.find_opt t c`
itérer une fonction sur tous les couples de t	`Hashtbl.iter f t`

- à la création d'un dictionnaire, il est conseillé de choisir n de l'ordre du nombre d'associations à créer ;
- si la clé c est déjà présente dans t, l'appel `Hashtbl.add t c v` va remplacer l'ancienne valeur associée à c par v ;
- `Hashtbl.find t c` renvoie un message d'erreur si la clé c n'est pas présente dans t ;
- `Hashtbl.find_opt t c` renvoie `Some v` si (c, v) est dans `t` et `None` sinon.