Programmation impérative et tableaux

1 Enregistrements avec champ modifiable

Dans un type enregistrement, on peut déclarer un ou plusieurs champs comme modifiable (ou *mutable*).

```
type etudiant = {
    nom : string;
    prenom : string;
    mutable classe : string;
  };;

let e = {
    nom = "Tournesol";
    prenom = "Tryphon";
    classe = "MPSI"
  };;
```

Si le champ `c` de l'enregistrement `a` est mutable, la modification de sa valeur se fait avec `a.c <-` e où e est une expression.

```
# e.classe <- "MP";;
- : unit = ()
# e;;
- : etudiant = {nom = "Tournesol"; prenom = "Tryphon"; classe = "
MP"}
```

2 Références

Les variables en OCaml rencontrées jusqu'à présent ne sont pas des variables au sens traditionnel des langages de programmation, puisqu'il est impossible de modifier leur valeur. En programmation impérative, il est indispensable de pouvoir utiliser des variables modifiables pour mémoriser une information évoluant au fil du programme.

En OCaml, on utilise alors une *référence* vers une valeur, c'est-à-dire une case mémoire dont on peut lire et écrire le contenu. Son type correspond à un type enregistrement `'a ref` qui contient un unique champ de type `'a` appelé `contents`.

```
# let b = {contents = 2};;
val b : int ref = {contents = 2}
# b.contents;;
- : int = 2
# b.contents <- 3;;
- : unit = ()
# b;;
- : int ref = {contents = 3}
```

En pratique, on définit une référence avec le mot-clé `ref` (qui est en fait une fonction `'a -> 'a ref`).

Le type de la valeur pointée par une référence est fixé à la création. Une référence pointant vers un objet de type `'a` a le type `'a ref`.

```
# let x = ref 0;;
val x : int ref = {contents = 0}
```

Pour accéder à la valeur de la référence `x`, on utilise l'*opérateur de référencement* `!` suivi du nom de la référence.

Pour modifier une référence, on utilise l'*opérateur d'affectation* `:=` précédé du nom de la référence, et suivi d'une expression.

```
# x := 3;;
- : unit = ()
# x := !x + 1;;
- : unit = ()
# !x;;
- : int = 4
```

Remarque : Pour incrémenter ou décrémenter une référence sur un entier, on dispose des fonctions `incr : int ref -> unit` et `decr : int ref -> unit`

```
# let a = ref 1;;
val a : int ref = {contents = 1}
# incr a;;
- : unit = ()
# !a;;
- : int = 2
```

3 Type `unit` et fonctions avec effets de bord

Une fonction est dite *à effet de bord* lorsqu'elle modifie un état en dehors de son environnement local. Par extension, un opérateur est dit *à effet de bord* lorsqu'il modifie l'un de ses opérandes ; par exemple, l'opérateur d'affectation `:=` est à effet de bord.

Il est fréquent qu'une fonction à effet de bord n'ait pas besoin de renvoyer une valeur. Dans ce cas, elle renverra une valeur de type `unit`. La seule valeur de type `unit` est ()

Remarque : Il s'agit de l'équivalent de `None` en Python (dont le type est `NoneType`).

Les fonctions d'affichage sont des fonctions à effet de bord : `print_string`, `print_char`, `print_int`, `print_float`.

Une fonction qui n'a pas besoin d'un argument aura un argument de type `unit`, on la définira et on l'appellera donc avec `()`

```
# let f () = print_string "Coucou";;
val f : unit -> unit = <fun>
# f ();;
Coucou- : unit = ()
```

Par exemple, la fonction `print_newline` est de type `unit -> unit`.

4 Séquences d'instructions

En Ocaml, les séquences d'instructions sont des expressions séparées par des points-virgules. L'évaluation de

$$e_1 \ ; \ e_2 \ ; \ \ldots \ ; \ e_n$$

provoque les effets éventuels de ces n expressions dans l'ordre, et a la valeur de la dernière expression.

Ce dernier point implique qu'il n'y a aucun intérêt à utiliser une séquence si les expressions dont la valeur n'est pas utilisée (c'est-à-dire toutes sauf la dernière) n'ont pas d'effet de bord.

Par ailleurs, ces différentes expressions (sauf éventuellement la dernière) sont la plupart du temps de type `unit` puisque seule la valeur de la dernière expression est prise en compte.

```
# let classe nom numero =
  print_string "Vivent les ";
  print_string nom;
  print_int numero;
  print_newline ();;
       val classe : string -> int -> unit = <fun>
# classe "MPSI" 0;;
Vivent les MPSI0
- : unit = ()
```

Lorsqu'on souhaite utiliser une séquence d'instructions à la place d'une expression dans une instruction conditionnelle, il est nécessaire d'encadrer celle-ci par les mots-clés `begin` et `end` ou d'utiliser des parenthèses.

```
let est_pair n =
  if n mod 2 = 0
  then
    begin
      print_endline "nombre pair";
      true;
    end
  else
    begin
      print_endline "nombre impair";
      false;
    end
;;
```

Remarque : La fonction `print_endline : string -> unit` permet d'afficher une chaîne de caractères et de revenir ensuite à la ligne.

5 Boucles

5.1 Boucles inconditionnelles

Il existe deux types de boucles inconditionnelles en OCaml :

`for` *indice* `=` e_1 `to` e_2 `do` e_3 `done`

`for` *indice* `=` e_1 `downto` e_2 `do` e_3 `done`

Dans le premier cas, l'indice de boucle (de type `int`) est incrémenté de 1 en 1 entre les valeurs des expressions e_1 et e_2, en effectuant le calcul de e_3 à chaque fois (il est donc préférable que e_3 ait un effet de bord, et pertinent que e_3 soit de type `unit`).
Le second cas est identique, mais l'indice de boucle est décrémenté.

```
# for i = 1 to 10 do print_int i;  print_char ' ' done;;
1 2 3 4 5 6 7 8 9 10 - : unit = ()
# for i = 10 downto 1 do print_int i;  print_char ' ' done;;
10 9 8 7 6 5 4 3 2 1 - : unit = ()
```

5.2 Boucles conditionnelles

La syntaxe d'une boucle conditionnelle est la suivante :

`while` e_1 `do` e_2 `done`

Tant que l'expression e_1 (nécessairement de type `bool`) a la valeur `true`, on évalue l'expression e_2.

```
# let a = ref 2019 in
    while !a > 0 do
      print_int (!a mod 2); a := !a / 2
    done;;
        11000111111- : unit = ()
```

On lira bien évidemment l'affichage de droite à gauche.

Remarque : Si l'expression e_2 n'a pas d'effet de bord, il est peu probable que la condition puisse cesser d'être vérifiée...

6 Structure de tableaux

Les *tableaux* (ou vecteurs) sont des structures correspondant aux vecteurs mathématiques. Un tableau est une suite finie de valeurs **de même type**, modifiables, l'accès à une composante se faisant à temps constant. La *longueur* d'un tableau (son nombre d'éléments) est fixée lors de la création et **ne peut être modifiée**.

En Caml, les éléments d'un tableau de longueur n sont numérotés de 0 à $n-1$.

Un tableau est délimité par `[|` et `|]`, ses éléments sont séparés par des points-virgules.

Un tableau contenant des valeurs de type `'a` a pour type `'a array`.

```
# let t1 = [|1;2;3;4;5;6|];;
val t1 : int array = [|1; 2; 3; 4; 5; 6|]
```

La fonction `Array.length` renvoie la longueur du tableau passé en argument.

```
# Array.length t1;;
- : int = 6
```

Pour créer un tableau, on peut utiliser la fonction `Array.make` : `Array.make n x` renvoie un tableau de taille `n` dans lequel tous les éléments valent `x`.

```
# Array.make 3 'a';;
- : char array = [|'a'; 'a'; 'a'|]
```

⚠ **ATTENTION !** *Les éléments du tableaux sont alors physiquement tous égaux.*

```
# let tabtab = Array.make 3 [|1|];;
val tabtab : int array array = [|[|1|]; [|1|]; [|1|]|]
```

```
# tabtab.(0).(0) <- 1515;;
- : unit = ()
# tabtab;;
- : int array array = [|[|1515|]; [|1515|]; [|1515|]|]
```

L'accès à l'élément d'indice `i` du tableau `t` s'écrit : `t.(i)`.

Cet élément peut être modifié par `t.(i) <-` e où t désigne un tableau et e est une expression.

```
# t1.(5);;
- : int = 6
# t1.(2) <- 42;;
- : unit = ()
# t1;;
- : int array = [|1; 2; 42; 4; 5; 6|]
# t1.(6);;
Exception: Invalid_argument "index out of bounds".
# t1.(-1);;
Exception: Invalid_argument "index out of bounds".
```

7 Exercices divers

Exercice 1 *Prévoir la réponse de l'interpréteur de commandes :*

```
#   let x = ref 0;;
val x : int ref = {contents = 0}
# let z = x;;
val z : int ref = {contents = 0}
# x := 4;;
- : unit = ()
# !z;;
```

Exercice 2 *Prévoir la réponse de l'interpréteur de commandes :*

```
# let a = ref 1;;
val a : int ref = {contents = 1}
# let f x = x + !a;;
val f : int -> int = <fun>
# f 3;;
- : int = 4
# a := 4;;
- : unit = ()
# f 3;;
```

Exercice 3

1. *Écrire une fonction* `copy : 'a array -> 'a array` *qui prend en argument un tableau et en renvoie une copie. Quelle est sa complexité ?*
2. *Écrire une fonction* `sub : 'a array -> int -> int -> 'a array` *qui prend en argument un tableau* `t`*, un entier* `i` *et un entier* `k` *et qui renvoie un tableau de* `k` *éléments correspondant à la portion de* `t` *qui démarre à l'indice* `i`*. Quelle est sa complexité ?*
3. *Écrire une fonction* `mem : 'a -> 'a array -> bool` *testant l'appartenance d'un élément à un tableau. Quelle est sa complexité ?*
4. *Écrire une fonction* `map : ('a -> 'b) -> 'a array -> 'b array` *qui prend en argument une fonction* `f` *de type* `'a -> 'b` *et un tableau* `[|a1; ...; an|]` *d'éléments de type* `'a` *et qui renvoie le tableau* `[|f a1; ...; f an|]`*.*
5. *Écrire une fonction* `append : 'a array -> 'a array -> 'a array` *calculant la concaténation de deux tableaux. Quelle est sa complexité ?*

Ces fonctions sont déjà implémentées dans le module `Array`*.*

Exercice 4 *Écrire une fonction* `tri_selection` *réalisant le tri par sélection d'un tableau, sans utiliser un autre tableau (on parle de tri* en place*). Quelle est sa complexité ?*

Exercice 5 *Écrire une fonction* `tri_insertion` *réalisant le tri par insertion d'un tableau, sans utiliser un autre tableau. Quelle est sa complexité ?*

Exercice 6 *On représente un polynôme par le vecteur de ses coefficients : si* $P = a_0 + a_1X + \dots a_{n1}X^{n-1}$ *est un polynôme de degré strictement inférieur à* $n \in \mathbb{N}^*$*,* P *peut être représenté par le tableau* $[|a_0; a_1; \dots; a_{n-1}|]$*, de longueur* n*.*
Écrire des fonctions d'addition et de multiplication de polynômes.

Exercice 7 *Écrire une fonction* `pascal` *qui génère un tableau de tableaux représentant le triangle de Pascal. Le tableau d'indice* i *sera le tableau* $\left[\left|\binom{i}{0}; \binom{i}{1}; \dots; \binom{i}{i}\right|\right]$*.*
La fonction prendra en argument l'indice maximal du tableau en sortie.

Exercice 8 *Soit* $t = [|t_0; \dots; t_{n-1}|]$ *un tableau d'entiers. On appelle plateau une sous-suite* $t_i, \dots, t_j$ *de composantes consécutives de* t *égales entre elles. Écrire une fonction parcourant une seule fois le tableau, permettant de trouver l'indice du début du plus long plateau ainsi que sa longueur.*