TRAVAUX PRATIQUES XX

Impératif en OCaml

Le but de ce TP est de découvrir l'impératif en OCaml.

A L'opérateur séquence `;`

A.1 Définition

En OCaml, il arrive que l'on veuille enchainer l'évaluation d'expressions qui ont des effets de bord. Par exemple :

```
let affiche_entier n =
  let _ = print_int n in
  print_char '\n'
```

Il existe un opérateur infixe qui simplifie l'écriture de tels code : `;` :

Définition 1 (Séquence).
L'opérateur infixe `;`, aussi appelé **séquence**, est un raccourci pour un `let _ = ... in ...`. Plus précisément, les deux expressions ci-dessous sont équivalentes :

```
(* ceci est une expression *)
  let _ = e0 in e1
```

```
(* ceci est une expression *)
  e0; e1
```

Ainsi, `e0; e1` évalue d'abord `e0` *puis* `e1`, et a comme type et comme valeur le type et la valeur de `e1`.

Exemple. La fonction `affiche_entier` précédente se réécrit :

```
let affiche_entier n =
  print_int n;
  print_char '\n'
```

Remarque. Notez que dans l'exemple ci-dessus, il n'y a pas de `;` « final ». En effet, le `;` prend deux arguments : une première expression (à sa gauche) et une seconde (à sa droite). Et à la fin de la fonction, il n'y a rien à faire ensuite...

0. Écrire une fonction qui prend en argument deux entiers, et les affiche séparés par un espace. Vous n'utiliserez pas le module `Printf` (et utilisez des `;` pour enchainer les évaluations).

Remarque. Vous devez être capable de réécrire `affiche_deux_entiers` sans `;` !!

A.2 `;` et if-then-else

La construction `if then else` est plus prioritaire que le `;`. Cela signifie que `if b then e0; e1 else e2` est parenthésé ainsi : `(if b then e0); e1 else e2`. Et... ceci ne compile pas, car le `else` n'est pas relié à un `if` !
Il faut donc parenthéser la séquence `e0; e1` ainsi :

```
(* ceci est une expression* )
  if b then (e0; e1) else e2
```

De même si on a une séquence dans le `else`.

Il est confortable de réserver les parenthèses `(` `)` aux « maths ». À cette fin, il existe un autre jeu de parenthèse, qui ont le même effet que `(` `)` mais qui à préférer quand on parenthèse « du code » :

```
(* ceci est une expression* )
  if b then begin e0; e1 end else e2
```

Ou, mieux présenté :

```
(* ceci est une expression* )
  if b then begin
    e0;
    e1
  end else
    e2
```

Remarque. Ces nouvelles parenthèses sont vraiment équivalentes aux parenthèses classiques. Ainsi, `2 * begin 3+3 end` fonctionne et vaut la bonne valeur.

A.3 Insistons

Le point-virgule en OCaml n'a rien à voir avec celui de C. En particulier, il ne sert pas à terminer une ligne.

La confusion entre les deux est une des erreurs qui m'exaspèrent le plus. Généralement, elle démontre que vous n'avez pas compris le sens de `;` en OCaml et que vous calquez juste sans réflechir la syntaxe de C. En particulier, voici les deux erreurs les plus communes (et avec lesquelles je suis impitoyable) :

- Au lieu de faire une déclaration locale, faire une déclaration globale terminée par un `;` . Cela n'a juste aucun sens puisqu'il faut une expression à gauche du `;` . Par exemple :
  `let est_pair n = let p = n mod 2; p = 0`
- Mettre un `;` pour terminer une ligne alors qu'il n'y a rien après. Par exemple :
  `let affiche_entier n = print_int n; print_char '\n';`

Si vous ne comprenez pas pourquoi ces deux points sont des erreurs, remplacez dedans le `;` par sa définition. Si vous ne comprenez toujours pas, il est sans doute temps d'aller réviser le tout premier TP OCaml et de revoir déclaration globale et expression (dont déclaration locale).

B Références et boucles for

B.1 Les références

Une référence est une façon d'obtenir une « variable mutable » en OCaml. Elle fonctionne comme un pointeur (immuable) vers un contenu mutable. Pour simplifier, au lieu de parler de la « valeur pointée par la référence » je parlerais de son « contenu ».

Définition 2 (Références).
Le type des références dont le contenu est de type t est `t ref`
La fonction `ref : 'a -> 'a ref` permet de créer une référence : `ref x` construit et renvoie une référence dont le contenu est initialisé à `x` .
L'opérateur `!` appliqué à une référence permet d'en obtenir le contenu : `!r` est le contenu de la référence `r`
L'opérateur infixe `:=` permet de modifier le contenu d'une référence : `r := y` est une expression de type `unit` qui a pour effet secondaire de modifier le contenu de `r` pour y mettre `y` .

B.2 Boucles for

En OCaml, une boucle (`for` ou `while`) est une expression de type `unit`. En particulier, la seule chose utile dans une boucle est ses effets secondaires.

Définition 3 (Boucles for).
Une boucle `for` s'écrit :

```
(* Ceci est une expression de type unit ! *)
  for i = debut to fin_incluse do
    expression_corps
  done
```

L'évaluation de cette boucle consiste à évaluer l'expression `expression_corps` avec d'abord `i` valant `debut` , puis debut+1, etc, jusqu'à `fin_incluse` .
L'expression du corps doti être de type `unit`, et la boucle est elle-même de type `unit`

Exemple. La fonction ci-dessous affiche les entiers de 1 à n (inclus) :

```
let affiche_plein_entiers n =
  for i = 1 to n do
    print_int i
  done
```

1. Modifiez la fonction ci-dessus pour qu'elle affiche un retour à la ligne entre les affichages d'entiers.
2. Écrire une fonction qui affiche les côtés d'un carré d'étoile $n \times n$. Par exemple, voici l'affichage attendu pour `n = 6` :

```
******
*    *
*    *
*    *
*    *
******
```

Remarque. La boucle `for` incrémente le compteur de boucle de 1 en 1. Si l'on veut décroitre de 1 en 1, il faut utiliser `downto` au lieu de `to`. Si l'on veut varier d'autre chose que 1, il ne faut pas utiliser une boucle `for`

C Les tableaux

C.1 Les fondamentaux

Le type des tableaux dont le contenu est `'a` en OCaml est le type `'a array`

Définition 4 (Accéder à et modifier un tableau).
Un tableau de longueur `len` est indicé de `0` à `len-1`
Si `arr` est un tableau et `i` un indice, alors `arr.(i)` est le contenu de la case d'indice `i`
L'opérateur infixe `<-` permet de modifier le contenu d'une case : `arr.(i) <- y` est une expression de type `unit` qui a pour effet secondaire de modifier la case `arr.(i)` pour y mettre `y`

Remarque. Confondre `:=` et `<-` est une étourderie commune.

Définition 5 (Créer un tableau).
La fonction `Array.make int -> 'a -> 'a array` permet de créer un tableau : `Array.make len x` créer un tableau de longueur `len` dont les cases sont initialisées à `x`
On peut aussi écrire un tableau en donnait toutes ses valeurs les unes après les autres (comme avec les listes !) [|x1 ; ... ; xk|] pour créer un tableau qui contient `x0` puis ... puiz `xk`.

3. Écrire et tester une fonction qui calcule la somme d'un tableau. On utilisera une référence pour stocker et mettre à jour la somme au fur et à mesure.

C.2 Fonctions utiles du module `Array`

- `Array.length : 'a array -> int` : renvoie la longueur d'un tabeau (en temps constant !)
- `Array.init : int -> (int -> 'a) -> 'a array` : permet de créer et d'initialiser un tableau. Ainsi, `Array.init len f` est le tableau `[|f 0; f 1; ...; f(len-1)|]`
- `Array.copy : 'a array -> 'a array` : permet de créer une copie (superficielle [1]) d'un tableau.
- `Array.make_matrix : int -> int -> 'a -> 'a array array` : comme `Array.make`, mais sert à créer un tableau de tableaux (une matrice). Le premier argument est le nombre de lignes, le second le nombre de colonne, le dernier la valeur initiale à mettre dans les cases.
- `Array.mem : 'a -> 'a array -> bool` : comme pour les listes.
- `Array.iter : ('a -> unit) -> 'a array -> unit` et `Array.map : ('a -> 'b) -> 'a array -> 'b` : comme pour les listes.
- `Array.for_all : ('a -> bool) -> 'a array -> bool` et `Array.exists : ('a -> bool) -> 'a array -> bool` : comme pour les listes.
- `Array.sort` : permet de trier. Je vous recommande d'aller lire sa documentation à chaque fois que vous en avez besoin.

4. En utilisant la fonction ci-dessous pour lire un entier, écrire une fonction `lit_n_entiers : int -> int array` qui lit `n` entiers depuis la ligne de commande et renvoie le tableau constitué de ces entiers.

```
let lit_entier () = Scanf.scanf " %d" Fun.id
  (* l'espace dans le scanf est importante ! *)
```

5. Mise en pratique : Résolvez le problème du camping du déblocage du niveau 4 de france-ioi.
Indication : faites une fonction récursive qui à (i, j) associe le côté du plus grand carré dont le coin en haut à gauche est (i, j)

1. C'est à dire que si les cas du tableau contiennent des pointeurs (par exemple d'autres tableaux ou des références), on recopie uniquement le pointeur au lieu d ecréer un double de ce qui est pointé.