TP n° 3 :

1 Programmation

Question 1 Reprendre le dernier exercice de la feuille de TD nn° 2, et les deux exercices de la feuille de TD nn° 3.

Vous pouvez tracer l'exécution d'une fonction avec la directive `#trace`. La directive annulant ces traces est `#untrace`.

```
# #trace f ;;
f is now traced.
# f 3 ;;
f <-- 3
f --> 4
- : int = 4
# #untrace f ;;
f is no longer traced.
# f 3 ;;
- : int = 4
```

Question 2 Nommez les fonctions anonymes utilisées pour réaliser les diverses fonctions que vous avez réalisées dans la feulle de TD nn° 3 et tracez les pour suivre les différentes étapes des calculs que les fonctions de ces exercices engendrent.

2 Espionner l'évaluation des expressions en CAML

2.1 Le problème

L'évaluation des expressions en CAML suit une stratégie par valeurs. La description de cette stratégie donnée en cours précise en particulier que

1. lors de l'évaluation d'une application (c'est à dire une expression de la forme `e1 e2`), les deux sous-expressions, l'expression fonctionnelle (`e1`) et l'expression argument () doivent être évaluées avant la réduction de cette application ;
2. la valeur d'un n-uplet est le n-uplet des valeurs de chacune des composantes.

Mais ce qui n'a pas été précisé c'est dans quel ordre les deux sous-expressions d'une application sont évaluées, ainsi que les n composantes d'un n-uplet.

Par exemple, dans l'expression

```
(List.map (function x -> 2*x)) [1+2 ; 2+3 ; 3+4]
```

est-ce l'expression fonctionnelle `List.map (function x -> 2*x)` qui est d'abord réduite ou bien est-ce son argument `[1+2 ; 2+3 ; 3+4]` ? Et dans l'évaluation du triplet

```
(12+3, char_of_int 65, sqrt 2.)
```

dans quel ordre sont évalués les trois composantes ?

L'un des buts de cette partie est de préciser ces dernier points.

2.2 Un outil d'espionnage

Pour ce faire vous allez utiliser les effets de bords provoqués par une instruction d'impression sur la sortie standard `Printf.printf`.

La fonction `printf` du module du même nom imprime sur la sortie standard les valeurs de ses arguments. Le premier de ses arguments est un format d'impression analogue aux formats d'impression des fonctions `printf` de la bibliothèque `stdio`. Le nombre et les types des arguments qui suivent dépendent des formats présents dans le premier argument.

Voici un exemple d'utilisation de cette fonction

```
# Printf.printf "%s\n" "elfe" ;;
elfe
- : unit = ()
```

On le voit la chaîne de caractères `"elfe"` a été imprimée (format `%s`), suivi d'un passage à la ligne (format `\n`). La valeur de cette expression est `()` qui est l'unique valeur de type `unit` en CAML.

On peut faire précéder l'évaluation d'une expression par une impression en utilisant le séquencement des expressions exprimé en CAML par le caractère point-virgule (`;`).

```
# Printf.printf "%s\n" "elfe" ;
  1 + 2 ;;
elfe
- : int = 3
```

Étudiants observateurs, vous aurez immédiatement remarqué l'apparition d'une notion de programmation impérative, la séquence, dans un TP de programmation fonctionnelle ! Pouah ! Quelle horreur ! Le temps qui s'écoule s'immisce dans notre monde fonctionnel duquel il était jusqu'à présent absent.

Mais c'est le problème même qui nous préoccupe qui le vaut puisque nous voulons savoir dans quel **ordre** les sous-expressions d'une expression donnée sont évaluées, et c'est cet ordre qui introduit le temps dans nos affaires.

Mais rassurez-vous, il est tout à fait possible de parvenir à nos fins dans un style purement fonctionnel, sans utiliser de séquence, si on se souvient que CAML suit une stratégie d'évaluation par valeurs. En particulier, le passage d'un paramètre se fait par valeurs. Autrement dit, si on veut insister, dans l'application d'une fonction à un argument, cet argument est évalué **avant** (encore le temps !) d'être transmis à la fonction.

```
# (function () -> 1 + 2) (Printf.printf "%s\n" "elfe") ;;
elfe
- : int = 3
```

Si on veut espionner l'évaluation d'une expression `e` par l'impression d'un message `m`, il suffit de former l'application

```
(function () -> e) (Printf.printf "%s\n" m)
```

2.3 Principe de l'espionnage

Pour déterminer si dans l'évalution d'une expression telle ou telle sous-expression est évaluée avant telle autre, il suffit de faire précéder l'évaluation de chacune des sous expressions par une impression.

En voici un exemple qui révèle dans quel ordre sont évalués les opérandes d'une addition

```
# (function () -> 2 * 3) (Printf.printf "%s\n" "6") +
  (function () -> 4 * 5) (Printf.printf "%s\n" "20") ;;
20
6
- : int = 26
```

Il ressort des impressions que l'opérande droit d'une addition est évalué avant l'opérande gauche.

2.4 À vous de jouer !

Question 3 Déterminer l'ordre dans lequel sont évalués les opérandes dans une somme de trois nombres `a + b + c`.

Question 4 Espionnez les trois expressions `e1`, `e2` et `e3` dans l'expression conditionnelle **`if`** `e1` **`then`** `e2` **`else`** `e3`.

Question 5 Dans quel ordre sont évalués les composantes d'un n-uplet ?

Question 6 Et les composantes d'une liste ?

Question 7 Dans une application `e1 e2` quel est l'ordre d'évaluation ?

Question 8 Par un espionnage adéquat, mettez en évidence que les sous-expressions du corps d'une fonction ne sont pas évaluées avant que cette fonction ne soit appliquée. Prenez par exemple la fonction **`function`** `x -> (2 + 3) * x`.