Programmation Fonctionnelle
Examens & Partiels passés

10 octobre 2023

Question 1 *On considérera le code suivant.*

```
1  let x z y =
2  let x =
3  let z =
4  y - z in
5  let x z y =
6  z - y in
7  x y z in
8  x + y
```

1. *Pour chaque occurrence des noms x, z et y, indiquer où se trouve son lieur — c'est-à-dire la déclaration qui correspond à cette occurrence — en commentant à la ligne où apparaît la variable, le numéro de la ligne de son lieur.*
2. *Dire ce que la fonction calcule.*

Question 2 *Trouver deux parenthésages du code suivant tel que dans un des cas la fonction calcule la somme des deux arguments, alors que dans l'autre cas le résultat est le premier argument moins le deuxième.*

```
let plus_moins x y =
let g = fun u -> - u in
match x with
| 0 -> g 0 + y
| x -> g 0 + g y - x
```

Question 3 *On considérera le code suivant.*

```
exception E

let f y l =
  let rec h = fun l ->
    match l with
    | [] -> raise E
    | x :: [] -> (x, [y])
    | x :: l' -> let g = h l' in
                 let s = min (fst g) x in
                 (fst g, s :: (snd g))
  in
  snd (h l)
```

1. *Remplir le tableau suivant, qui doit détailler le calcul de* `f 9 [5;1;7;3;4]`*. Chaque ligne doit représenter les valeurs des déclarations locales lors de l'appel à la fonction* `h` *sur l'argument* `l` *et quand* `y=9`*. Une fois le tableau rempli, donner le résultat du calcul de* `f 9 [5;1;7;3;4]`*.*

`l =`	`g =`	`s =`	`h l =`

`[4]`
`[3;4]`
`[7;3;4]`
`[1;7;3;4]`
`[5;1;7;3;4]`
	pas de valeur	pas de valeur	

2. *Expliquer ce que la fonction* `f` *calcule. Indication : la fonction fait deux choses au même temps.*

Question 4 *Soit le code suivant :*

```
let f x y z =
  let f y = z + y in
  let z x y = x + y - z in
  z (f y) (f x)
```

Donner une expression plus simple pour la fonction globale `f`*.*

Question 5 *Soit le code suivant :*

```
type student = { mutable age : int; name : string}

let s1 = { age = 3; name = "A"}
let s2 = { age = 3; name = "B"}
let s3 = { s1 with name = "C"}
let s4 = s1
```

Donner les valeurs de `s1`*,* `s2`*,* `s3` *et* `s4` *après l'affectation de* `6` *à l'age de* `s1`*.*

Question 6 *Même question que la précédente, mais en considerant le code suivant :*

```
type student = { age : int ref; name : string}

let s1 = { age = ref 3; name = "A"}
let s2 = { age = ref 3; name = "B"}
let s3 = { s1 with name = "C"}
let s4 = s1
```

Question 7 *Soit le code suivant :*

```
let t = let r = [| 0; 1; 2 |] in [| r; r; [|0; 1; 2|] |]
```

Donner la valeur de `r` *avant et après l'execution de chacune des lignes suivantes*

```
t.(0).(2) <- 3;;
t.(0) <- t.(2);;
t.(0).(2) <- 3;;
```

Question 8 *Soit le code suivant :*

```
let l1 = List.fold_left (fun l x -> x :: l) [] [1;2;3]
let l2 = List.fold_right (fun x l -> x :: l) [1;2;3] []
```

Donner les valeurs de `l1` *et* `l2`*.*

Question 9 *On considère le code suivant :*

```
let get_and_set1 =
  let mem = ref 0 in
  fun x ->
  let v = !mem in
  mem := x; v
```

```
let get_and_set2 =
  fun x ->
  let mem = ref 0 in
  let v = !mem in
  mem := x; v
```

Donner les valeurs de `a1`, `b1`, `c1`, `a2`, `b2` *et* `c2` *après les affectations suivantes. Quel implémentation de la fonction correspond mieux à son nom ?*

```
let a1 = get_and_set1 3
let b1 = get_and_set1 7
let c1 = get_and_set1 2

let a2 = get_and_set2 3
let b2 = get_and_set2 7
let c2 = get_and_set2 2
```

Question 10 *Soit le code suivant :*

```
let g = List.map (fun x -> x := !x + 1; if !x mod 2 = 0 then ref !x
    else x)

let l = let r = ref 0 in [r;r;r;r;r]
```

Donner le résultat de `g l`.

Question 11 *Soit la fonction suivante :*

```
let mystere l = let rec aux l x =
                  match l with
                  | [] -> []::x
                  | _ :: s -> aux s (l::x)
  in aux l []
```

Donner le résultat de l'évaluation de `mystere [true; false; true]`. *Expliquer* **brièvement** *ce que fait la fonction* `mystere`.

Fermetures

Question 12 *Écrire une fermeture (fonction à état interne)* `loge` *(logarithme entier) telle que* `loge n` *renvoie le plus grand entier* `k` *tel que* `2^k` *soit inférieur ou égal à* `n`. *Votre implémentation doit être récursive terminale. Elle procèdera en multipliant l'état interne par* 2 *à chaque appel récursif.*

Question 13 *Écrire une fonction* `mem_list` *qui donne la possibilité à l'utilisateur d'entrer des éléments un par un dans une liste, et d'en renvoyer le contenu. Ainsi, la séquence d'instructions ci-dessous doit se comporter comme décrit en commentaire :*

```
mem_list 2  (* renvoie [2] *)
mem_list 3  (* renvoie [4 ; 2 ] *)
mem_list 5  (* renvoie [5 ; 3 ; 2] *)
```

Question 14 *On considère la fonction* `remember` *suivante :*

```
type 'a option = None | Some of 'a

let remember =
  let cache = ref None in
  (fun x ->
```

```
  match !cache with
  | Some y -> y
  | None -> cache := Some x; x)
```

(a) Quel est le type de `remember` *?*

(b) Donner la valeur de sortie de chaque invocation de `remember` *dans les instructions suivante :*

```
remember 4
remember 2
```

Question 15 *Écrire une fermeture (fonction à état interne)* `fibo` *telle que* `fibo n` *est le terme de rang* `n` *de la suite de Fibonacci. On rappelle que chaque terme de la suite de Fibonacci est obtenu en additionnant les deux précédents et que les deux premiers termes valent 1. Ainsi,* `fibo` *renverra respectivement 1, 1, 2, 3, 5, 8 pour* `n` *allant de 0 à 5.*
Indication : on veillera à ce que la fonction définie renvoie bien ces valeurs pour 0, 1, 2...