Devoir Surveillé 1

7 octobre 2023

INFORMATIQUE MP2I

DURÉE DE L'ÉPREUVE : **2 heures**

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Ce sujet comporte quatre pages numérotées de 1/8 à 8/8

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Tournez la page S.V.P.

Vue d'ensemble du sujet

Ce sujet est composé de 5 parties indépendantes, utilisant le langage de programmation OCaml.

Les différentes parties sont indépendantes et peuvent être traités dans n'importe quel ordre. Il n'est pas nécessaire d'avoir répondu aux questions d'une partie pour répondre aux questions d'une autre partie.

- La partie I s'intéresse au typage des variables en OCaml.
- La partie II s'intéresse à la ligne de commande et à l'arborescence des fichiers.
- La partie III s'intéresse à la somme des chiffres de la représentation d'un entier en base 10.
- La partie IV s'intéresse à la suite de Fibonacci et son extension aux indices négatifs.
- La partie V consiste en un problème plus ouvert.

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat.

Quand l'énoncé demande de coder une fonction, sauf indication explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites.

Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle.

Partie I. Inférence de type – 10 pts

☐ 1 – Donner les types des fonctions suivantes, en justifiant :

```
let f a b =
  let (c, d) = b in
  (c d) + (d a) + a;;

let g c d =
  ((c d) + 1, (d 2) * 4);;
```

Réponse 1.

```
# let f a b =
  let (c, d) = b in
  (c d) + (d a) + a;;
val f : int -> ((int -> int) -> int) * (int -> int) -> int = <fun>
# let g c d =
  ((c d) + 1, (d 2) * 4);;
val g : ((int -> int) -> int) -> (int -> int) -> int * int = <fun>
```

Détails :

```
a : int;
d : int -> int; (d a : int)
c : (int -> int) -> int; (c d : int)
b : ((int -> int) -> int) * (int -> int); ((c, d) = b)
f a b : int

d : int -> int; (d 2 : int)
c : (int -> int) -> int; (c d : int)
g c d : int * int
```

☐ 2 – Donner un exemple de fonction de type

`(int -> int) -> (int -> int) -> int`.

Réponse 2.

```
# let fa fb fc =
  fb 0 + fc 0;;
val fa : (int -> int) -> (int -> int) -> int = <fun>
```

Partie II. Gestion des fichiers en ligne de commande – 10 pts

Le fichier `source.ml` du répertoire courant contient le code suivant :

source.ml

```
let f x y = (x + 1) * (y - 1)
let u = f (f 2 3) (f 4 5)
let () = print_int u
```

☐ 3 – Donner une commande permettant de compiler `source.ml` afin de créer un exécutable nommé `exec`.

Réponse 3. `ocamlc source.ml -o exec`

☐ 4 – Que fait alors la commande `./exec` ?

Réponse 4. Elle affiche 133.

☐ 5 – On suppose que l'on se trouve dans un répertoire `TEST`, que ce dernier est vide et que l'on exécute les commandes suivantes.

```
florian@florian-VirtualBox:~/TEST$ ls
florian@florian-VirtualBox:~/TEST$ mkdir a
florian@florian-VirtualBox:~/TEST$ mkdir b
florian@florian-VirtualBox:~/TEST$ mkdir c
florian@florian-VirtualBox:~/TEST$ mkdir d
florian@florian-VirtualBox:~/TEST$ echo "hello" > a/t.txt
florian@florian-VirtualBox:~/TEST$ cp a/t.txt d/foo.txt
florian@florian-VirtualBox:~/TEST$ cd c
florian@florian-VirtualBox:~/TEST/c$ mkdir ../b/f
florian@florian-VirtualBox:~/TEST/c$ mkdir e
florian@florian-VirtualBox:~/TEST/c$ echo "bonjour" > ../f.txt
florian@florian-VirtualBox:~/TEST/c$ cd ..
florian@florian-VirtualBox:~/TEST$ rm a/t.txt
```

Dessiner l'arborescence finale des fichiers et répertoires (on utilisera `TEST` comme racine de l'arborescence).

Réponse 5.

```
florian@florian-VirtualBox:~/TEST$ tree
.
+-- a
|
+-- b
|   |
|   +-- f
|
+-- c
|   |
|   +-- e
|
+-- d
|   |
|   +-- foo.txt
|
+-- f.txt

6 directories, 2 files
```

Partie III. Somme des chiffres – 10 pts

Dans cette partie, on considère la représentation décimale des nombres.

☐ 6 – Écrire une fonction `somme_des_chiffres : int -> int = <fun>` qui calcule la somme des chiffres d'un entier.

Réponse 6.

```
let rec somme_des_chiffres n =
  if n < 10 then
    n
  else
    n mod 10 + somme_des_chiffres (n/10);;
```

```
let somme_des_chiffres n =
  let rec somme_aux n acc =
    if n = 0 then
      acc
    else
      somme_aux (n / 10) (acc + n mod 10)
  in somme_aux n 0;;
```

☐ 7 – On définit la fonction suivante :

```
let rec mystere n =
  if n < 10 then
    n
  else mystere (somme_des_chiffres n);;
```

Montrer que pour tout nombre entier strictement positif n, cette fonction renvoie 9 si n est un multiple de 9, et $n \mod 9$ sinon.

Réponse 7. D'abord, la fonction mystere termine car pour tout nombre $n \geq 10$, la somme des chiffres de n est strictement plus petite que n.
Ensuite, si $0 < n < 10$, la fonction est correcte. Montrons que pour tout $n \geq 10$, n et la somme de ses chiffres ont le même reste dans la division euclidienne par 9.

$$\forall i \in \mathbb{N}, 10^i - 1 = \sum_{j=0}^{i-1} 9 \times 10^j$$

$$\forall i \in \mathbb{N}, 10^i = 1 \mod 9$$

$$\forall n \in \mathbb{N}, n = \sum_{i=0}^{k} a_k 10^k, n = \sum_{i=0}^{k} a_k \mod 9$$

□

☐ 8 – Écrire une fonction `prochain : int -> int -> int = <fun>` qui prend en entrée deux nombres n et k et qui renvoie le plus petit nombre $n^* \geq n$ tel que la somme des chiffres de n^* vaut k.

Réponse 8.

```
let rec prochain n k =
  if somme_des_chiffres n = k then n
  else prochain (n+1) k
```

☐ 9 – Montrer que la fonction `prochain` termine. Pour cette question, on suppose que le type `int` correspond aux entiers naturels et n'est pas soumis aux dépassements de capacité.

Réponse 9. Pour tout $(n,k) \in \mathbb{N}^2$, il existe un nombre supérieur ou égal à n dont la somme des chiffres vaut k. Par exemple :

$$\underbrace{11\ldots1}_{k \text{ fois}}\ \underbrace{00\ldots0}_{\log_{10}(n) \text{ fois}}$$

Partie IV. Fibonacci – 10 pts

On a vu dans le cours qu'une implémentation naïve pour calculer les termes de la suite de Fibonacci recalcule plusieurs fois (même beaucoup) la même valeur. Pour éviter ce problème, on peut calculer simultanément deux termes consécutifs :

```
let rec fib2 n =
  if n = 0 then
    (1, 1)
  else
    let (u, v) = fib2 (n-1) in
    (v, u + v);;
```

☐ 10 – Quelle ligne aurait-on pu ajouter pour vérifier explicitement la précondition $n \geq 0$?

Réponse 10. `assert(n>=0);`

☐ 11 – Montrer que cette fonction termine pour tout entier n positif.

Réponse 11. Si n est positif, soit `fib2` termine, soit il effectue un seul appel récursif avec un argument positif et strictement plus petit.
Si on avait une boucle infinie, on aurait alors une suite infinie d'entiers positifs strictements décroissants, ce qui est impossible. ☐

☐ 12 – Montrer que cette fonction renvoie (f_n, f_{n+1}), où f_n est le n-ième terme de la suite de Fibonacci.

Réponse 12. Par récurrence :

$n = 0$: $(f_0, f_1) = (1, 1)$
$n > 0$: $u = f_{n-1}$ et $v = f_n$ par hypothèse de récurrence, alors $u + v = f_{n+1}$

☐

On souhaite étendre la suite de Fibonacci aux indices négatifs.

☐ 13 – Pour que la relation de récurrence $f_{n+2} = f_{n+1} + f_n$ soit vérifiée, quelles doivent être les valeurs de f_{-1}, f_{-2}, f_{-3} et f_{-4} ?

Réponse 13. $f_{-1} = 0$
$f_{-2} = 1$
$f_{-3} = -1$
$f_{-4} = 2$

☐ 14 – Proposer une nouvelle version du code de `fib2` qui donne un résultat correct pour les nombres négatifs également.

Réponse 14.

```
let rec fib2 n =
  if n = 0 then (1, 1)
  else if n > 0 then
    let (u, v) = fib2 (n-1) in (v, u+v)
  else
    let (u, v) = fib2 (n+1) in (v-u, u)
```

☐ 15 – Montrer que pour tout $n \in_m athbb{N}^*$, $f_{-n-1} = (-1)^{n-1} f_{n-1}$.

Réponse 15. On a $f_n = f_{n+2} - f_{n+1}$.
Par récurrence forte :
$f_{-1-1} = f_{-2} = 1 = (-1)^0 f_0$
$f_{-2-1} = f_{-3} = -1 = (-1)^1 f_1$

$$\begin{aligned} f_{-(n+2)-1} &= f_{-n-1} - f_{-(n+1)-1} \\ &= (-1)^{-n-1} f_{n-1} - (-1)^{-n-2} f_{(n+1)-1} \\ &= (-1)^{-n-1}(f_{n-1} + f_n) \\ &= (-1)^{-(n+2)-1} f_{(n+2)-1} \end{aligned}$$

Partie V. Flavius Josèphe – 10 pts

n soldats juifs (dont Flavius Josèphe), cernés par des soldats romains, décident de se suicider. Ils se mettent en cercle, et un premier soldat est choisi au hasard pour être exécuté ; puis le k-ième à partir de sa gauche est exécuté. Tant qu'il y a des soldats, la séléction continue de la même façon. Le but est de trouver à quelle place doit se tenir un soldat pour être le dernier.

☐ 16 – Écrire une fonction `josephe_2 : int -> int = <fun>` qui prend en entrée le nombre n de soldats et donne la position cherchée (en comptant vers la gauche depuis le premier soldat exécuté, qui est donc en position $0 (= n)$), dans le cas où $k = 2$.
Toute trace de recherche (schéma, calcul manuel sur de petites valeurs, etc.), même incomplète sera prise en compte dans l'évaluation

Josèphe, peu enthousiaste à l'idée de mourir, serait parvenu à trouver cette place, dans le cas $n = 41$ et $k = 3$, lors du siège de Jotapata par Vespasien, en 67 après J.-C..

Réponse 16.

```
let rec josephe_2 n =
  if n = 1 then
    (* S'il n'y a qu'un seul soldat, il survit *)
    0
  else
    let i = josephe_2 (n-1) in
    (* Sinon, on en tue un, il reste (n-1) soldats *)
    if i = n - 2 then
    (* Le nouveau dernier soldat est le soldat 1 *)
      1
    else
    (* Le nouveau premier soldat est le soldat 2 *)
      i + 2;;
```

Fin de l'épreuve