Examen de Programmation Fonctionnelle

2e session — 16 Juin 2009

Durée : 3h

Documents autorisés : cours, TPs, projets.
Livres interdits — Échange de documents interdit — Ordinateurs et téléphones interdits

Vous devez répondre aux questions en OCaml, en utilisant uniquement un style fonctionnel (pas de références, mais vous pouvez utilisez des exceptions). Une fonction écrite en style impératif ne rapportera aucun point.

Le barème ci-dessous est indicatif et est susceptible d'être modifié.

Vous pouvez utiliser les fonctions de la bibliothèque standard.

1 Exercice (7 points)

1. Donner le type et la valeur de l'expression suivante. Justifier rapidement.

```
print_int 4
```

2. Donner le type et la valeur de l'expression suivante. Justifier.

```
(fun x -> x+1) 3
```

3. Donner le type de l'expression suivante. Justifier.

```
(fun x -> Some x)
```

4. Donner le type et la valeur de l'expression suivante. Justifier.

```
(fun x y -> x y) (fun x -> x 1) (fun x -> x)
```

5. Donner le type et la valeur de l'expression suivante. Justifier.

```
let a = 100 in
a + let a = 10 in a + 1
```

6. Donner le type et la valeur de l'expression suivante. Justifier.

```
let g x = 7 in List.map (fun p -> p g) [(fun f -> f 4)]
```

7. On définit la fonction `f` de la façon suivante.

```
let f s x = try int_of_string s with Failure _ -> x;;
```

Quel est son type ? Justifier.

8. Sachant que l'interface du module `List` contient :

```
val fold_right2 :
      ('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> 'c -> 'c
```

donner le type des deux expressions suivantes. Justifier.

```
(* 1 *)
List.fold_right2 (fun a b c -> a + b + c) [240] [1] 27

(* 2 *)
List.fold_right2
   (fun (a, b) c -> function None -> Some (a + int_of_string b) | _ -> c)
```

L'expression suivante est-elle bien typée ? Justifier.

```
List.fold_right2 (fun a b c -> a) [240] [1] (Some 27);;
```

9. On définit :

   ```
   type 'a t = E of (int * 'a) t | Z of 'a
   ```

   Écrire deux valeurs différentes de type `unit t`.

10. On définit :

    ```
    type ('a, 'b) t1 = {a : 'a; c : 'a -> 'a * 'b}
    ```

    Définir une valeur de type `(('a -> 'a), bool) t1`.

2 Exercice (2 points)

1. Qu'est-ce qu'un type abstrait ? Expliquez le concept en une dizaine de lignes.
2. Donner un exemple d'utilisation (sans écrire de code).
3. Enfin, expliquez sur un exemple quelle syntaxe utiliser pour définir un type abstrait en OCaml (interface et implémentation).

3 Exercice (2 points)

Écrire une version du programme suivant qui n'utilise ni boucles, ni références, ni données mutables. La fonction doit avoir exactement le même type et le même comportement.

```
let somme n =
   let resultat = ref 0 in
   for i = 1 to n do
     print_endline "Entrez un entier :";
     resultat := !resultat + (read_int ());
   done;
   !resultat
```

4 Exercice (4 points)

1. Définir un type pour représenter des arbres binaires, avec des valeurs à chaque noeud et chaque feuille. Le type de ces valeurs doit être quelconque, les valeurs des feuilles ont toutes le même type, les valeurs des noeuds binaires aussi, mais ces deux types peuvent être différents.
2. La fonction `miroir` calcule l'arbre miroir, c'est-à-dire l'arbre obtenu en échangeant les fils gauche et droit de chaque noeud binaire. Quel est le type de cette fonction ? Écrire son implémentation.
3. Écrire le type puis le code d'une fonction `split` qui prend en paramètre un arbre dont les valeurs étiquetant les noeuds et les feuilles sont de type `int`, et qui renvoie un couple d'arbres étiquetés par des valeurs de type `int option` :

- le premier des deux arbres est obtenu en remplaçant par `None` toutes les valeurs strictement négatives de l'arbre initial ;
- le deuxième est obtenu en remplaçant par `None` toutes les valeurs positives ou nulles de l'arbre initial.

4. Quels changements apporter pour transformer la fonction `split` en une fonction générique qui prend en paramètre le critère de test pour séparer les deux arbres ? Quel est le type de la nouvelle fonction ?
5. Écrire une fonction de parcours générique de ce type arbre. Quel est son type ? Réimplémenter `inline` et `split` à l'aide cette fonction.

5 Exercice (5 points)

On souhaite représenter un flux de données, par exemple des données provenant d'un périphérique (clavier ou capteur...). Les données arrivent progressivement et l'on souhaite pouvoir commencer à traiter les données arrivées avant que toutes les données soient reçues.

Pour représenter un flux, on définit le type suivant :

```
type flux = Suite of (string * (unit -> flux)) | Fini;;
```

Le constructeur `Fini` représente un flux vide (toutes les données sont arivées). Le constructeur `Suite` a deux paramètres : une chaîne représentant des données déjà arrivées, et la fonction à appeler pour avoir la suite du flux.

1. Écrire une fonction `affiche_flux : flux -> unit` qui prend un flux et le lit jusqu'au bout en affichant son contenu à l'écran.
2. Écrire une fonction `flux_clavier : unit -> flux` qui crée un flux tapé au clavier, ligne par ligne. Le flux se termine lorsque la chaîne tapée est `"\quitter"`.
3. Écrire une fonction `lit_n : int -> flux -> string` qui renvoie les `n` prochains caractères du flux (dès qu'ils sont arrivés). Vous pourrez utiliser les fonctions suivantes du module `String` :

   ```
   val sub : string -> int -> int -> string
   val length : string -> int
   ```

   (`String.sub s b l` renvoie la sous-chaîne de `s` de taille `l` commençant au caractère d'indice `b`, et lève l'exception `Invalid_argument` en cas d'erreur.)
4. Écrire l'interface du module de gestion des flux en abstrayant le type `flux`. Quelles fonctions faut-il écrire pour créer et utiliser un flux ? Implémenter ces fonctions.
5. Réécrire les fonctions `affiche_flux` et `flux_clavier` en utilisant ce module.
6. On souhaite ajouter un « finaliseur » à nos flux, c'est-à-dire une fonction (de type `unit -> unit`) qui sera appelée automatiquement lorsque le flux aura été lu en entier (par exemple pour fermer un fichier si le flux est lu à partir d'un fichier). Modifier le type `flux` pour ajouter ce finaliseur, et réécrire la fonction de création d'un flux. Quelle(s) autre(s) modification(s) faut-il faire dans le module ?
7. Quel est l'intérêt d'avoir un type abstrait pour les flux avec finaliseurs ?