

**Modèles de Programmation Séquentielle en OCaml****Module MPSOC (LI313)**

Examen du 1 février 2007

---

*Documents autorisés - Durée 2 heures*

Le sujet se décompose en 3 parties indépendantes. La première partie se compose de 3 exercices indépendants. La deuxième partie se compose de 3 exercices dépendants. Néanmoins on peut répondre aux questions en admettant les réponses précédentes. La troisième partie se compose de 4 exercices.

Il est demandé de séparer les réponses en 2 copies. La première copie pour les parties 1 et 2 et la deuxième copie pour la partie 3.

---

## 1 Programmation fonctionnelle et impérative

**Exercice 1.** Tri fusion d'une liste.

- (1) Écrire une fonction `separer` : `'a list -> 'a list * 'a list` qui partage une liste  $l$  en deux listes  $l_1$  et  $l_2$  telles que les tailles de  $l_1$  et  $l_2$  ne diffèrent que d'un au maximum.
- (2) Écrire une fonction `fusionner` : `int list -> int list -> int list` qui prend en argument deux listes ordonnées d'entiers  $l_1$  et  $l_2$  et renvoie une liste ordonnée  $l$  contenant tous les éléments de  $l_1$  et de  $l_2$ .
- (3) Enfin, écrire une fonction `tri-fusion` : `int list -> int list` utilisant les deux fonctions précédentes pour trier une liste d'entier.

*Indication* : l'algorithme de tri fusion repose sur une approche diviser pour régner : si une liste contient au plus un élément, elle est trivialement ordonnée. Par contre, si elle contient deux éléments ou plus, il suffit de la partager en deux listes plus petites (`separer`) qui seront alors chacune triée par un appel récursif à `tri-fusion`, puis enfin de fusionner les listes résultantes (`fusionner`).

**Exercice 2.** Tri bulle d'un tableau.

- (1) Écrire une fonction `compareVoisins` qui prend en argument un tableau et un indice  $i$ . Elle doit comparer les éléments du tableau aux indices  $i$  et  $i + 1$  afin de les échanger si l'élément qui est à la position  $i$  est supérieur à l'élément qui est à la position  $i + 1$ . Ainsi, après l'appel de la méthode, on sait que la valeur de l'indice  $i$  dans le tableau modifié est inférieur à celle d'indice  $i + 1$ .
- (2) Quel est l'effet d'appels à la fonction `compareVoisins` sur tous les indices successifs d'un tableau ?

- (3) En déduire une fonction `tri-bulle` qui prend en argument un tableau et trie ce tableau (on ne fera pas d'appel à la fonction `compareVoisins` mais on inclura son code dans la fonction `tri-bulle`).

**Exercice 3.** Référence et environnement.

Expliquer la différence entre les deux fonctions `f1` et `f2`.

```
let f1 = let c = ref 0 in          | let f2 = function () ->
  function () ->                  |   let c = ref 0 in
    begin c := !c + 1 ; !c end ;; |     begin c := !c + 1 ; !c end ;;
```

Indiquer ce que renvoient les appels suivants.

```
f1 ();;
f1 ();;
f2 ();;
f1 ();;
f2 ();;
```

## 2 Modules et foncteurs

Le but de ces exercices est de commencer à écrire une librairie d'opérations sur les polynômes en utilisant les modules pour abstraire la structure d'anneau sur les coefficients.

**Exercice 4.** Structure d'anneau.

On donne la signature OCaml suivante permettant de représenter des structures d'anneaux :

```
module type ANNEAU =
  sig type t (* type des éléments de l'anneau *)
    val zero : t
    val unit : t
    val plus : t -> t -> t
    val symm : t -> t
    val mult : t -> t -> t
    val equal : t -> t -> bool
    val tostring : t -> string
  end ;;
```

Le type `t` est le type (abstrait) des éléments de l'anneau. Les champs suivants sont les opérations sur la structure d'anneau. Nous avons ajouté le champ `equal` pour tester l'égalité de deux éléments sans supposer que les éléments en formes canoniques (on ne peut donc pas utiliser l'égalité générique `=`). Le champ `tostring` est utilisé pour afficher les éléments de l'anneau.

- (1) Donner une implémentation de l'anneau `Int` des entiers et de l'anneau `Bool` des Booléens ayant tous deux l'interface `ANNEAU`.
- (2) Écrire un foncteur permettant de créer les anneaux  $\mathbb{Z}/n\mathbb{Z}$  pour  $n$  entier positif arbitraire.

**Exercice 5.** Structure de polynômes.

On donne l'interface de la structure des polynômes :

```
module type POLYNOME = sig
  type c
```

```

type t
val zero : t
val unit : t
val symm : t -> t
val plus : t -> t -> t
val mult : t -> t -> t
val equal : t -> t -> bool
val toString : t -> string
val monome : c -> int -> t
val eval : t -> c -> c
end ;;

```

- (1) Vérifiez que les polynômes ont une structure d'anneau, c'est-à-dire qu'une structure de polynôme peut être vue par OCaml comme une structure d'anneau.
- (2) En déduire une représentation plus concise de la définition de la signature POLYNOME.
- (3) Le type `c` représente le type des coefficients. Quel est à votre avis le sens de la fonction `monome` ? de la fonction `eval` ?
- (4) Nous voulons écrire une implémentation des polynômes qui soit paramétrée par le nom de la variable. Donner la signature du foncteur `Make` qui permet de réaliser une telle abstraction.

**Exercice 6.** Un polynôme en  $X$  et  $Y$  peut être vu comme un polynôme en  $X$  dont les coefficients sont des polynômes en  $Y$  (rappelez-vous qu'un polynôme est en particulier un anneau). On veut s'assurer que cela est bien possible à partir des polynômes à une variable.

- (1) Donner une implémentation naïve du foncteur de signature :

```

module Make2 (A : ANNEAU) (X : VAR) (Y : VAR) : POLYNOME;;

```

dont le sens est évident.

- (2) En fait, on voudrait voir les polynômes à deux variables avec l'interface suivante et non comme des polynômes de polynômes :

```

module Make2 (A : ANNEAU) (X : VAR) (Y : VAR) : sig
  include ANNEAU
  type c = A.t
  val monome : c -> int -> int -> t
  val eval : t -> c -> c -> c
end ;;

```

Donner une implémentation de ce foncteur.

### 3 Interopérabilité avec le langage C

Le but de ce problème est de pouvoir gérer une base de données dont les fonctions ont été codées en C. Cette base de données provient d'un organisme nationale qui aime à ficher les activités de nos concitoyens. Cet organisme vous a récemment fait confiance et a décidé de vous employer pour écrire une bibliothèque de fonctions OCaml permettant de gérer de tels fiches. Le problème est qu'ils ne souhaitent pas donner le code source de leurs fonctions écrites en C et que nous allons devoir tout faire avec le peu d'information qu'ils nous ont données.

**Exercice 7.** Les fiches provenant du monde C sont codées sous la forme d'un pointeur sur une structure contenant une chaîne de caractères représentant le nom de l'individu et un entier long non signé représentant une clé de hachage calculée à l'aide d'une fonction à laquelle nous n'avons pas accès. Des éléments de ce type devront être manipulés dans notre programme en OCaml, nous le représenterons donc sous la forme d'un type abstrait. Plus particulièrement nous avons la définition suivante en C pour le type `conteneur_pt` qui nous intéresse :

```
typedef struct {
    char * name;
    unsigned long int key;
} conteneur;

typedef conteneur* conteneur_pt;
```

Dans notre programme OCaml nous allons aussi manipuler des fiches un peu plus enrichies, elles seront de type `conteneur_ml` et nous permettront la création de fiches C de type `conteneur_pt`.

- (1) Donner la définition du type abstrait `conteneur_c` permettant de manipuler en OCaml des données C de type `conteneur_pt`.
- (2) Nous allons manipuler en OCaml la clé de hachage contenu dans le type C `conteneur_pt`. Donner un type OCaml, que vous nommerez `type_et`, permettant de manipuler une telle donnée (la clé) et expliquer pourquoi vous faites un tel choix.
- (3) Le type `conteneur_ml` représentera des quadruplets d'éléments de types sommes et est donné explicitement par : `type conteneur_ml = name_t*date_t*prof_t*key_t;;`. Chacun des types apparaissant dans la définition de `conteneur_ml` est un type somme contenant deux constructeurs : un constructeur paramétré par un type de base et un constructeur indiquant l'absence de valeur (le champ n'est pas renseigné). Le type `name_t` permet de représenter le nom de l'individu donné sous la forme d'une chaîne de caractères (ainsi le type de base sera `string`), `date_t` sa date de naissance représenté par un entier, `prof_t` sa profession représentée par une chaîne de caractères et enfin `key_t` la clé de hachage représentée à l'aide du type défini à la question précédente. Donner La définition exacte de ces quatre types et l'ordre dans lequel doit être fait toutes les définitions de types que vous avez faites jusqu'ici.

**Exercice 8.** Supposons ici que l'on ait accès aux fonctions OCaml suivantes :

- `ml_key_of_conteneur_c` : `conteneur_c -> key_t` qui prend en entrée un élément de type `conteneur_c` et qui renvoie sa clé de hachage.
- `ml_name_of_conteneur_c` : `conteneur_c -> name_t` qui prend en entrée un élément de type `conteneur_c` et qui renvoie le nom.
- `aux_ml_create_conteneur_c` : `string*int*string -> conteneur_c` qui prend en entrée un triplet représentant successivement le nom, la date de naissance et la profession de l'individu à fichier et qui renvoie l'élément de type `conteneur_c` correspondant. Cette fonction peut créer des fiches de type `conteneur_c` dont le nom est "ERROR" dans le cas où une erreur s'est produit lors du calcul de la clé de hachage (par exemple lorsqu'une collision apparaît dans la base de donnée).

Comme nous l'avons dit plus haut, nos nouveaux employeurs refusent de nous donner accès aux sources des fonctions permettant de créer une clé de hachage. Nous devons donc nous débrouiller avec les fonctions données ci-avant pour pouvoir créer des fiches de type `conteneur_ml` qui soient complètes (i.e. avec une clé de hachage renseignée).

- (1) Écrire la fonction `ml_create_conteneur_c` de type `conteneur_ml -> conteneur_c` qui, si l'une des trois premières composantes de l'entrée n'est pas renseignée lève une exception (que vous aurez définie) et sinon, renvoie l'élément de type `conteneur_c` correspondant.
- (2) Écrire la fonction `ml_create_conteneur_ml_with_key` de qui prend entrée la fiche (de type `conteneur_ml`) d'un individu et qui la renvoie avec la composante de type `key_t` renseignée lorsqu'aucune erreur n'a été détectée. Si une erreur est effectivement détectée cette fonction lèvera une exception (que vous définirez).

**Exercice 9.** Nous allons maintenant écrire les fonctions C permettant de faire le lien entre le monde OCaml et les fonctions C de la bibliothèque à laquelle nous avons accès.

Voici les entêtes des fonctions C auxquelles nous avons accès :

```
char * name_of_conteneur(conteneur_pt cont);
/* renvoie la chaîne de caractère du conteneur */

unsigned long int key_of_conteneur(conteneur_pt cont);
/* renvoie la clé de hachage du conteneur */

conteneur_pt create_conteneur(char * name, long int birth, char * prof);
/* création d'un conteneur à partir d'un nom, une date de naissance
   et une profession */
```

Pour vous aider à créer ces fonctions C nous vous rappelons quelques lignes du manuel de OCaml :

- les  $n$ -uplets OCaml sont représentés par des `value` blocs (de tag 0) en C et chacune des valeurs des composantes peut être obtenue par la macro `Field` (tout se passe comme pour un tableau) ;
- les valeurs OCaml de type `string` peuvent être transformées en un pointeurs de type `char *`. Ceci se fait à l'aide de la macro `String_val(v)` qui prend un élément de type `value` contenant une chaîne de caractères OCaml (type `string`) et qui renvoie le pointeur C correspondant. *Vice versa*, un pointeur `s` de type `char *` peut être transformée en un élément de type `value` contenant une chaîne de caractères OCaml à l'aide de la macro `caml_copy_string(s)` qui renvoie une telle valeur ;
- le type `value` peut représenter un quelconque pointeur C (en particulier il est totalement possible de représenter un élément `cont` de type `conteneur_pt` par le type `value` en effectuant directement un cast de la forme `(value) cont`).

- (1) Écrire les fonctions C d'entêtes
 

```
value stub_name_of_conteneur(value cont_pt)
(resp. value stub_key_of_conteneur(value cont_pt))
```

 qui permet d'utiliser la fonction C `name_of_conteneur` (resp. `key_of_conteneur`) à partir d'une fonction OCaml `aux_ml_name_conteneur_c` (resp. `aux_ml_key_of_conteneur`) de

`type conteneur_c -> string` (resp. `conteneur_c -> type_et` où `type_et` est le type que vous aurez défini plus haut). Donner aussi la déclaration de la fonction OCaml `aux_ml_name_conteneur_c` (resp. `aux_ml_key_of_conteneur_c`) et en déduire la définition de `ml_name_conteneur_c` (resp. `ml_key_of_conteneur_c`) définie à la question précédente.

- (2) Écrire la fonction C d'entête `value stub_create_conteneur` (value `triplet_cont_ml`) qui permet d'utiliser la fonction C `create_conteneur` à partir de la fonction OCaml `aux_ml_create_conteneur_c`. En déduire la définition de la fonction `aux_ml_create_conteneur_c`.

**Exercice 10.** Expliquer comment vous organiseriez vos fichiers sources C et OCaml (sachant que les binaires des fonctions C auxquelles vous avez accès sont réunis dans le fichier compilé `secret.o`). Donner les commandes exactes de compilation pour votre projet.