NFP 119 : Examen Programmation Fonctionnelle

Session du 30 Juin 2008

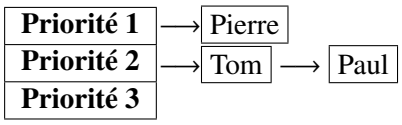
Il vous est recommandé de rédiger de façon concise et précise, tout en écrivant lisiblement. Il sera tenu compte de la présentation dans la note de la copie. Tous les documents sont autorisés.
Le barème n'est donné qu'à titre indicatif.

Partie I

Exercice 1 (6 points)

On veut gérer la liste d'attente dans un service d'urgences à l'aide d'une **file de priorité de patients**. Un **patient** est caractérisé par son nom, son numero de sécurité sociale et sa priorité, un entier compris entre 1 et n, n étant la priorité maximale (i.e. très urgent). Les patients sont **insérés** et **enlevés** de la file selon leur priorité.

Si un patient avec priorité k arrive, il sera rangé en dernier parmi les patients de même priorité. On enlève un patient dans la file s'il est le premier arrivé parmi tous ceux ayant la priorité maximale. **Exemple** : On suppose que dans le service la priorité maximale est 3. Supposons que 3 patients arrivent dans l'ordre suivant : Pierre a priorité 1, Tom a priorité 2 et Paul a priorité 2. Si on organise la file en trois sous-files correspondant à chacune des 3 priorités, on aura :



Il n'y a aucun patient de priorité 3. Le patient de priorité maximale est donc Tom. Si on affiche la liste de patients dans l'ordre dans lequels ils seront traités cela devra donner : Tom, Paul, Pierre.

Dans cet exercice vous devez implanter en Ocaml

1. Le type `patient` défini comme un enregistrement.
2. Une variable `filePatients` composée d'un tableau de n listes vides de patients. Cette variable sera donnée en paramètre aux fonctions que vous devez implanter, et modifiée par celles-ci.
3. les trois opérations suivantes :
   (a) `insere` prend un patient et une file en paramètres et insère celui-ci dans la file selon sa priorité ;
   (b) `enleve` prend une file en paramètre, et enlève de celle-ci le patient de plus grande priorité, et retourne ce patient en résultat. Cette opération lève une exception si la file est complétement vide.
   (c) `afficheFile` affiche tous les patients d'une file dans l'odre où ils seront traités.
4. Vous écrirez un petit programme de création d'une file de priorité `fpp` en y insérant un par un trois patients, puis vous afficherez cette file.

5. On aimerait rendre les fonctions précédentes génériques pour n'importe quelle type de donnée à insérer dans la file, porvu que cette donnéé soit munie d'une priorité. Peut-on définir un type des files de priorité qui généralise le tableau de listes de patients utilisé ici ? Comment ?

Solution :

```
type patient = {nom: string; nss: int; priorite: int}

let affichePatient p =
 print_string"Nom␣="; print_string p.nom;
 print_string"␣␣Priorite␣="; print_int p.priorite;
print_newline();;


type filePrio = patient list array

let insere p (file: filePrio) =
 let rec insereFin p l =
  match l with
    []  -> [p]
  | x::r -> x::(insereFin p r)
 in file.(p.priorite-1) <- insereFin p file.(p.priorite-1);;

 let enleve (file: filePrio) =
  let rec maxPrio i =
  if i<0 then failwith "enleve"
  else if (file.(i) <> []) then
           let max = List.hd file.(i) in
           (file.(i) <- List.tl file.(i)); max
  else maxPrio (i-1)
in maxPrio (Array.length file-1);;

let afficheFile f =
  for i=(Array.length f-1) downto 0
  do  List.iter affichePatient f.(i) done
;;


let p1 = {nom = "paul"; nss= 1; priorite= 1};;

let p2 = {nom = "tom"; nss = 2; priorite= 1};;

let p3 = {nom = "titi"; nss = 4; priorite = 2};;

let p4 = {nom = "pppp"; nss= 5; priorite= 2};;


let createFile n = Array.create n ([]: patient list);;

let f = createFile 3;;

insere p1 f; insere p2 f; insere p3 f; insere p4 f;;
```

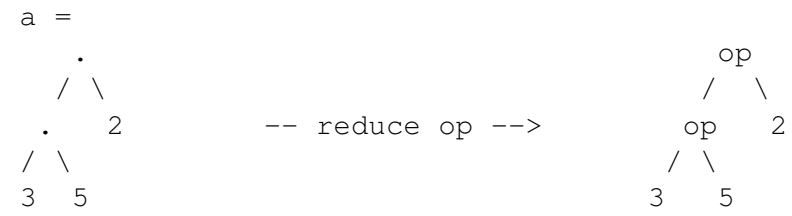
```
(* 5 *)

type 'a cellAvecPriorite = {cont: 'a; priorite: int}

(* Avec ce type, le code ne change pas! C'est l'injection des donnees *)
(* qui change *)
```

Exercice 2 (4 points)

Considérez la définition d'arbres binaires polymorphes donnée plus bas :

```
type 'a arbre = Feuille of 'a
              | Noeud of 'a arbre * 'a arbre
```

où les données de l'arbre se trouvent uniquement dans les feuilles. Ecrivez une fonction `reduce` qui prend en paramètre une opération *op* : $'a * 'a \rightarrow 'a$ et qui renvoie le résultat d'appliquer l'opération *op* sur chaque noeud de l'arbre, tel qu'illustré par l'exemple suivant :



Si par exemple, `op = +`, l'appel `reduce op a` renvoie 10 ; si `op = max`, l'appel `reduce op a` renvoie 5.

1. Donnez une définition de la fonction `reduce` et donnez son type.
2. Utilisez cette fonction (avec un appel pertinent), afin de dériver deux autres fontions : la fonction qui calcule la somme de tous les entiers d'un arbre ; la fonction qui renvoie le patient avec la plus grande priorité dans un arbre de patients (les patients sont décrits dans partie I, exercice 1).

Solution :

```
let rec reduce op a =
 match a
 with Feuille e -> e
  |   Noeud (g,d) -> op (reduce op g) (reduce op d)
;;

let sommeIntArbre a = reduce (+) a;;

let plusPrioritaire a =
 let maxPrio p1 p2 = if p1.priorite > p2.priorite then p1 else p2
in reduce maxPrio a;;
```

Partie II

Exercise 1 (2 points) *Soit* `f` *la fonction suivante (x et y étant des entiers) :*

```
let rec f x = if x>0 then x + (f (x-1)) else 0
```

Démontrez $\forall x,\ (f\ x) \geq x$.
Suggestion : Prouvez la propriété séparément pour $x < 0$ *et* $x \geq 0$. □

Exercise 2 (2 points) *Soit* `g` *la fonction suivante :*

```
let rec g l =
  match l with
    | [] -> []
    | n::l' -> (n+1) :: (g l')
```

Démontrez la propriété suivante : $\forall l, |g\ l| = |l|$.
(Rappel : $|g\ l|$ *et* $|l|$ *représentent respectivement la longueur des listes* `g` *l et l).* □

Exercise 3 (4 points) *Soit* `modulo` *la fonction suivante, qui retourne le reste de la division entière de ces deux arguments :*

```
let rec modulo a b = if (a<b) then a else (modulo (a-b) b);;
```

1. *Donnez l'équation de* `modulo`.
2. *Démontrez que* $\forall x \in \mathbb{N},\ (\forall y > 0,\ \exists q \geq 0,\ x = qy + (\texttt{modulo}\ x\ y))$.

□

Exercise 4 (2 points) *Soit* `ack` *la fonction suivante :*

```
let rec ack x y =
  if x = 0 then y + 1
  else if y = 0 then (ack (x-1) 1)
  else (ack (x-1) (ack x (y-1)))
```

Cette fonction termine pour tout $x, y \in \mathbb{N}$, *démontrez le.* □