Algorithmique-Programmation : AP1 - Rattrapage - 7 février 2011

Sans documents - durée : 1h45 -
Calculette et ordinateur : sans objet

Les exercices sont indépendants. Ne vous bloquez donc pas sur une question.

Pour répondre à certaines questions, il se peut que vous ayez besoin de définir des fonctions auxiliaires. Dans ce cas indiquez clairement ce que font ces fonctions auxiliaires en donnant par exemple leur interface.
Pour les fonctions explicitement demandées dans l'énoncé, n'écrivez leur interface que si celle-ci est explicitement demandée dans l'énoncé.

Exercice 1 (**Session à compléter**)

Donner les réponses de l'interprète OCaml pour chaque phrase. On suppose que les phrases sont entrées dans cet ordre. Si une phrase est mal typée, indiquez *erreur de typage* et expliquez pourquoi en quelques mots. Inutile de recopier les phrases, donnez la réponse de Ocaml (au moins type et valeur) en face du numéro de la phrase.

```
1# let n = "ab" in n ^ n;;

2# n;;

3# let n = 15 in let n = true in n + 1;;

4# let ff a = if a < 0 then a else a+1;;

5# (ff 3 , ff 2);;

6# let tt (ff, x) = if (ff x) then x else x+1;;

7# tt ((function x -> (x mod 2) = 0), 1);;

8# let uu g x =  if x<0 then g (-x) else g x;;

9# let vv g x = g (if x<0 then (-x) else x) ;;

10# uu (function x -> x+1) 3;;

11# vv (function x -> x+1) 3;;

12# let fonct = uu (function x -> 2*x)  ;;

13# [fonct 3; fonct (-3)];;

14# let ww x y z = (x y) + (x z);;

15# let zz x y z = (x y) ^ (x z);;

16# let yy x y z t = t (x y) (x z);;
```

Exercice 2

Question 1

Ecrire une fonction récursive `enlever` qui prend en paramètre une liste `l` et un élément `e` et retourne la liste `l` dont on a ôté toutes les occurrences de `e`.

Quel est le type le plus général de la fonction `enlever` ?

Question 2

Réécrire la fonction précédente `enlever` en utilisant la fonctionnelle `filter`.

Question 3

Réécrire la fonction précédente `enlever` en utilisant la fonctionnelle `fold_left` ou `fold_right`.

Question 4

Compléter l'interface suivante :

```
(* interface toto
type : ..........
args : l
pre : ...........
post : ........
raises : ......
*)
let nettoyer l = enlever [] l;;
```

Exercice 3 (**Course de ski**)

On désire établir le classement final d'une course de ski. On dispose pour cela de la liste des temps de course de chaque skieur, représentée par une liste de couples de la forme (n, t) où n est le numéro de dossard d'un skieur (un entier de type int) et t son temps de course en millisecondes (de type int).

Question 5

En utilisant la méthode de tri par insertion, écrire une fonction `classement` qui prend la liste des temps de course des skieurs et retourne la liste ordonnée dans l'ordre croissant des temps de course. Les exaequo seront classés dans l'ordre croissant des numéros de dossard.

Question 6

En utilisant la fonction précédente écrire une fonction premier qui prend en paramètre la liste des temps de course (non ordonnée) et retourne le numéro de dossard du gagnant.

Exercice 4

Question 7

Écrire une fonction `until` qui prend en paramètre une valeur a, une fonction f et un prédicat p (fonction qui retourne soit true, soit false) et qui retourne la liste $[a; f(a); f(f(a)); \dots f^k(a)]$ telle que $\forall\ 0 \leq i < k, p(f^i(a)) = false \wedge p(f^k(a)) = true$. On remarquera que dans tous les cas, a est élément de la liste résultat. Le dernier élément de la liste vérifie la condition modélisée par le prédicat p.

Question 8

Quel est le type le plus général de la fonction `until` ?

Question 9

A l'aide de la fonction précédente, écrire la fonction `intervalle` qui prend en paramètre deux entiers n et m et qui retourne la liste des entiers compris (au sens large) entre n et m, si $n \leq m$, sinon la liste résultat est vide. Par exemple `intervalle 2 8` retourne la liste [2;3;4;5;6;7;8] et `intervalles 2 0` retourne la liste [].

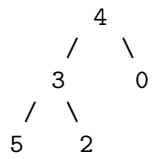
Exercice 5 (**Arbres**)

On définit le type des arbres binaires d'entiers ce la façon suivante :

```
type arbre = Feuille of int | Noeud of int * arbre * arbre
```

Question 10

Définir l'arbre ci-dessous en Ocaml. On l'appellera `ex1`.



Question 11

Ecrire une fonction `elements` qui renvoie la liste des entiers (aux feuilles et aux noeuds) d'un arbre, en suivant un parcours *Gauche Racine Droite*. Par exemple, `elements ex1` retourne la liste `[5; 3; 2; 4; 0]`.

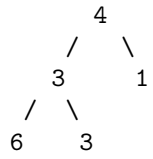
Vous en donnerez une version qui utilise la concaténation de listes (@) et une autre version qui n'utilise que des ajouts en tête (cette 2ème version pourra utiliser un accumulateur si besoin).

Question 12

Ecrire une fonction `feuilles` qui calcule le nombre de feuilles d'un arbre. Par exemple, `feuilles ex1` retourne 3.

Question 13

Ecrire une fonction `mapf` qui prend 2 paramètres : un arbre `a` et une fonction `f`. La fonction reconstruit un arbre en appliquant la fonction `f` sur les feuilles de l'arbre `a` et garde les valeurs aux noeuds de `a`. Par exemple `mapf succ ex1` retourne l'arbre ci-dessous (`succ` est la fonction successeur) :



Question 14

Quel est le type de la fonction `mapf` ?

Question 15

a- Modifier le type des arbres de manière à ce qu'il devienne polymorphe.

b- Quel est alors le type d'un arbre étiqueté par des chaînes de caractères ?

c- Quel sera, dans ce contexte, le nouveau type de la fonction `elements` qui retourne la liste des valeurs d'un arbre ?

d- Quel sera, dans ce contexte, le nouveau type (le plus général) de la fonction `mapf` ?

Rappel : fonctionnelles classiques sur les listes :

```
let rec map f li = match li with
  [] -> []
| x::l -> (f x)::(map f l);;

let rec filter p li = match li with
  [] -> []
| x::l -> if (p x) then x::(filter p l) else (filter p l);;

let rec fold_right f l e = match l with
  [] -> e
| a::r -> f a (fold_right f r e);;

let rec fold_left f e l = match l with
  [] -> e
| a::r -> fold_left f (f e a) r;;
```
