info421 : Programmation fonctionnelle
TD 5 : récursion terminale

Responsable : Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
email : Pierre.Hyvernat@univ-savoie.fr
http://lama.univ-savoie.fr/~hyvernat/

Exercice 1 : petits exercices

Il est souvent possible de transformer une fonction récursive en fonction récursive terminale en utilisant un *accumulateur* qui permet de faire passer les résultat entre les appels récursifs. Ceci permet d'économiser de la mémoire car Caml ne sauvegarde pas le contexte lors des appels récursifs terminaux.

Question 1. Programmez la fonction `somme : int -> int` qui calcule la somme $1+2+...+n$ de manière récursive terminale.

Question 2. Programmer la fonction `factorielle` de manière récursive terminale.

Question 3. On peut écrire une fonction `iter` qui itère une fonction un certain nombre de fois : `iter : ('a -> 'a) -> int -> ('a -> 'a)`.

Une version de cette fonction est donnée par :
```
let rec iter f n x =
  if n = 0
  then x
  else f (iter f (n-1) x)
```
Programmez une version récursive terminale pour cette fonction.

Question 4. La fonction `fold_right` sur les listes est définie comme suit :
```
let rec fold_right f l o =
  match l with
   [] -> o
 | a::l -> f a (fold_right f l o)
```
et son type est `('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`.

Écrivez une fonction *similaire* récursive terminale : `fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`

Question 5. Écrivez une fonction `intervalle` qui prend deux arguments entiers et renvoie l'intervalle des entiers compris entre ces deux nombres :
```
# intervalle 3 10;;
- : int list = [3; 4; 5; 6; 7; 8; 9]
```

Question 6. Ré-écrivez la fonction qui renverse une liste.

Question 7. Ré-écrivez la fonction qui calcule la taille d'une liste.

Exercice 2 : calcul de la puissance d'un nombre

Question 1. Écrivez une fonction `puissance : float -> int -> float` qui permet de calculer x^n...

Combien d'opérations sont nécessaires pour faire ce calcul ?

Quelle quantité de mémoire est nécessaire pour faire ce calcul ?

Question 2. Une version un peu plus efficace (pour la mémoire) est d'utiliser un *accumulateur*. Programmez une fonction `puissance_acc x n acc : float -> int -> float -> float` qui pour les arguments x, n et a calcule $a \times x^n$.

Pour éviter de garder des valeurs en mémoire entre les appels récursifs, débrouillez-vous pour faire des appels récursifs *terminaux*.

Combien d'opérations sont nécessaires pour faire ce calcul ?

En sachant que Caml ne sauvegarde pas le contexte lors des appels récursifs terminaux, quelle quantité de mémoire est nécessaire pour faire ce calcul ?

Servez-vous de cette fonction pour redéfinir une fonction puissance.

Question 3. Pour limiter le nombre de calculs lors du calcul de la puissance, on va maintenant utiliser l'astuce suivante :

$$\begin{cases} x^0 = 1 \\ x^{2n} = (x^n)^2 \\ x^{2n+1} = x(x^n)^2 \end{cases} .$$

Écrivez une fonction `puissance_chinoise : float -> int -> float` pour utiliser cette remarque. (Pas besoin de faire une version récursive terminale pour le moment.)

Question 4. Essayer maintenant de faire une fonction qui calcule la puissance avec :
- une complexité en temps similaire la celle de la question 3,
- une complexité en espace similaire à celle de la question 2.

Pour ceci, utilisez la formule

$$\begin{cases} x^0 = 1 \\ x^{2n} = (x^2)^n \\ x^{2n+1} = x(x^2)^n \end{cases} .$$