info421 : Programmation fonctionnelle
TD 2 : récursion, listes

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

Exercice 1 : Déclarations et environnement

Question 1. Dans le morceau de code suivant, donnez la liste des variables actives de l'environnement aux endroits `(*1*)`, `(*2*)`, `(*3*)`, `(*4*)`, `(*5*)`, `(*6*)`, `(*7*)` et `(*8*)`.

Précisez, pour chaque variable, s'il s'agit d'une *définition* (et donnez la valeur correspondante et le type correspondant), ou bien d'un *paramètre* (et donnez son type).

```
(*0*) (* on suppose que l'environnement est vide *)
let z = 17;; (*1*)
let n = z (*2*) / 2;; (*3*)
let g n x = (*4*)
  if x<0
  then n + z
  else n - z;;
let rec f n = (*5*)
  if n < 10
  then n (*6*)
  else let m = 32 in (f (n-7) + f(n-10)) mod m (*7*);;
(*8*)
```

Exercice 2 : fonctions récursives

Question 1. Les coefficients binomiaux sont définis par

$$\binom{0}{k} = 0 \qquad \binom{n}{0} = 1 \qquad \binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$$

Définissez une fonction récursive à deux arguments `bin : int -> int -> int` qui calcule les coefficients binomiaux.

Comment pouvez-vous calculer le nombre d'additions utilisées par Caml pour calculer le coefficient binomial $\binom{17}{12}$ en utilisant votre fonction ?

Question 2. Les nombres de Catalan sont définis par la relation de récurrence

$$C_0 = 1 \qquad \text{et} \qquad C_n = \sum_{i=0}^{n-1} C_i \times C_{n-1-i}$$

Définissez une fonction récursive qui permet de calculer les nombres de Catalan.

Exercice 3 : listes.

Question 1. Dans le morceau de code suivant, donnez la liste des variables actives de l'environnement aux endroits `(*1*)`, `(*2*)` et `(*3*)`.

Précisez, pour chaque variable, s'il s'agit d'une *définition* (et donnez la valeur correspondante), ou bien d'un *paramètre* (et donnez son type).

```
(*0 on suppose que l'environnement est vide *)
let rec f n l = (*1*) match l with
      []   ->  n
    | a::l -> let m = n+1 in
              (*2*) f m l;;
(*3*)
```

Question 2. Écrivez une fonction `longueur : 'a list -> int` qui calcule la longueur d'une liste.

Question 3. Écrivez une fonction `applique` qui prend une fonction et une liste en arguments, et qui applique la fonction à tous les éléments de la liste.

Quel est le type de cette fonction ?

Question 4. Écrivez une fonction `concatene : 'a list -> 'a list -> 'a list` qui concatène deux listes ensemble.

Question 5. Écrivez une fonction `renverse : 'a list -> 'a list` qui renverse une liste.

Que pensez-vous de la complexité de votre fonction ?

Question 6. Toutes les fonctions précédentes* suivent le schéma suivant :

- tant que la liste n'est pas vide, on fait un appel recursif sur la queue de la liste, et on applique une fonction "`f`" sur :
  - . le premier élement de la liste
  - . le résultat de l'appel récursif,
- quand on arrive sur la liste vide, on renvoie une valeur de base "`o`"

Une version générique de ce principe est la fonction :

```
let rec liste_rec f l o =
  match l with
    x::xs -> let r = (liste_rec f xs o)
             in f x r
  | [] -> o
```

Rajoutez des annotations de type sur la déclaration de cette fonction et donnez son type complet tels que Caml l'afficherait.

Question 7. Reprogrammez les fonctions précédentes uniquement à partir de la fonction `liste_rec`.

Remarque : toutes ces fonctions sont définies dans la librairie `List` :

- `longueur` s'appelle `List.length`,
- `applique` s'appelle `List.map`,
- `concatene` s'appelle `@` et on l'utilise de manière infixe,
- `renverse` s'appelle `List.rev`,
- `liste_rec` s'appelle `List.fold_right`.

* sauf peut-être la fonction `renverse`, suivant la manière dont vous l'avez programmée