Institut National Universitaire Champollion

TP 6 - Les listes

Programmation fonctionnelle en Caml
L3 INFO - Semestre 6

Rappel - La liste vide s'écrit `[ ]` et sert de cas de base dans la plupart des fonctions récursives sur les listes. Une liste non vide peut-être filtrée par `(a : :q)` où a est l'élément de tête et l le reste de la liste, une liste possédant au moins deux éléments peut être filtrée par `(a : :b : :l)`.

1 - Premières fonctions non récursives sur les listes

Écrire les fonctions suivantes :

1. `deuxieme : 'a list -> 'a` qui extrait le deuxième élément d'une liste.
2. `auMoinsTrois : 'a list -> bool` testant si une liste a au moins trois éléments.
3. `sommeTrois : int list -> int` qui fait la somme des trois premiers éléments d'une liste d'entiers
4. `troisEstPair : int list -> bool` testant si le troisième élément d'une liste est pair.
5. `ajoutDeuxFois : 'a * 'a list -> 'a list` qui ajoute deux fois un élément en tête d'une liste
6. `permute : 'a list -> 'a list` qui échange les deux premiers éléments d'une liste.

2 - Premières fonctions récursives sur les listes

Écrire les fonctions suivantes :

1. `construitListe : int -> int list` qui, à un entier n, associe la liste $[n; n-1; \ldots; 1; 0]$.
2. `longueur : 'a list -> int` qui calcule la longueur d'une liste.
3. `dernier : 'a list -> 'a` qui donne le dernier élément d'une liste.
4. `somme : int list -> int` qui calcule la somme des éléments d'une liste donnée.
5. `taillePaire : 'a list -> bool` qui teste (sans utiliser la fonction longueur) si une liste possède un nombre pair d'éléments.
6. `rangImpair : 'a list -> 'a list` permettant de construire la liste des éléments de rang impair d'une liste donnée.

3 - Encore des fonctions récursives sur les listes

Écrire les fonctions suivantes :

1. `appartient : 'a * 'a list -> bool` qui teste si un élément appartient à une liste.
2. `maximum : 'a list -> 'a` qui calcule le maximum d'une liste d'entiers.
3. `occurrences : 'a * 'a list -> int` qui, étant donnés un élément et une liste, calcule le nombre d'occurrences de l'élément dans la liste.
4. `fois2 : int list -> int list` qui, étant donnée une liste d'entiers, construit la liste des doubles des entiers de la liste.
5. `insere : int * int list -> int list` qui insère un entier dans une liste d'entiers ordonnée dans l'ordre croissant.

4 - Encore des fonctions récursives sur les listes

Écrire les fonctions suivantes :

1. `ieme : ieme : int * 'a list -> 'a`
2. `prendre : int * 'a list -> 'a list` qui, étant donnés un entier p et une liste l, retourne la liste des p premiers éléments de l.
3. `enleve : int * 'a list -> 'a list` qui, étant donnés un entier p et une liste l, retourne la liste privée des p premiers éléments de l.
4. `melange : 'a list * 'b list -> ('a * 'b) list` qui prend une paire de listes et retourne la liste des paires correspondante. Si les deux listes n'ont pas la même longueur, on s'arrêtera à la liste la plus courte.

5 - Suite de Fibonacci

Vous connaissez tous la suite de Fibonacci : $(1,1,2,3,5,8,13,\ldots)$ qui commence par $1,1$ et dont chaque terme est la somme des deux précédents.

Écrivez la fonction `Fibonacci : int -> int list` qui associe à l'entier n la liste des n premiers termes de la suite de Fibonacci. Pour cela, vous définirez localement une fonction récursive qui au couple d'entiers (a,b) associe, dans le cas général, la liste de Fibonacci commençant par $[a;b;\ldots]$.

```
Fibonacci 8 ;;
[1 ;1 ;2 ;3 ;5 ;8 ;13 ;21]
```

6 - Fusion de listes croissantes

1. Écrire une fonction `estCroissante : 'a list -> bool` qui indique si les éléments consécutifs d'une liste sont bien ordonnés dans l'ordre croissant.
2. Écrire une fonction `fusion : 'a list * 'a list -> 'a list` qui, étant données deux listes supposées croissantes, renvoie la fusion de ces deux listes, toujours ordonnée croissante.

7 - Nombres premiers par le crible d'Ératosthène

Nous souhaitons écrire un programme permettant d'obtenir une liste de couples de nombres premiers jumeaux, c'est à dire des couples (a,b) avec a et b premiers et $b-a=2$. par exemple, $(3,5)$ et $(5,7)$ sont des couples de nombres premiers jumeaux.

1. Écrire une fonction `generer : int -> int list` qui, étant donné un entier n, construit la liste ordonnée des entiers de 2 à n.
2. Écrire une fonction `eliminer : int * int list -> int list` qui, étant donnés une liste et un entier a, élimine tous les multiples de l'entier a dans la liste.
3. Écrire une fonction `eratos : int -> int list` qui, étant donné un nombre entier n, construit la liste des nombres premiers inférieurs ou égaux à n. Pour cela, on utilisera la méthode du crible d'Ératosthène : partant de la liste $[2;\ldots;n]$, son premier élément est premier mais on doit éliminer tous les multiples de cet élément qui eux ne le sont pas. On itère ce procédé jusqu'à la fin de la liste.
4. Écrire une fonction `jumeaux : int list -> (int * int) list` qui, étant donnée une liste de nombres premiers, construit la liste des couples de nombres premiers jumeaux qu'elle contient.
5. Écrire enfin une fonction `listeJumeaux : int -> (int * int) list` permettant de donner la liste des couples de nombres premiers jumeaux inférieurs à une limite n fixée.

8 - Représentation des ensembles par des listes

On choisit de représenter un ensemble fini par la liste Caml de ses éléments. Cette liste ne contient aucune répétition. Ainsi l'ensemble $\{1,2,3\}$ est représenté par une liste contenant 1, 2 et 3 dans un ordre quelconque. $[1;2;3]$, $[1;3;2]$, $[2;3;1]$ ou $[3;1;2]$ sont donc des représentations acceptables de $\{1,2,3\}$.

Écrire les fonctions définissant les opérations ensemblistes suivantes :

1. `appartient : 'a * 'a list -> bool` qui détermine si un élément appartient à un ensemble.
2. `union : 'a list * 'a list -> 'a list` et `intersection : 'a list * 'a list -> 'a list` qui calcule respectivement l'union et l'intersection de deux ensembles.
3. `inclus : 'a list * 'a list -> bool` qui détermine si un premier ensemble est inclus dans un second.
4. `disjoint : 'a list * 'a list -> bool` qui détermine si deux ensembles sont disjoints.
5. `egaux : 'a list * 'a list -> bool` qui détermine si deux ensembles sont égaux.
6. `complement : 'a list * 'a list -> 'a list` qui calcule le complémentaire d'un premier ensemble dans un second.
7. `ensemble : 'a list -> 'a list` qui, à partir d'une liste quelconque, construit une liste contenant les mêmes éléments mais privés de leurs redondances
8. `parties : 'a list -> 'a list list` qui construit l'ensemble des parties d'un ensemble