

Introduction à la programmation fonctionnelle

Cours 6

kn@lri.fr
http://www.lri.fr/~kn

Au menu

On va présenter plusieurs algorithmes avancés sur les listes :

- ◆ Affichage générique d'une liste
- ◆ `List.filter`, `List.map`, `List.fold_left` (leur code et équivalences)
- ◆ Suppression des doublons dans une liste triée
- ◆ Insertion dans une liste triée
- ◆ Générer toutes les sous-listes d'une liste
- ◆ Tri rapide

Nous allons coder ces fonctions en direct. Leur code sera ensuite mis en ligne, *après* le TP.

Lors des révisions, il sera plus intéressant de regarder à nouveau la vidéo du cours plutôt que de lire le fichier de code final.

Sert de préparation à l'examen.

2 / 9

Plan

- 1 Présentation du cours ✓
- 2 Le système Unix ✓
- 3 IPF (2) : fonctions récursives, inférence de types ✓
- 4 IPF (3) : Types structurés, filtrage, polymorphisme, ordre supérieur ✓
- 5 IPF (4) : Exceptions, Listes (1) ✓
- 6 IPF (5) : Fonctions anonymes, Itérateurs, Application partielle ✓
- 7 IPF (6) : Algorithmes avancés sur les listes
 - 7.1 Exercices simples
 - 7.2 Algorithmes avancés

Affichage générique d'une liste

Écrire une fonction `pr_list` :

```
string -> ('a -> unit) -> 'a list -> unit
```

Prenant en argument :

- ◆ une chaîne représentant un séparateur
- ◆ une fonction permettant d'afficher un élément
- ◆ une liste d'éléments

Le séparateur ne doit pas être affiché pour les listes de longueur 0 ou 1.

S'en servir pour écrire une fonction `pr_i11` affichant une liste de listes d'entiers.

4 / 9

1. Donner le code de `List.filter`, `List.map`, `List.fold_left`
2. Écrire `List.rev` en utilisant `List.fold_left`
3. Écrire `List.iter` en utilisant `List.fold_left`
4. Écrire `List.map` en utilisant `List.fold_left` et `List.rev`

5 / 9

Plan

- 1 Présentation du cours ✓
- 2 Le système Unix ✓
- 3 IPF (2) : fonctions récursives, inférence de types ✓
- 4 IPF (3) : Types structurés, filtrage, polymorphisme, ordre supérieur ✓
- 5 IPF (4) : Exceptions, Listes (1) ✓
- 6 IPF (5) : Fonctions anonymes, Itérateurs, Application partielle ✓
- 7 IPF (6) : Algorithmes avancés sur les listes
 - 7.1 Exercices simples ✓
 - 7.2 Algorithmes avancés

1. Insérer une valeur au bon endroit dans une liste triée

```
insert : ('a -> 'a -> int) -> 'a -> 'a list -> 'a list
```
2. Retirer les doublons d'une liste triée

```
uniq : ('a -> 'a -> int) -> 'a list -> 'a list
```

Les deux fonction prennent en paramètre une fonction de comparaison `comp : 'a -> 'a -> int` telle que `comp a b` renvoie un entier négatif si $a < b$, nul si $a = b$ et positif si $a > b$.

On suppose que les éléments de la liste donnée en argument sont ordonnés selon cette fonction `comp`.

La fonction prédéfinie `compare` peut être utilisée comme comparaison générique.

6 / 9

Liste des sous-listes

Écrire une fonction récursive

```
sublists : 'a list -> 'a list list
```

telle que `sublists 1` qui renvoie la liste de toutes les sous-listes de 1. Exemple:

```
sublists [ 1;2;3;4 ] =>
[ []; [ 4 ]; [ 3 ]; [ 3; 4 ];
  [ 2 ]; [ 2; 4 ]; [ 2; 3 ]; [ 2; 3; 4 ];
  [ 1 ]; [ 1; 4 ]; [ 1; 3 ];
  [ 1; 3; 4 ]; [ 1; 2 ];
  [ 1; 2; 4 ]; [ 1; 2; 3 ]; [ 1; 2; 3; 4 ] ]
```

(l'ordre n'est pas significatif)

8 / 9

Quick sort



Algorithme de tri efficace, basé sur le principe « diviser pour régner »

- ♦ Si la liste est vide ou de taille 1, elle est triée
- ♦ Sinon on choisit le premier élément x (appelé pivot):
 - ♦ On partage en deux la liste de départ en l_t (liste des valeurs plus petites que x) et l_g (liste des valeurs plus grandes que x)
 - ♦ On trie récursivement l_t en l_{ts} et l_g en l_{gs}
 - ♦ On renvoie la liste $l_{ts} @ [x] @ l_{gs}$

Attention, le pivot doit permettre de partager à peu près la liste en deux sous-listes de même tailles, sinon l'algorithme n'est pas efficace

Le TP permettra de comparer cet algorithme à un algorithme plus naïf