Option Informatique en Spé MP et MP*
Les fonctions `list_it` et `it_list` (itérateurs sur listes)

Voici les types respectifs de ces fonctions :

```
it_list : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
list_it : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
```

Chacune de ces fonctions prend en argument (à l'ordre près) : une fonction f, une liste $\ell = (x_1, \ldots, x_n)$ et un élément de base b.

- `it_list f b l` calcule $f(f(\ldots(f(b, x_1), x_2), \ldots, x_n)$;
- `list_it f l b` calcule $f(x_1, f(x_2, \ldots, f(x_n, b)\ldots)$.

Bien entendu, si f est une loi associative et commutative, le résultat est le même. Quelques exemples :

- la somme des éléments d'une `int list` se calcule avec `it_list (prefix +) 0` ;
- le produit des éléments d'une `int list` se calcule avec `it_list (prefix *) 1` ;
- la mise à plat d'une liste de listes se fait avec `it_list (prefix @) []`.

Un moyen mnémotechnique simple pour retenir l'ordre des arguments, étant entendu que la fonction est toujours à la première place : dans `it_list`, la liste est en dernier (et le cas de base juste avant) ; dans `list_it` c'est l'inverse. Dessinez les arbres «peignes» qui correspondent aux deux évaluations posibles. Programmation :

```
let rec it_list f a = function
 | [] -> a
 | t::q -> it_list f (f a t) q;;

let rec it_list f a l = match l with
 | [] -> a
 | t::q -> it_list f (f a t) q;;

let rec list_it f l b = match l with
 | [] -> b
 | t::q -> f t (list_it f q b);;
```