ENSEIRB-Matmeca
2024-2025

1ère année - Filière Info
PG104

TD n°4 - Types inductifs : listes

▷ Les listes chaînées ne font pas partie de la bibliothèque standard d'Ecmascript. Afin de pouvoir réaliser les algorithmes récursifs de cette feuille, les fonctions suivantes permettent de construire et déconstruire des listes :

```
import { anyToString }           from '#src/utils/printers.js';
import { head, tail, nil, cons } from '#src/utils/list.functional.api.js';

let aList = cons(1, cons(2, nil));
console.log(anyToString(aList));          // → (|1, 2|)
console.logA(aList);                      // → (|1, 2|)

let anotherList = cons(3, tail(aList));
console.logA(anotherList)                 // → (|3, 2|)
```

Exercice 1: Construction de liste

1. Écrire une fonction `listIota` *récursive* qui prend deux paramètres a et b et qui construit la liste des entiers compris entre a et b.

   Exemple :

   ```
   listIota(1,4); // → cons(1, cons(2, cons(3, nil)))
   listIota(4,1); // → nil
   ```

2. Écrire une fonction `listDisp` *récursive* qui prend une liste en paramètre et renvoie une chaîne de caractères décrivant la liste.

   Remarques : on considérera que la liste ne contient que des entiers ; il est tout à fait possible, voire encouragé d'utiliser une fonction intermédiaire.

   Exemple :

   ```
   listDisp(nil);            // → "(||)"
   listDisp(listIota(1, 4)); // → "(|1,2,3|)"
   ```

Exercice 2: Longueur de liste

Écrire une fonction récursive `listLength` qui accepte en entrée une liste et qui renvoie le nombre d'éléments à l'intérieur de la liste.

Exemple :

```
listLength(nil);            // → 0
listLength(listIota(1, 6)); // → 5
```

Exercice 3: Valeur absolue

Écrire une fonction récursive `listAbsRec` qui accepte en entrée une liste de nombres et qui retourne la liste constituée de la valeur absolue de chaque élément de la liste fournie en paramètre.

Exemple :

```
listAbsRec(cons(1, cons(-2, nil))); // → (|1,2|)
```

Exercice 4: Concaténation de liste

1. Écrire une fonction récursive `listAppend` qui accepte en entrée deux listes et qui renvoie la concaténation de ces deux listes.

   Exemple :

   ```
   listAppend(listIota(1, 3), listIota(1, 4)); // → (|1, 2, 1, 2, 3|)
   ```

2. (*Bonus*) Écrire une version récursive terminale de la fonction `listAppend`.

Exercice 5: Rotation de liste

1. Écrire une fonction `listRotateLeft` qui accepte en entrée une liste et renvoie la rotation circulaire vers la gauche des éléments de cette liste.

   Exemple :

   ```
   listRotateLeft(listIota(1,5));  // (|1,2,3,4|) → (|2,3,4,1|)
   ```

2. (*Bonus*) Écrire une fonction `listRotateRight` qui accepte en entrée une liste et renvoie la rotation circulaire vers la droite des éléments de cette liste.
   Ne pas hésiter à créer des fonctions intermédiaires si besoin.

   Exemple :

   ```
   listRotateRight(listIota(1,5)); // (|1,2,3,4|) → (|4,1,2,3|)
   ```

Exercice 6: Tri par insertion

1. Écrire une fonction récursive `insert` permettant d'insérer un nombre e dans une liste triée l, et qui renvoie la nouvelle liste.
2. Écrire une fonction récursive `listSortInsertNumbers` qui trie une liste de nombres l en utilisant l'algorithme de tri par insertion, et renvoie la liste triée.

3. Reprendre votre code précédent pour écrire une fonction générique de tri par insertion qui prenne en entrée une liste l et un opérateur de comparaison *op*.
   L'utiliser pour réaliser une fonction qui trie des chaînes de caractères sans faire de différence entre les majuscules et les minuscules.

Exercice 7: Tri fusion

1. Écrire une fonction récursive `listMerge` prenant en paramètres deux listes de nombres supposées triées, et qui renvoie la liste fusionnée.

   Exemple :

```
const left  = cons(1, cons(6, cons(8, cons(47, cons(87, nil)))));
const right = cons(3, cons(8, cons(9, cons(21, cons(50, nil)))));
listMerge(left, right); // → (|1,3,6,8,8,9,21,47,50,87|)
```

2. Écrire une fonction `listSplit` prenant une liste l de nombres en paramètre, et retournant deux listes contenant chacune la moitié des éléments de l.

   Exemple :

```
const l4 = cons(3, cons(8, cons(9, cons(21, cons(50, nil)))));
listSplit(l4) // → [ (|3,8,9|), (|21,50|) ]
```

3. Écrire une fonction récursive `listMergeSort` qui trie une liste de nombres l en utilisant l'algorithme de tri fusion, et renvoie la liste triée.

Exercice 8: Listes et palindromes

Écrire une fonction récursive `listIsPalindrome` prenant en paramètres une liste d'éléments, et renvoyant vrai si la liste est un palindrome. Discuter la complexité et proposer éventuellement différentes méthodes.