```
        new List("Les principes des langages de programmation",
        null)));
```

et utiliser la fonction

```
System.out.println(mem("Le discours de la méthode",b));
```

Le programme affiche alors `true`.

La fonction `mem` ci-dessus est programmée récursivement. Il est également possible de la programmer avec une boucle `while`

```
static boolean mem (String s, List b) {
 while (b != null) {if (equal(s,b.hd)) return true; b = b.tl;}
 return false;}
```

Exercice 6.1

Combien d'opérations demande la recherche d'un élément dans une liste ? Et l'ajout d'un nouvel élément ? Et sa suppression ?

Exercice 6.2

Un *multiensemble* est un ensemble dans lequel un même élément peut apparaître plusieurs fois. C'est donc formellement une fonction d'un ensemble vers l'ensemble des entiers naturels qui à chaque objet associe une multiplicité. On peut représenter les multiensembles finis comme des listes, en autorisant les répétitions.

Écrire une fonction qui prend en argument deux listes et calcule le cardinal de l'intersection des deux multiensembles représentés par ces listes.

Écrire une fonction qui prend en argument deux listes et calcule le nombre d'éléments qui apparaissent à la même place dans les deux listes.

Écrire un programme qui cache au Mastermind, c'est-à-dire qui tire au hasard une liste de `n` pions choisis parmi des pions de `p` couleurs, et, à chaque coup, demande à l'utilisateur une liste de la même forme et indique le nombre de pions à la bonne place et à la mauvaise place dans la proposition du joueur.

6.1.2 Les listes d'associations

Une *liste d'associations* est une liste de couples qui est fonctionnelle, c'est-à-dire telle que pour tout `k`, il existe au plus un élément `v` tel que le couple `(k,v)` appartienne à la liste. Le premier élément d'un couple de la liste s'appelle une *clé* et le second une *valeur*. Les listes d'associations sont un moyen de représenter des fonctions de domaine fini, comme les dictionnaires.