Sauf mention contraire, il est interdit d'utiliser des méthodes des exercices précédents pour résoudre l'exercice courant!

Les boucles sont interdites.

1 Rappels

On considère le squelette de classe liste suivant vu en cours.

```
class Liste{
  private int val;
  private Liste suiv;

  public Liste(int x){//construit la liste avec un entier x
    this.val = x;
    this.suiv = null;
  }

  public Liste(int x, Liste l){
    this.val = x;
    this.suiv = l;
  }

  public static boolean estVide(Liste l){
    return l==null
  }

  Liste copie(Liste l){
    //action : retourne une copie de l (en recopiant tous les maillons)

    if(estVide(l)){
      return null;
    }
    else{
      return new Liste(l.v, copie(l.suiv));
    }
  }

  public static String toString(Liste l){
        if(estVide(l))
            return "";
        else
            return l.v+"␣"+toString(l.suiv);
    }
}
```

On rappelle que, lorsque dans une spécification on vous ajoute une contrainte du type "il est demandé de ne créer que x maillons", cela signifie que le nombre de fois où vous appelez un constructeur de Liste est x.

Exercice 1. Longueur

Question 1.1.
Ecrire une fonction `int longueur(Liste l)` qui retourne la longueur l, c'est à dire le nombre d'entiers qu'elle contient.

Exercice 2. Croissant

Question 2.1.
Ecrire une fonction `boolean croissant(Liste l)` qui retourne vrai ssi les entiers de la l sont triés par ordre croissant (au sens $\leq$) .

Exercice 3. AjoutFin, et réflexion sur le type de retour void

Question 3.1.
Ecrire une fonction `Liste ajoutFinV1(Liste l, int x)` qui retourne une liste correspondant à l, dans laquelle on ajoute x à la fin. De plus, on demandera de ne créer qu'un seul maillon.

Question 3.2.
Finalement, pourquoi le code précédent de ajoutFin ne fournit pas la spécification suivante :
`void` `ajoutFinV2(Liste l, int x)`, modifie la liste l en ajoutant x à la fin.

Question 3.3.
Adapter le code précédent pour écrire la fonction suivante :
`void ajoutFinV2(Liste l, int x)`, qui, étant donnée une liste l non vide, modifie l en ajoutant x à la fin.

La conclusion est donc que cette modélisation des listes (avec la liste vide modélisée par null, et toutes les méthodes en static) a pour inconvénient que les méthodes de type de retour void ayant vocation à modifier un paramètre de type liste posent problème pour la liste vide. C'est donc pour cela qu'au lieu d'écrire des méthodes `void m(Liste l, ...)`, on préfèrera écrire `Liste m(Liste l, ...)`, puis faire l'appel `l = m(l,..)`.

Question 3.4.
Ecrire une fonction `Liste ajoutFinV3(Liste l, int x)` qui retourne une liste indépendante correspondant à l, dans laquelle on ajoute x à la fin. Remarque : il est inutile d'utiliser `copie`.

Exercice 4. Concaténation

Question 4.1.
Ecrire une méthode `Liste concatV1(Liste l1, Liste l2)` qui retourne une liste contenant tous les éléments de l1, puis tous ceux de l2. De plus, on demandera de ne créer aucun maillon.

Question 4.2.
La liste retournée par concatV1 n'est pas indépendante de l1 (ni de l2). Constatons les effets de cela : que voit-on affiché si l'on éxecute le programme suivant ?

```
Liste c1 = creerListe(new int []{1,2,3});//c1 = (1,2,3) (on suppose que la méthode créerListe existe)
Liste c2 = creerListe(new int []{4,5,6});//c2 = (4,5,6)
Liste resc = concatV1(c1, c2);
System.out.println(toString(resc));
c1.suiv.v = 20;
c2.suiv.v=50;
System.out.println(toString(resc));
```

Question 4.3.
Ecrire une méthode `Liste concatV2(Liste l1, Liste l2)` qui retourne une liste indépendante de l1 et l2, et contenant tous les éléments de l1, puis tous ceux de l2.

Question 4.4.
Que vont-on affiché si l'on éxecute le programme précédent en remplaçant concatV1 par concatV2?