Listes Chainées 2

Ce TD est dédié aux listes chainées et à l'algorithme trifusion.

Chaque maillon d'une liste est représenté par la structure suivante :

```
typedef structure cellule
{
entier valeur;
LienSur structure cellule suivant;
}Cellule;
```

On pourra dans la suite utiliser les fonctions suivantes sans les implémenter.
- `entier longueur(LienSur Cellule liste)` est une fonction qui renvoie la longueur d'une liste chainée d'entiers.
- `liberer(LienSur Cellule cellule)` libère la place occupée par la cellule `cellule`.

▸ **Exercice 1.** *Écrire une fonction* `Mirroir(transforme LienSur Cellule liste)` *qui inverse l'ordre de tous les éléments de la liste.*

▸ **Exercice 2.** *Écrire une fonction* `Liberer_liste(transforme LienSur Cellule liste)` *qui libère la place de la liste donnée en paramètre.*

Les fonctions suivantes servent à l'implémentation de l'algorithme trifusion qui trie les éléments d'une liste. Le déroulement de l'algorithme est le suivant :
- Si la liste est vide ou ne contient qu'un seul élément alors elle est déjà triée et l'algorithme ne fait rien.
- Sinon la liste est coupée en deux listes que l'on trie grâce à une application récursive de l'algorithme. Ces deux listes sont fusionnées en une nouvelle liste triée.

▸ **Exercice 3.** *Écrire une fonction*
`Ciseaux(transforme LienSur Cellule listea,transforme LienSur Cellule listeb, entier coupure )` *qui coupe la liste* `listea` *en deux listes. La première liste doit être de longueur* `coupure`*. On suppose que* `coupure` *est inférieur à la taille de la liste* `listea` *donnée en entrée.*

▸ **Exercice 4.** *Écrire une fonction*
`Fusion(transforme LienSur Cellule listea,transforme LienSur Cellule listeb)` *qui prend en paramètre deux listes supposées triées et les fusionne en une liste triée dans la liste* `listea`.

▸ **Exercice 5.** *Écrire une fonction récursive* `TriFusion(transforme LienSur Cellule liste)` *qui trie la liste* `liste` *selon l'algorithme de trie Trifusion.*