Listes simplement chaînées

Sujet 02 : Insertion triée dans une liste simplement chaînée

Cette fois-ci nous allons travailler sur des listes simplement chaînées "de base", c'est-à-dire que nous allons écrire les primitives élémentaires permettant d'accèder à cette structure de données. La politique de gestion (i.e. LIFO ou FIFO) vient après. Autrement dit, les fonctions `Empiler` ou `Depiler` peuvent être définies à partir de ces primitives.

```
struct Cellule {
    float valeur;
    struct Cellule *suivant;
}

typedef struct Cellule *Liste;
```

Travail demandé

1. `Liste ConsVide(void)` : Construit une liste vide (`NULL`).
2. `int Vide(Liste l)` : Indique si la liste `l` est vide.
3. `Liste Cons(float tete, Liste queue)` : Construit une nouvelle liste dont la valeur de la première cellule est `tete` et dont le suivant sera `queue`. Il s'agit d'un insertion en tête de liste.
4. `float Tete(Liste l)` : Retourne la valeur en tête de liste (i.e. la valeur qui se trouve dans la première cellule).
5. `Liste Queue(Liste l)` : Retourne la queue de la liste, c'est-à-dire la liste `l` privée de sa première cellule (il s'agit donc du suivant de la première cellule).
6. `void Editer(Liste l)` : Affiche tous les éléments de la liste. Si possible, écrivez cette fonction de manière récursive.
7. `int Taille(Liste l)` : Compte le nombre d'éléments dans la liste (avec la récursivité c'est très simple...).
8. `void Detruire(Liste l)` : Détruit toutes les cellules de la liste (si elle n'est pas vide). Pour cette fonction, je vous conseille vivement d'envisager une solution récursive...
9. `Liste Inverser(Liste l)` : Construit une liste contenant les mêmes valeurs que `l` mais dans l'ordre inverse, i.e. $(1, 2, 3, 4) \rightarrow (4, 3, 2, 1)$.
10. `void Inserer(Liste *l, float valeur)` : Là, nous considérons que la liste `l` passée en paramètre est déjà triée (par construction, i.e. on est parti d'une liste vide et nous n'avons fait que des insertion triées). Cette fonction doit insérer la nouvelle valeur à sa place dans la liste.