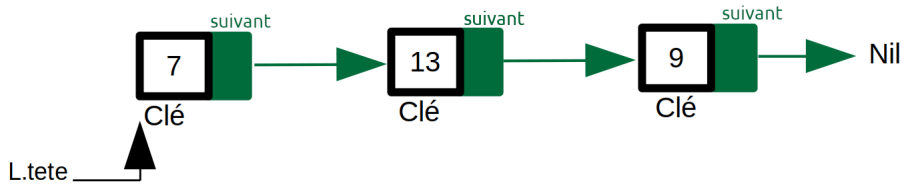
Travaux dirigés #3 – Récursivité et listes chaînées

Sauf mention contraire, les codes sont à donner en `Pseudo-code`.

Les listes chaînées. Une liste chaînée `L` est une suivantession d'éléments contenant chacun une valeur `cle` et une *référence* sur l'élément suivant `suivant`. Les éléments d'une liste chaînée sont appelées des *maillons*. Schématiquement :



Les méthodes suivantes permettent de manipuler ces structures en langage naturel (ou *pseudo-code*). On suppose donc que `ListeChainee` est une classe. Leur implémentation est cachée, le type liste chainée est donc abstrait.

- `ListeChainee()` : créer une liste chaînée vide
- `maillon()` : créer un maillon vide (la clé et le suivantesseur sont `NIL`)
- `L.tete` : premier maillon de la liste `L` (`L.tete = NIL` si la liste est vide).
- `x.cle` : donnée stockée dans le maillon `x`
- `x.suivant` : référence vers le prochain maillon de la liste (ou `NIL` s'il n'y a pas de suivantesseur)

Exercice 1

And the plants and the animals, they are linked.

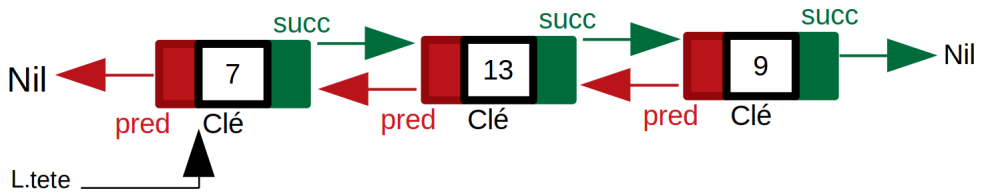
Pour chacune des questions suivantes, il est conseillé de réfléchir au fonctionnement des méthodes sur la liste donnée en exemple avant d'écrire les fonctions. **Pour les trois premières questions, deux versions, itérative et récursive, sont demandées.**

1. Définir une fonction `taille` retournant le nombre d'éléments contenu dans une liste chaînée, passée en paramètre de la fonction.
2. Définir une fonction `ajouter_fin` permettant d'ajouter un élément à la fin d'une liste chaînée, tous deux passés en paramètres de la fonction.
3. Définir une fonction `supprimer_fin` permettant de supprimer le dernier élément d'une liste chaînée, passée en paramètre de la fonction. Si la liste est vide, la fonction ne fait rien.
4. Définir une fonction `copie` retournant la copie d'une liste chaînée passée en paramètre de la fonction.
5. Définir une fonction `concatener` retournant la liste résultant de la concaténation de deux listes chaînées `L1` et `L2`, passées en paramètres de la fonction et supposées non vides. Les listes `L1` et `L2` devront rester inchangées.
6. Définir une fonction `supprimer_element` supprimant la première occurence d'un élément d'une liste chaînée, tous deux passés en paramètres de la fonction. Si l'élément n'est pas présent dans la liste, la fonction ne fait rien.
7. Quel est le nombre d'opérations nécessaires pour ajouter ou supprimer un élément en fin de liste, en fonction de la valeur `taille(L)` ? Comment pourrait-on enrichir la structure de données pour avoir une insertion *instantanée ?*

Exercice 2 — *A link to the past.*

Cet exercice considère la gestion de listes *doublement* chaînées, où chaque maillon contient en plus de la `cle` et du `suivant` une référence sur le maillon précédent `precedent`. Concrètement :

— `x.precedent` : maillon précédent dans la liste (ou `NIL` s'il n'y a pas de prédécesseur au maillon `x`)



1. Définir la fonction `ajouter_fin` permettant d'ajouter un élément à la fin d'une liste doublement chaînée, tous deux passés en paramètres de la fonction.
2. Définir la fonction `supprimer_element` pour une liste doublement chaînée.

Exerice 3 — *This is (not) my last sort.*

1. L'algorithme présenté ci-dessous est destiné à trier un tableau par ordre croissant. Partant de cet algorithme, définissez une fonction `tri_liste` pour trier une liste doublement chaînée dans l'ordre croissant.

Algorithme 1: TriBulle(T)

Entrée: Un tableau d'entiers `T`
Sortie: Le tableau `T` trié dans l'ordre croissant

```
1 pour i de 1 à taille(T) faire
2     pour j de 1 à taille(T)-1 faire
3         si T[j] > T[j + 1] alors
4             échanger T[j] et T[j + 1]
```