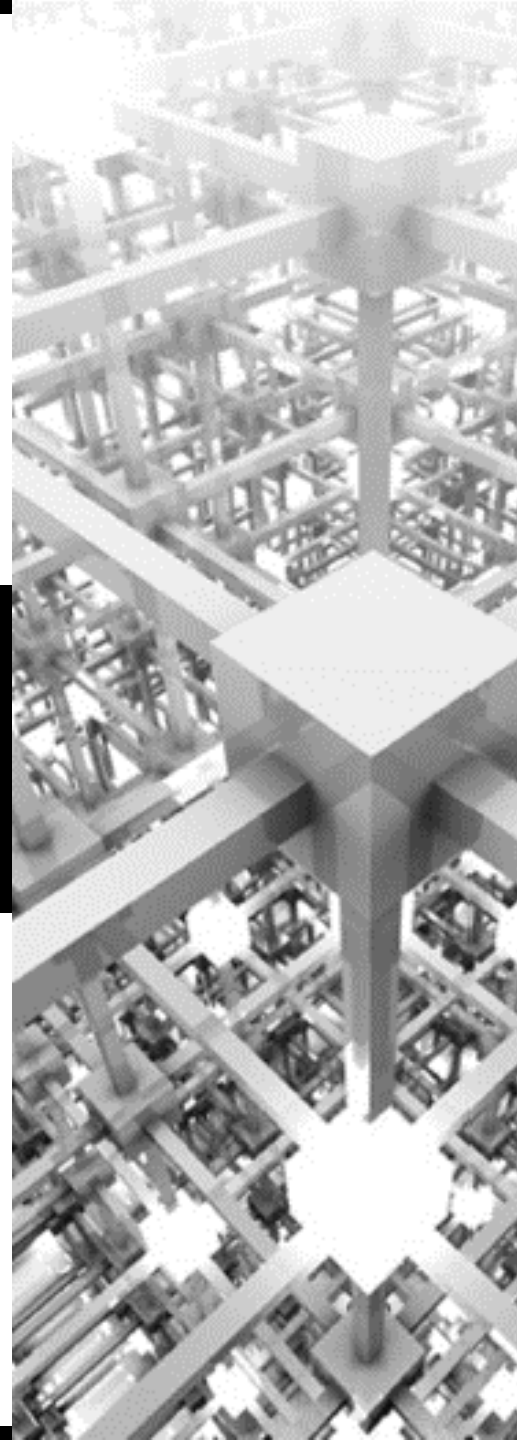
ENSAM

STRUCTURES DE DONNÉES EN C

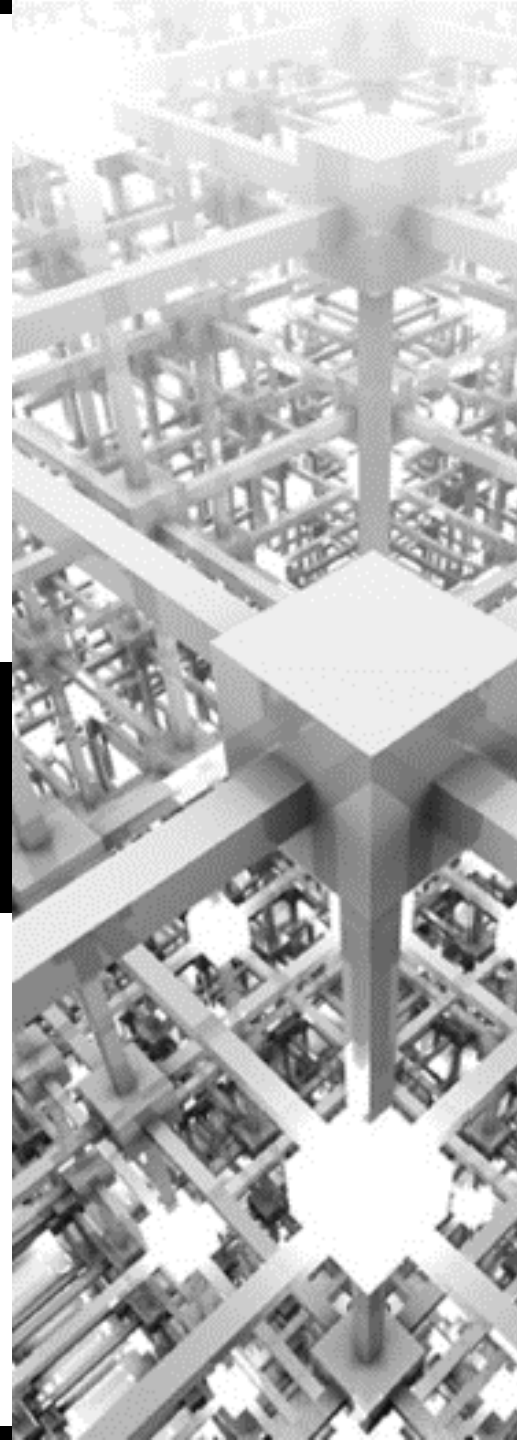
1ère Année «Cycle ingénieur : Intelligence Artificielle et Génie Informatique»

2021/2022

Dep. Informatique
Pf. CHERGUI Adil



ENSAM



Séance 1

LES LISTES CHAINÉES

Objectifs de la séance :

- Étudier le principe et les objectifs des liste chainées
- Explorer les différents types de listes

Pf. Adil CHERGUI

INTRODUCTION

L'objectif des structures de données

L'informatique a révolutionné le monde moderne grâce à sa capacité à traiter de grandes quantités de données, des quantités beaucoup trop grandes pour être traitées à la main. Cependant, pour pouvoir manipuler efficacement des données de grande taille il est en général nécessaire de bien les structurer : **tableaux (dynamiques et statiques), listes, piles, arbres, tas, graphes**... sont des structures qui servent à cette finalité.

Une multitude de structures de données existent, et une multitude de variantes de chaque structure, chaque fois mieux adaptée à un contexte ou algorithme en particulier.

Les tableaux permettent de stocker de grandes quantités de données de même type. Mais, même s'il est possible d'agrandir un tableau une fois plein (ou constaté trop petit), ceci consiste en une opération **coûteuse** (réallocation, copie, etc).
A ce genre de structure, on préfère souvent **les structures dynamiques**, qui grossissent selon les besoins. Ainsi, seule la place nécessaire est réservée.

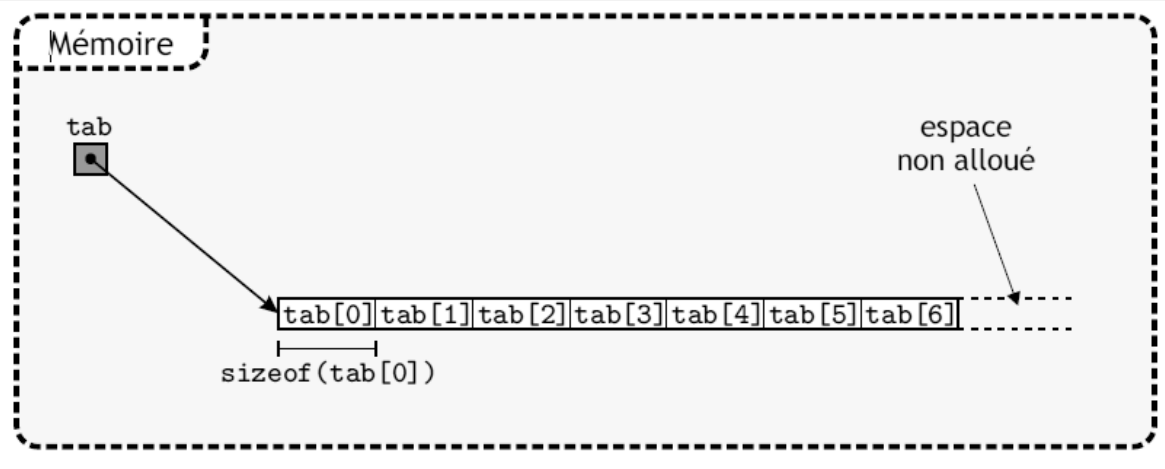
Une application de ce principe qui sera vue en détail dans ce chapitre consacré à la structure de **listes chaînées**.

STRUCTURES DE DONNÉES CLASSIQUES

Les tableaux statiques

Les tableaux représentent la structure de stockage de donnée la plus simple. Ils sont en général implémentes nativement dans la majorité des langages de programmation et sont donc simples à utiliser.

Un tableau représente une zone de mémoire consécutive d'un seul bloc (*cf.* Figure ci-dessous) ce qui présente à la fois des avantages et des inconvénients :



Avantages et inconvénients

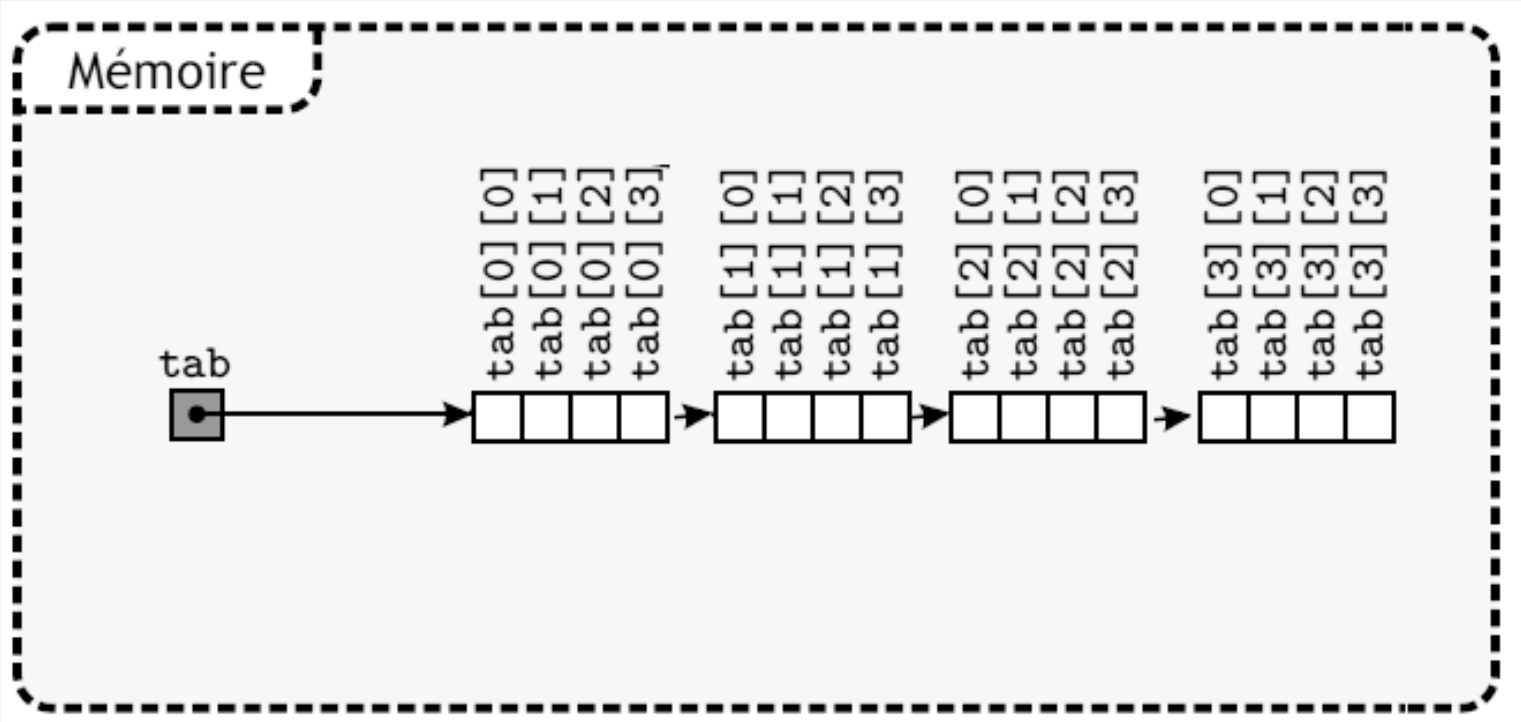
- La mémoire est en un seul bloc consécutif avec des éléments de taille constante à l'intérieur (la taille de chaque élément est défini par le type du tableau), donc il est très facile d'accéder au iéme élément du tableau. L'instruction tab[i] se contente de prendre l'adresse mémoire sur laquelle pointe tab et d'y ajouter i fois la taille d'un élément.
- Il faut fixer la taille du tableau avant de commencer à l'utiliser. Les systèmes d'exploitation modernes gardent une trace des processus auxquels les différentes zones de mémoire appartiennent : si un processus va écrire dans une zone mémoire qui ne lui appartient pas (une zone que le noyau ne lui a pas alloué) il y a une erreur de segmentation.
- La structure de type tableau pose des problèmes pour insérer ou supprimer un élément car ces actions nécessitent des décalages du contenu des cases du tableau qui prennent du temps dans l'exécution d'un programme.

STRUCTURES DE DONNÉES CLASSIQUES

Allocation statique d'un tableau

Il existe deux façons d'allouer de la mémoire à un tableau.

– la plus simple permet de faire de l'allocation ***statique***. Par exemple ***int*** **T[100];** qui va allouer un tableau de 100 entiers pour **T**. De même ***int*** **tab[4][4];** va allouer un tableau à deux dimensions de **taille 4 × 4**. En mémoire ce tableau bidimensionnel peut ressemblera à ce qu'on voit dans la ci-dessus. Ici, les éléments sont stockés ligne à côté de ligne, et il se trouve dans la même zone que toutes les variables de type ***int***. On appelle cela de l'allocation statique car on ne peut pas modifier la taille du tableau en cours d'exécution.



STRUCTURES DE DONNÉES CLASSIQUES

Allocation dynamique d'un tableau

La deuxième façon utilise soit la commande **new** (syntaxe C++), soit la commande **malloc** (syntaxe C) et permet une allocation dynamique (dont la taille dépend des entrées par exemple). L'allocation du tableau s'écrit alors :

int* tab = new int[100];

ou

int* tab = (int*) malloc(100*sizeof(int));.

En revanche cette technique ne permet pas d'allouer directement un tableau à deux dimensions. Il faut pour cela effectuer une boucle qui s'écrit alors comme dans l'exemple de code ci-contre :

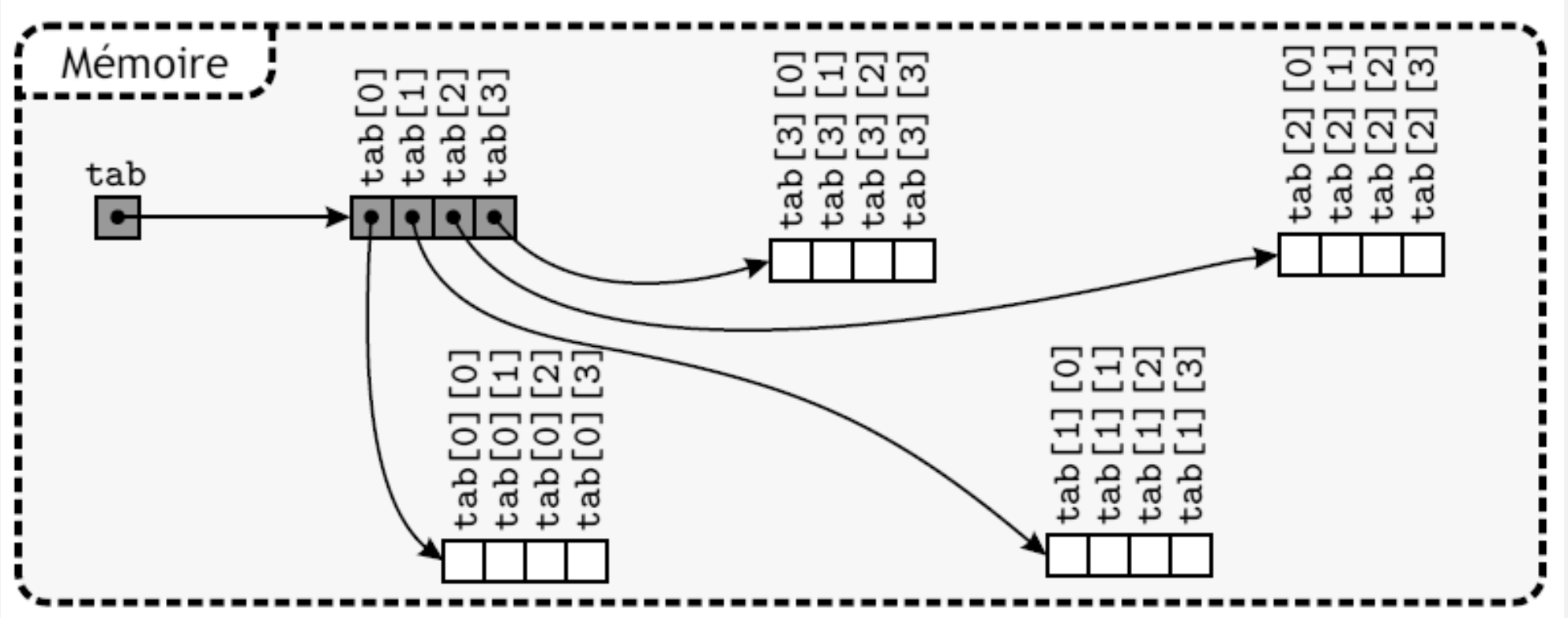
Exemple

```
1  #include <stdio.h>
2  int main()
3  {
4        int i;
5        int** tab2;
6        int** tab3;
7        tab2 = (int**) malloc(4*sizeof(int*));
8        for (i=0; i<4; i++)
9        {
10             tab2[i] = (int*) malloc(4*sizeof(int));
11       }
12       /* ou en utilisant new */
13       tab3 = new int*[4];
14       for (i=0; i<4; i++) {
15       tab3[i] = new int[4];
16       }
17     return 0;
18 }
```

STRUCTURES DE DONNÉES CLASSIQUES

Allocation dynamique d'un tableau

Attention, un tableau alloué statiquement ne se trouve pas dans la même zone mémoire qu'un tableau alloué avec l'une des méthodes d'allocation dynamique :



LES LISTES CHAINÉES

Définitions

Ce type classique de stockage de valeurs peut donc être coûteux en temps d'exécution.

Il existe une autre structure, appelée **liste chaînée**, pour stocker des valeurs, elle peut implémenter pratiquement n'importe quoi, par exemple,

- une liste d'entiers [3; 2; 4; 2; 5],
- une liste de courses [pommes; beurre; pain; fromage],
- ou une liste composée de plusieurs éléments tel qu'une pages Web contenant chacune une image et un lien vers la page Web suivante.

Ce type de structure permet plus aisément **d'insérer** et de **supprimer** des valeurs dans une liste linéaire d'éléments.

Une **liste chaînée** est une **structure linéaire** qui n'a pas de **dimension fixée** à sa création. Ses éléments de même type sont éparpillés dans la mémoire et reliés entre eux par des pointeurs.

Sa **dimension** peut être modifiée selon la **place disponible en mémoire**.

La liste est accessible **uniquement** par sa tête de liste c'est-à-dire son premier élément.

LES LISTES LINÉAIRES

Définitions

Une **liste linéaire** est une structure de données correspondant à **une suite d'éléments**. Les éléments **ne** sont **pas indexés** dans la liste, mais pour chaque élément (sauf le dernier) on peut **accéder** à **l'élément suivant**. Par conséquent, on ne peut accéder à un élément qu'en passant par **le premier élément** de la liste et en parcourant tous les éléments jusqu'à ce qu'on atteigne l'élément recherché. Ce type de structure permet plus aisément d'insérer et de supprimer des valeurs dans une liste linéaire d'éléments.

Les différents types de liste chaînée

Il existe différents types de listes chaînées :

- **Liste chaînée simple** constituée d'éléments reliés entre eux par des pointeurs.
- **Liste chaînée ordonnée** où l'élément suivant est plus grand que le précédent. L'insertion et la suppression d'élément se font de façon à ce que la liste reste triée.
- **Liste doublement chaînée** où chaque élément dispose non plus d'un mais de deux pointeurs pointant respectivement sur l'élément précédent et l'élément suivant. Ceci permet de lire la liste dans les deux sens, du premier vers le dernier élément ou inversement.
- **Liste circulaire** où le dernier élément pointe sur le premier élément de la liste. S'il s'agit d'une liste doublement chaînée alors de premier élément pointe également sur le dernier.

LES LISTES SIMPLEMENT CHAINÉES

Eléments utilisées dans les listes

Une liste simple est une structure de données telle que chaque élément contient :

- des informations caractéristiques de l'application (les caractéristiques d'une personne par exemple),
- un pointeur vers un autre élément ou une marque (NULL) de fin s'il n'y a pas d'élément successeur.

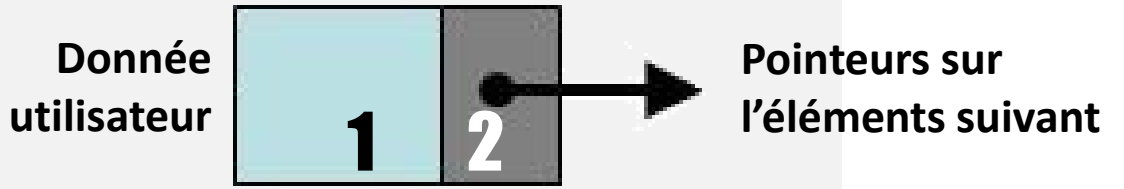
Chaque élément est créé à partir d'un **structure prototype** qui ce fait nommé dans communauté par plusieurs noms : **Maillon, Cellule, Element, Nœud, Case**, …

Nous allons choisir dans ce cours les nominations suivantes :

```
typedef struct cell
{
        int val;
        struct cell *next;
}cellule;
```

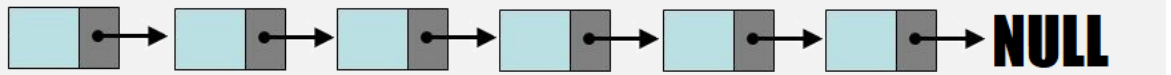
Représentation graphique

Pour bien comprendre les manipulations des listes chaînées, il est très intéressant de les représentées graphiquement. La forme d'une cellule est composée de deux rectangles, le rectangle 1 représente le contenu de la cellule, le rectangle 2 représente le lien vers la cellule suivante.



représentation d'un élément d'une liste simplement chaînée.

Ainsi, les étapes de composition et de manipulation des listes chaînées se font assimiler de manière graphique avant d'implémenter leurs codes.



LES LISTES SIMPLEMENT CHAINÉES

Fonctions attendus

Au vu de l'utilisation des listes chaînées, il se dessine clairement quelques fonctions indispensables ::

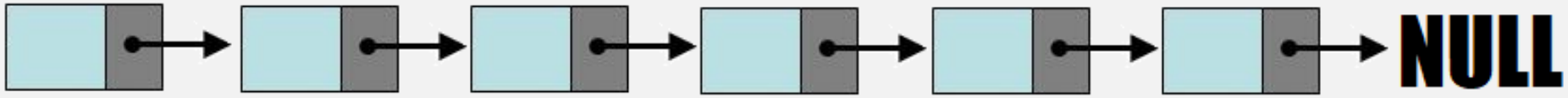
- Initialisation
- Ajout d’un élément
- Suppression d’un élément
- Accès à l’élément suivant
- Accès aux données utilisateur
- Accès aux premiers éléments de la liste
- Calcul de la taille de la liste
- Suppression de la liste entière

TYPES DE LISTES CHAINÉES

Types de liste

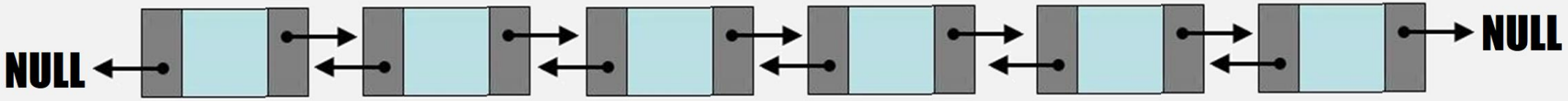
Liste simplement chainnée

La liste chainée simple permet de circuler que dans un seul sens, c'est ce modèle :



Liste symétrique ou doublement chainée

Avec le modèle double chaque élément possède l'adresse du suivant et du précédent ou des marques de fin s'il n'y en a pas. Il est alors possible de parcourir la chaine dans les deux sens :

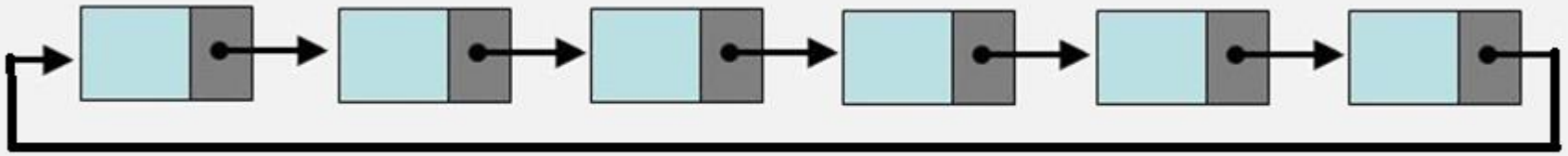


TYPES DE LISTES CHAINÉES

Types de liste

Liste circulaire simple

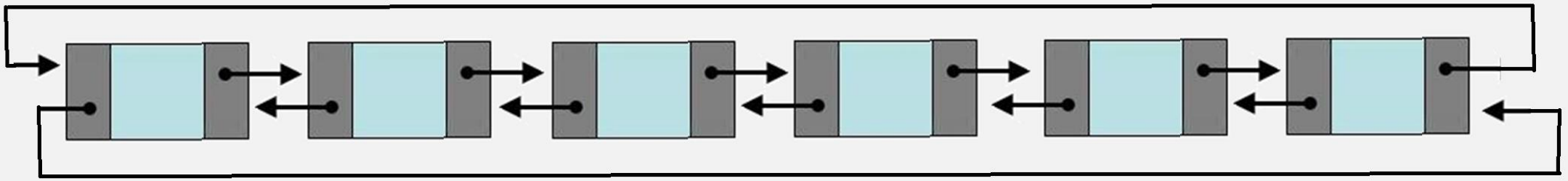
Dans une liste circulaire simple le dernier prend l'adresse du premier et la circulation est prévue dans un seul sens :



Dans ce modèle premier et dernier n'ont plus la même importance, le premier peut être n'importe quel maillon de la chaine et le dernier celui qui le précède.

Liste circulaire double

Même principe que précédemment mais avec une circulation possible dans les deux sens :



Comparaisons: Listes chainées contre tableaux

Tableaux	Liste Chainées
Taille en mémoire : Par nature, la tableau a une taille définie même dans le cas d'un tableau dynamique dont la taille peut être réévaluée périodiquement. La taille du tableau fait partie de la définition du tableau.	La liste chainée dynamique n'a pas pour sa définition de nombre d'éléments. Ils sont ajoutés ou soustraits à la demande, pendant le fonctionnement du programme.
Soustraire un élément : impossible dans un tableau d'enlever une case du tableau. Il est éventuellement possible de masquer un élément mais pas de retirer son emplacement mémoire.	Aucun problème pour soustraire un élément de la liste et libérer la mémoire correspondante.
Accès élément : Le tableau permet d'accéder à chaque élément directement. C'est très rapide.	Pour accéder à un élément il faut parcourir la liste jusqu'à lui, ça peut être long.
Tris : Les tris de tableau ne nécessitent pas de reconstruire les liens entre les emplacements mémoire du tableau. Il y a juste à manipuler les valeurs afin de les avoir dans l'ordre voulu	Il ne suffit pas de manipuler les valeurs il faut aussi reconstruire la chaine. Le mieux est de construire sa chaine en mettant les éléments dans l'ordre dés le départ plutôt que d'avoir à réorganiser l'ordre des éléments dans la chaine. Trier une chaine revient à construire une autre chaine en insérant dans l'ordre voulu chaque élément.