جامعة ابن طفيل
UNIVERSITÉ IBN TOFAÏL

Faculté des Sciences
Département d'Informatique

Filière : Intelligence Artificielle

SMI (S3)

Partie 2 : Implémentation des Types de Données Abstraits en C

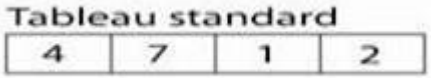
---

1 Les listes chainées (*Linked List*)

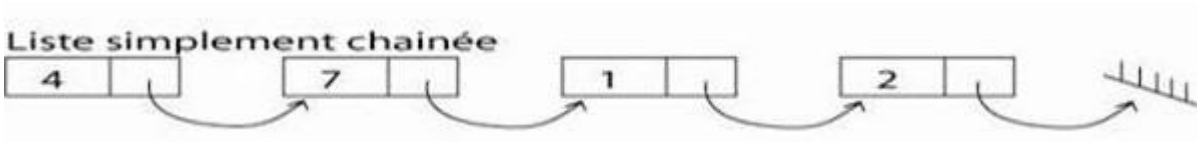
Introduction

- Au domaine de la mécanique, on parle d'une chaîne :

- En informatique, on parle de données enchaînées :
  - sous forme d'un tableau :



  - ou bien d'une structure dynamique qu'on appelle liste chaînée :



Plan de première partie

I.Types des listes chaînées

I.1. C'est quoi une liste chaînée ?

I.2. Types des listes chaînées.

II. Définition et représentation des **listes dynamiques simples**

II.1. C'est quoi une liste dynamique simple ?

II.3. C'est quoi le maillon d'une liste ?

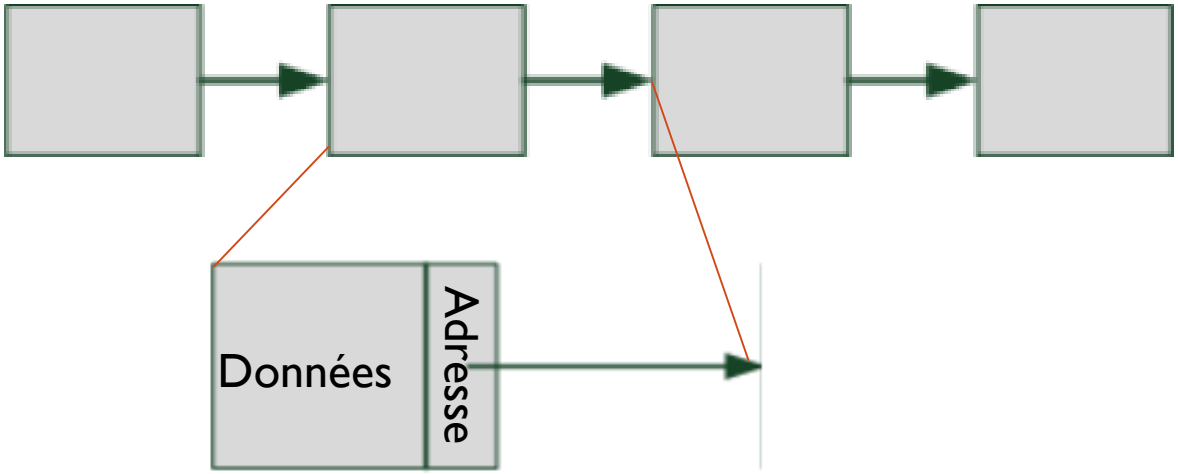
II.3. Comment construire un maillon ?

II.4. Comment représenter une liste?

II,5, Comment gérer une liste ?

II.6. Structure du contrôle d'une liste

C’est quoi une liste chainée ?



- Une liste chainée est une liste d’entités (éléments) de même type constitué de :
  - **Données** relatives à cette entité.
  - **L'adresse** de l’élément suivant ou une marque de fin s'il n'y a pas de suivant.(**Chainage**)

Types des listes chaînées

- Selon l'adresse de l'élément suivant, on distingue deux types de liste :

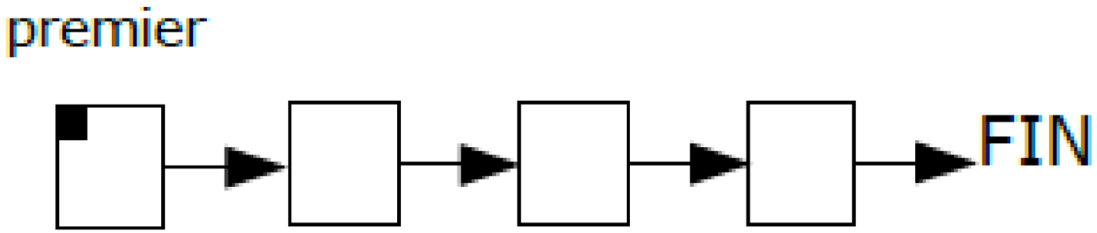
1. Liste dynamique (allocation dynamique) : l'**adresse** de l'élément suivant est récupérée par un pointeur sur l'adresse d'une zone mémoire allouée.
2. Liste statique (allocation statique) : l'adresse dans ce cas est
   a) Un **indice de tableau** récupéré avec un entier,
   b) Ou bien une **position dans un fichier** : c'est un indice qui vaut le numéro d'ordre de l'objet dans le fichier multiplier par la taille en octet du type de l'objet. Elle est récupérée avec un entier.

Types des listes chaînées

On doit parler d'un autre typage que statique/dynamique, celui lié à la nature de chainage entre éléments de la liste.

- *Liste simple (chainage simple) :*

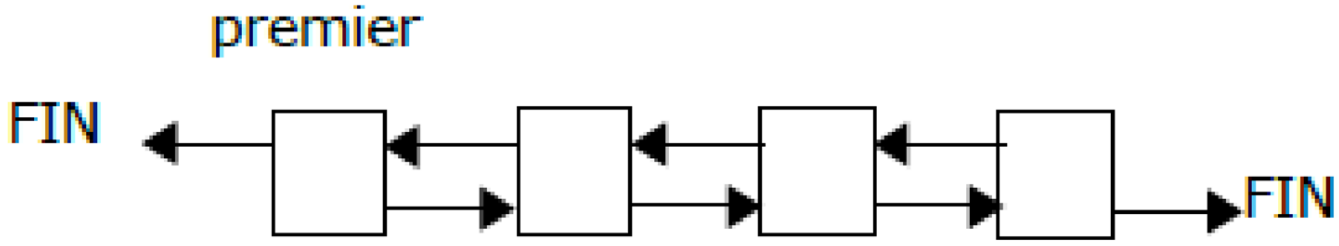
Une liste chaînée simple permet de parcourir les données dans un seul sens (de début jusqu'à la fin de la liste). Elle est représenté par le modèle simple suivant :



Types des listes chainées

- *Liste symétrique ou doublement chainée (chainage double) :*

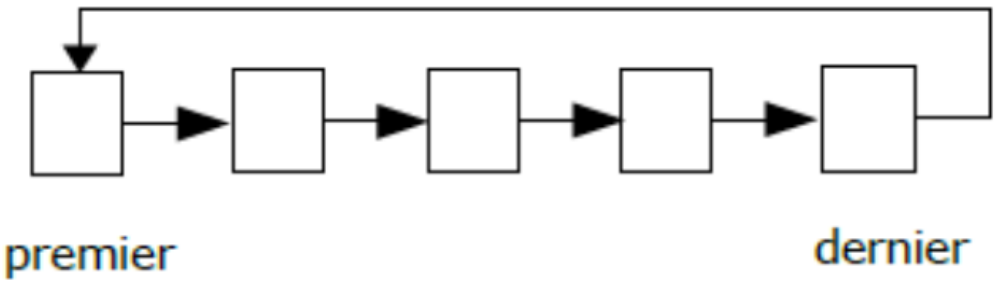
Avec le modèle double, figure ci-dessous, chaque élément possède l'adresse du suivant et du précédent ou des marques de fin s'il n'y en a pas. Il est alors possible de parcourir la chaine dans les deux sens :



Types des listes chainées

- *Liste circulaire simple (chainage circulaire simple) :*

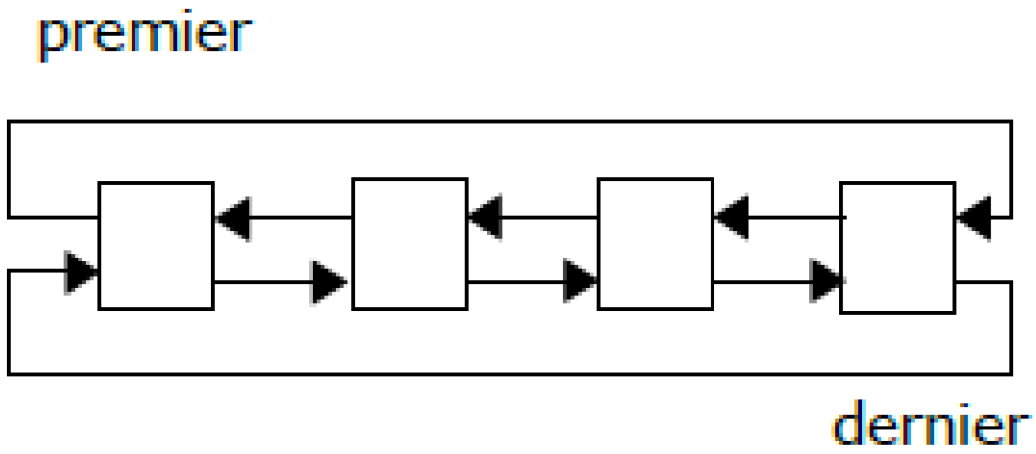
Dans une liste circulaire simple, le dernier élément prend l'adresse du premier, et la circulation est prévue dans un seul sens :



Types des listes chainées

- *Liste circulaire double (chainage circulaire double) :*

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



Introduction (pourquoi une liste chainée?)

I.Types des listes chaînées

I.1. C'est quoi une liste chainée ?

I.2. Types des listes chaînées.

II. Définition et représentation des listes dynamiques simples

II.1. C'est quoi une liste dynamique simple ?

II.3. C'est quoi le maillon d'une liste ?

II.3. Comment construire un maillon ?

II.4. Comment représenter une liste?

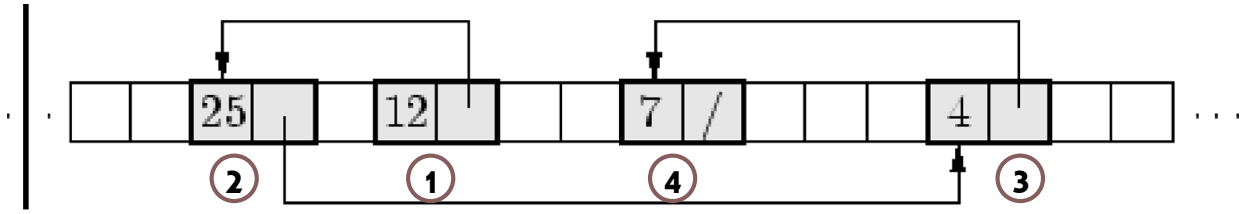
II.5. Comment gérer une liste ?

II.6. Structure du contrôle d'une liste

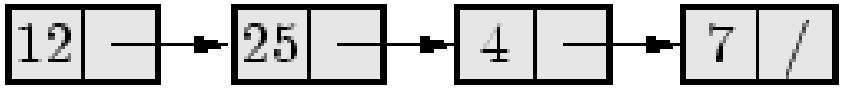
C'est quoi une liste dynamique simple ?

- Une *liste chaînée dynamique simple* est une structure de données dans laquelle les éléments sont alloués dynamiquement et rangés linéairement :

- alloués dynamiquement : les éléments sont alloués un après l'autre, à la différence du tableau, ils n'ont aucune raison d'êtres contigus en mémoire. Les éléments sont éparpillés dans la mémoire.



- rangés linéairement : Chaque élément est lié à son successeur (suivant) et il n'est donc pas possible d'accéder directement à un élément quelconque de la liste.



C’est quoi le maillon d’une liste ?

- Définition : Un maillon (nœud) étant une structure qui contient des informations d’un élément à stocker et un pointeur sur le prochain maillon de la liste. Alors qu’une liste est composée des instances d’un maillon de même type.

- Donc un maillon est l'élément de base d'une liste chaînée constitué de:
  - **champs de données,**
  - **et d'un pointeur vers un maillon suivant.**

- Maillon suivant : Le champ *pointeur vers un maillon* pointe vers le maillon suivant de la liste. S'il n'y a pas de maillon suivant, le pointeur vaut NULL (la fin de la liste).

Comment construire un maillon ?

- Pour définir un maillon de la liste le mot clé *struct* sera utilisé.
- L'élément de la liste contiendra ou moins un champ donnee (de n'importe quel type) et un pointeur « suivant » de type maillon.
- Le pointeur «suivant» doit être du **même type** que le maillon, sinon il ne pourra pas pointer vers l'élément suivant de la liste. Voilà un exemple :

```
typedef struct ElementListe {
        char donnee;
        struct ElementListe *suivant;
} Element;
```

Comment représenter une liste ?

- Une façon simple de représenter une liste, consiste à se dire qu'une liste soit :
  - une liste **vide** (un pointeur qui pointe NULL),
  - ou bien constituée d'une **tête** (qui est donc la valeur du premier élément de la liste) et d'une **queue** (qui est le reste de la liste).
- Pour accéder à un élément, la liste est parcourue en commençant avec la tête, le pointeur suivant permettra le déplacement vers le prochain élément.
- Le déplacement se fait dans une seule direction, du premier vers le dernier élément.

Comment gérer une liste ?

- 1ère méthode : méthode simple

On déclare un pointeur de type maillon qui va pointer sur le début de la liste, ou NULL si la liste est encore vide.

Pointeur sur début de la liste

- 2ème méthode : méthode de contrôle

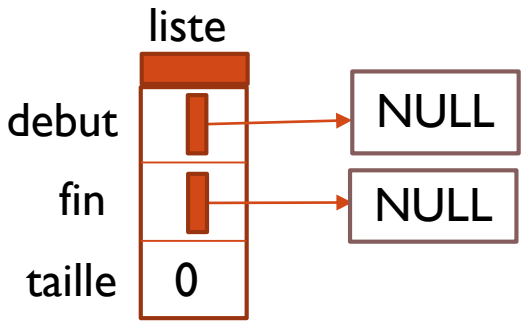
Pour avoir le contrôle d'une liste on en défini une structure pour sauvegarder les éléments suivants :

- Un pointeur debut (tête) : contiendra l'adresse du premier élément de la liste, ou NULL si la liste est vide.
- Un pointeur fin (queue) : contiendra l'adresse du dernier élément de la liste ou NULL si la liste est vide.
- Une variable *taille* (nombre d'éléments) : contient le nombre actuel d'éléments.

Structure du contrôle d'une liste

- Pour contrôler une liste on définit une structure dans laquelle on stocke les 3 éléments du contrôle comme suivant :

```
typedef struct ListeRepere {
        typeDeMaillon *debut;
        typeDeMaillon *fin;
        int taille;
} Liste;
```



- typeDeMaillon est le type des éléments à gérer par cette structure.
- Quelque soit la position dans la liste, les pointeurs debut et fin pointent toujours respectivement vers le 1er et le dernier élément si la liste n'est pas vide.
- La variable *taille* contiendra le nombre actuel d'éléments de la liste quelque soit l'opération effectuée sur la liste.

Exercice d'application n°1

1. Construire un maillon (élément de base) d'une liste chaînée contenant les données d'un produit : code, nom, prix, quantite.
2. Définir une structure du contrôle de cette liste.
3. Déclarer une liste (variable structurée) avec initialisation à NULL de ses éléments (debut=NULL, fin=NULL, taille=0).

Solution d'Exercice d'application n°1

Structure du maillon produit :

```
Typedef struct elementListe{
        int code;
        char nom[20];
        float prix;
        int quantite;
         struct elementListe *suivant;
} produit;
```

Solution d'Exercice d'application n°1

Structure du contrôle de liste

```
Typedef struct {
        produit *debut;
        produit * fin;
        int taille;
} MaListe;
```

Déclaration avec initialisation

```
MaListe liste={NULL,NULL,0};
```

Opérations sur une liste simplement chainée

- La liste chaînée de caractères utilisée sera vide au départ.
- L'exemple suivant défini le type abstrait **Noeud** permettant de créer le type **Liste** représentant une liste chaînée de **caractères** (**char**).

```
typedef char Type;
```

```
typedef struct Noeud * Liste;
```

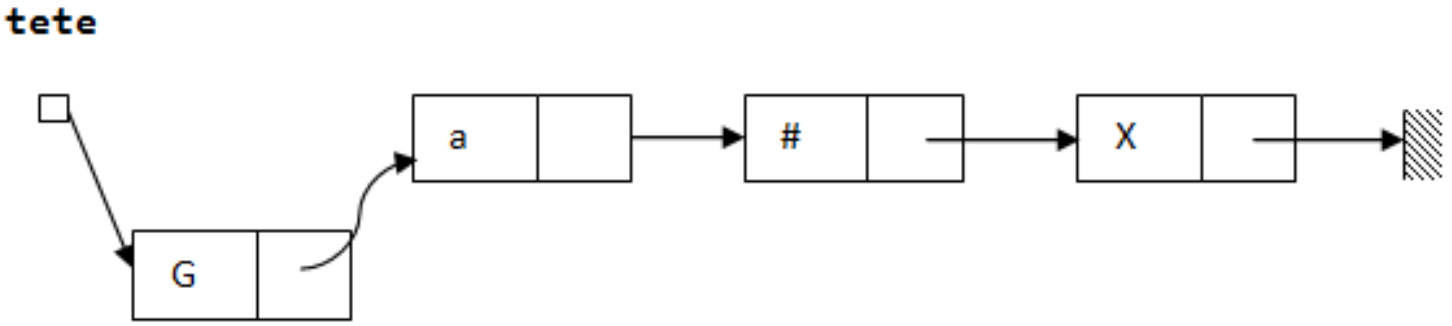
```
typedef struct Noeud{
    Type info;
    Liste suivant;
}Noeud;
```

Insertion d'un élément

- Pour insérer un élément dans une liste chaînée, il faut savoir où l'insérer.
- Les trois insertions possibles dans une liste chaînée sont :
  - en tête de liste
  - en fin de liste
  - à une position donnée

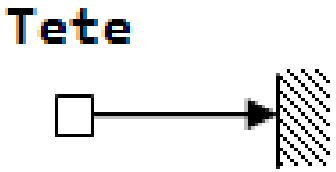
Insertion de nœuds en tête de liste chaînée

- L'insertion en tête, se fait en créant un élément, lui assigner la valeur que l'on veut insérer, puis pour terminer, raccorder cet élément à la liste.
- Lors d'une insertion en tête, on devra donc assigner au **suivant** l'adresse du premier élément de la liste.



Insertion de nœuds en tête de liste chaînée

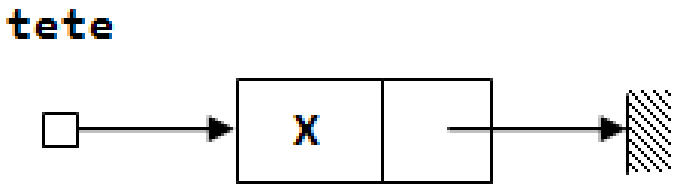
- Illustrons un chaînage en tête : au départ, la liste chaînée est vide, et on suppose que l'on saisit, dans l'ordre, les caractères 'X', '#' et 'a'.
  - Initialisation de la liste à **NULL** :



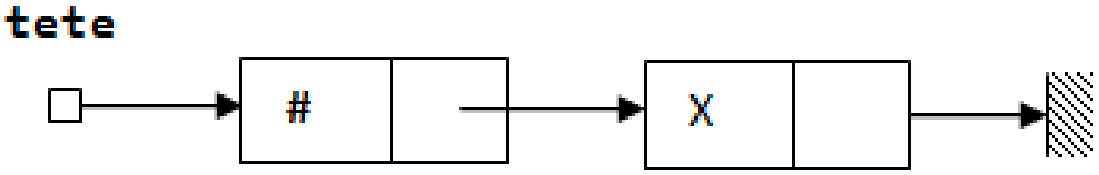

```
// ...
main(){
    Liste tete;
    tete = NULL;
}
```

Insertion de nœuds en tête de liste chaînée

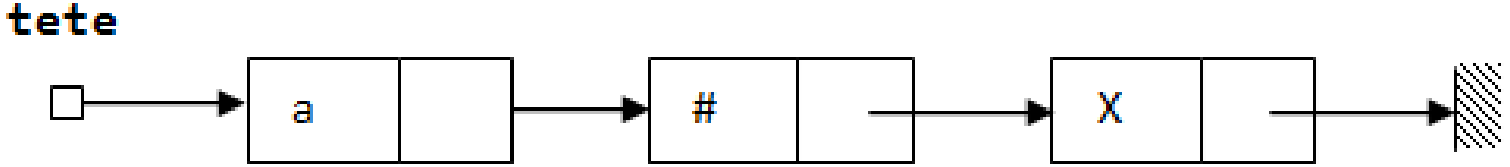
- Ajout de 'X' :



- Ajout de '#' :



- Ajout de 'a' :



Insertion de nœuds en tête de liste chaînée

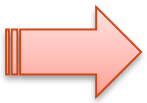
- Le programme suivant implémente les différentes étapes illustrées :

```
// ...
main(){
    // ...
    // Création du noeud contenant 'X'
    Noeud * nouveau = (Noeud*)malloc(sizeof(Noeud));
    nouveau->info = 'X';
    nouveau->suivant = tete;
    tete = nouveau;
    // Création du noeud contenant '#'
    nouveau = (Noeud*)malloc(sizeof(Noeud));
    nouveau->info = '#';
    nouveau->suivant = tete;
    tete = nouveau;
    // Création du noeud contenant 'a'
    nouveau = (Noeud*)malloc(sizeof(Noeud));
    nouveau->info = 'a';
    nouveau->suivant = tete;
    tete = nouveau;
}
```

Insertion de nœuds en tête de liste chaînée

- Pour afficher le contenu de la liste, on doit parcourir la liste avec un pointeur, pour ne pas perdre la tête de la liste, jusqu'au bout et afficher toutes les valeurs qu'elle contient :

```
// ...
main(){
    // ...
    Liste tmp = tete;
    while(tmp != NULL){
        printf("%c ", tmp->info);
        tmp = tmp->suivant;
    }
}
```



```
a # X
```