

Chapitre III

Structure de données Les listes chaînées

1. Introduction : Objectif et Motivation

- Rappelons qu'un programme informatique est un ensemble d'instructions et de données qui permettent à l'ordinateur de traiter des informations.
- Ces informations sont organisées dans la mémoire principale et dans les mémoires auxiliaires.
- Par ailleurs, notons que l'ordinateur définit au niveau matériel :
 - Un jeu d'instructions élémentaires (additionner, soustraire, multiplier...),
 - Quelques représentations de données élémentaires (bits).
- Mais, il faut signaler que le matériel ne supporte que des données élémentaires, donc non structurées.
- On peut ainsi définir une donnée structurée comme étant une collection organisée de données élémentaires.
- En informatique, on appelle une telle collection, une *structure de données*.
- Une *structure de données* est donc une manière d'organiser et de représenter des informations en mémoire centrale et dans les mémoires auxiliaires.
- Une structure de données contient deux types de renseignement :
 - les informations à stocker,
 - les liens entre ces informations.
- On peut distinguer deux types de liens :
 - Liens implicites : l'adresse de l'information est calculée en fonction de l'emplacement physique de l'information en mémoire.
 - liens explicites : les pointeurs.

1. Introduction : Objectif et Motivation

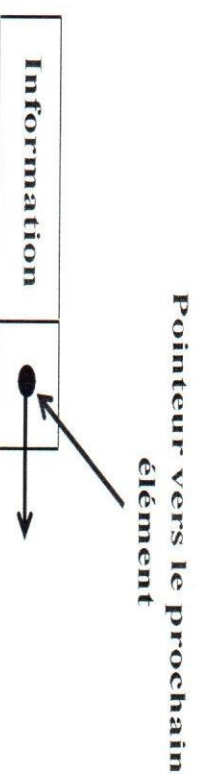
- En langage C, il existe un grand nombre de structure de données :
Exemple : les tableaux les structures, les listes chaînées, les piles, les files, les arbres binaires...
- Le choix d'une structure dépend, en général, des opérations à réaliser sur les informations. Certaines structures sont plus adaptés que d'autres à certaines opérations.
- *L'objectif* est la gestion d'un ensemble fini d'éléments dont le nombre n'est pas fixé.
- Les éléments de cet ensemble peuvent être de différentes types : nombres entiers ou flottants, chaînes de caractères,...
- On ne s'intéressera pas aux éléments de l'ensemble en question mais aux opérations que l'on effectue sur cet ensemble, indépendamment de la nature de ses éléments.
- En programmation informatique, ces ensembles sont considérés comme des objets dynamiques.
- En effet, le nombre de leurs éléments varie au cours de l'exécution du programme, puisqu'on peut y ajouter et supprimer des éléments en cours de traitement.
- Plus précisément les opérations que l'on s'autorise sur les ensembles sont les suivantes :
 - tester si un ensemble est vide.
 - ajouter un élément à un ensemble.
 - supprimer un élément à un ensemble.
 - rechercher un élément dans un ensemble.
- Cette gestion des ensembles doit, pour être efficace, répondre à deux critères :
 - un minimum de place mémoire utilisée,
 - et un minimum d'instructions élémentaires pour réaliser une opération.

1. Introduction : Objectif et Motivation

- Notons que la structure de données la plus utilisée pour manipuler des données sont les tableaux.
- Mais, dans certains cas, les tableaux ne constituent pas la meilleure solution pour stocker et manipuler les données.
- En effet, Si on veut supprimer un élément au milieu d'un tableau, il faut recopier les éléments temporairement, réallouer le tableau, puis le remplir à partir de l'élément supprimé.
- Ces manipulations sont coûteuses en ressources.
- En langage C, les *listes chaînées* (ou tout simplement les listes) peuvent être considérées comme des structures de données très pratiques pour manipuler des éléments d'un ensemble d'objets dynamiques.
- En effet, une liste chaînée est différente par rapport à un tableau dans le sens où ses éléments sont répartis dans la mémoire et reliés entre eux par des pointeurs.
- Ceci permet d'ajouter et de supprimer des éléments d'une liste à n'importe quel endroit, sans modifier la liste entière.

2. Les listes chaînées

- Une *liste chaînée* est une structure de base de la programmation informatique.
- C'est une structure de données dans laquelle les objets sont stockés de manière ordonnée (l'ordre signifie que chaque objet possède une position dans la liste).
- Cependant, contrairement au tableau, pour lequel l'ordre est déterminé par les indices, l'ordre d'une liste chaînée est déterminé par un pointeur dans chaque objet.
- Les listes chaînées sont des structures dynamiques capables,
 - d'accueillir un nombre infini d'éléments,
 - et de supporter toutes les opérations sur les ensembles dynamiques (insertion, suppression, recherche,...).
- Dans la suite du chapitre, on utilise la liste chaînée pour représenter un ensemble d'éléments.
- Chaque élément est contenu dans une *cellule* (appelée aussi *maillon*), et chaque cellule est composée d'une partie contenant l'information et d'au moins un pointeur.



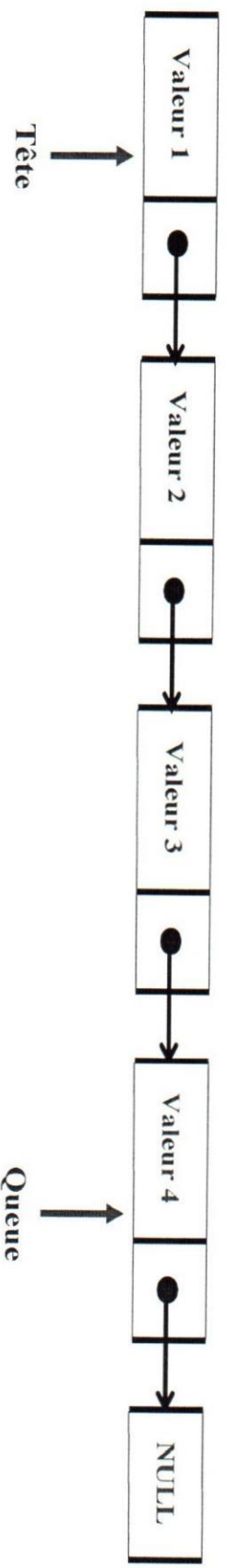
Représentation d'une cellule avec un seul pointeur.

2. Les listes chaînées

- Chaque élément est accessible par celui qui le précède grâce au pointeur.
- Une liste chaînée est donc une succession de cellules, dont le dernier porte une adresse de fin (le pointeur NULL). La première cellule de la liste est appelée tête et la dernière cellule est appelée queue.
- L'accès à la liste chaînée se fait par un pointeur vers le premier élément.
- Pour atteindre un des éléments de la liste,
on est obligé de parcourir la liste depuis le début.
A partir d'un élément quelconque,
on ne peut atteindre que le suivant.
- Une liste chaînée peut prendre différentes formes :
 - Elle peut être *simple* (dite aussi *simplement chaînée*) : chaque cellule est composé de l'information et d'un seul pointeur (pointe sur l'adresse de la cellule suivante).
 - Ou *doublement chaînée* : chaque cellule est composé de l'information et de deux pointeurs (l'un pointe sur l'adresse de la cellule suivante et l'autre sur celle de la cellule précédente).
 - Ou *circulaire* : la tête pointe sur la queue de la liste, et la queue pointe sur la tête de la liste.

2. Les listes chaînées : 2.1 Les listes simplement chaînées

- Rappelons qu'une liste simplement chaînée est une liste où chaque cellule contient des informations accessibles et modifiables par l'utilisateur ainsi qu'un pointeur vers la cellule suivante.



- Comme les listes simplement chaînées sont une structure de données importante en programmation informatique,
- il est fondamental de s'intéresser à la manière de les implanter en vue de leurs manipulations informatiques.
- Deux modes d'implantation peuvent être envisagés :
 - Implantation par pointeur.
 - Implantation par tableau.

2.1 Les listes simplement chaînées : Implantation par pointeur

Déclaration

- Le langage C permet de réaliser des listes chaînées à l'aide des structures.
- En effet, les cellules sont des objets constitués :
 - d'un champ de données,
 - d'un pointeur vers la cellule suivante.
- La déclaration correspondante est la suivante, où l'on suppose ici que les valeurs stockées dans chaque cellule sont de type entier.

Code en C

```
typedef int element;  
struct cellule  
{  
    element valeur;  
    struct cellule * suivant;  
};  
typedef struct cellule cellule, * liste;
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Remarque 1:

Notons qu'on peut créer des listes chaînées contenant des valeurs qui sont :

- de différents type, c-à-d des entiers, des flottants, des caractères,
- des tableaux, des structures,
- une combinaison de plusieurs types,
- ou des listes chaînées.

Exemple :

```
typedef double element;  
struct cellule  
{  
    element valeur;  
    struct cellule * suivant;  
};  
typedef struct cellule cellule, * liste;
```

Remarque 2:

Les instructions suivantes permettent de déclarer, de façons différentes mais équivalentes, une liste chaînée :

```
struct cellule *l;  
cellule *l;  
liste l;
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Création et initialisation

- Pour créer une liste chaînée, on doit l'initialiser à NULL (macro de stdlib.h), cela va nous permettre d'allouer la première cellule.
- L'initialisation d'une liste chaînée, ou la création d'une liste vide, est généralement faite par la fonction suivante :

```
liste creerListeVide( )  
{  
    return NULL;  
}
```

- Pour tester si une liste est vide, on utilise la fonction suivante :

```
int ListeVide(liste l)  
{  
    return l==NULL;  
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Insertion d'une cellule (Ajout d'une valeur dans une liste) :

- Il est toujours possible d'insérer des cellules (ajouter des valeurs) à une liste chaînée donnée.
- En effet, il y a plusieurs façons d'ajouter des cellules dans des listes chaînées :
 - ajouter une nouvelle cellule en tête de la liste,
 - ajouter une nouvelle cellule à la fin de la liste.
 - ajouter une nouvelle cellule à un emplacement quelconque dans la liste.
- Exemple : Les fonctions `ajouterElementTeteListe()` et `ajouterElementFinListe()` permettent respectivement l'ajout d'un élément en tête et à la fin d'une liste.

Ajout d'un élément en tête de la liste

```
liste ajouterElementTeteListe(element val, liste l)
{
    liste ln;
    ln=malloc(sizeof(cellule));
    ln->valeur=val;
    ln->suitant=l;
    return ln;
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Insertion d'une cellule (Ajout d'une valeur dans une liste) :

Ajout d'un élément à la fin de la liste

```
liste ajouterElementFinListe(element val, liste l)
{
    liste ln, lp;
    ln=malloc(sizeof(cellule));
    ln->valeur=val;
    ln->successeur=NULL;
    if(l==NULL)
        return ln;
    lp=l;
    while(lp->successeur != NULL)
        lp= lp->successeur;
    lp->successeur =ln;
    return l;
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Suppression d'une cellule (supprimer une valeur dans une liste) :

- Il est possible également de supprimer des cellules (des valeurs) dans une liste chaînée donnée.
- En effet, il y a plusieurs façons de supprimer des cellules dans des listes chaînées :
 - supprimer une cellule en tête de la liste,
 - supprimer une cellule à la fin de la liste,
 - supprimer une cellule quelconque dans une liste.

Suppression d'un élément en tête de la liste

```
liste supprimerElementTeteListe(liste l)
{
    liste lp;
    if (l==NULL)
        return NULL;
    lp=l->suivant;
    free(l);
    return lp;
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Suppression d'une cellule (supprimer une valeur dans une liste) :

Suppression d'un élément à la fin de la liste

```
liste supprimerElementFinListe(liste l)
{
    liste lp1, lp2;
    if (l == NULL)
        return NULL;
    if (l->suitant == NULL)
    {
        free(l);
        return NULL;
    }
    lp1 = l;
    while (lp1->suitant->suitant != NULL)
        lp1 = lp1->suitant;
    lp2 = lp1->suitant;
    lp1->suitant = NULL;
    free(lp2);
    return l;
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Suppression d'une cellule (supprimer une valeur dans une liste) :

Suppression d'un élément quelconque dans la liste (Méthode itérative)

```

liste supprimerIterativementElementListe(element val, liste l)
{
    liste lp1, lp2;
    if (l != NULL)
        if (l->valeur == val)
        {
            lp1 = l;
            l = l->successeur;
            free(lp1);
        }
        else
        {
            lp2 = l;
            while (lp2->successeur != NULL && lp2->successeur->valeur != val)
                lp2 = lp2->successeur;
            if (lp2->successeur != NULL)
            {
                lp1 = lp2->successeur;
                lp2->successeur = lp2->successeur->successeur;
                free(lp1);
            }
        }
    return l;
}

```

2.1 Les listes simplement chaînées : Implantation par pointeur

Suppression d'une cellule (supprimer une valeur dans une liste) :

Suppression d'un élément quelconque dans la liste (Méthode récurrente)

```
liste supprimerRecurrenteElementListe(element val, liste l)
{
    liste lp;
    if (l!=NULL)
        if (l->valeur==val)
        {
            lp=l;
            l=l->suitant;
            free(lp);
        }
    else
        l->suitant=supprimerRecurrenteElementListe(val, l->suitant);
    return l;
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Recherche dans une liste :

- Étant donnée une liste chaînée `liste` et une valeur `val`, l'objectif ici est d'effectuer un parcours de liste pour rechercher la valeur `val` dans `liste`. Pour ceci, on peut envisager deux types de fonction :
 - une fonction qui teste si la valeur recherchée `val` figure dans la liste `liste`.
 - une fonction qui renvoie l'adresse de la cellule contenant la valeur recherchée `val`.

fonctions qui testent si `val` figure dans `liste`

Recherche dans une liste (Méthode récurrente)

```
int rechercherRecliste(element val, liste l)
{
    if (l==NULL)
        return 0;
    if (l->valeur==val)
        return 1;
    return rechercherRecliste(val, l->suitant);
}
```

Recherche dans une liste (Méthode itérative)

```
int rechercherIterliste(element val, liste l)
{
    while (l!=NULL)
    {
        if (l->valeur==val)
            return 1;
        l=l->suitant;
    }
    return 0;
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Recherche dans une liste :

fonctions qui renvoient l'adresse de la cellule contenant val

Recherche dans une liste (Méthode récursive)

```
liste rechercherLRec(element val, liste l)
{
    if (l==NULL)
        return NULL;
    if (l->valeur==val)
        return l;
    return rechercherLRec(val, l->suisvant);
}
```

Recherche dans une liste (Méthode itérative)

```
liste rechercherLIter(element val, liste l)
{
    while (l!=NULL)
    {
        if (l->valeur==val)
            return l;
        l=l->suisvant;
    }
    return NULL;
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Affichage d'une liste :

- La fonction suivante permet d'afficher les valeurs stockée d'une liste.

Affichage d'une liste

```
void afficherListe(liste l)
{
    while(l != NULL)
    {
        printf("%d ", l->valeur);
        l = l->suitant;
    }
    printf("\n");
}
```

2.1 Les listes simplement chaînées : Implantation par pointeur

Exercice :

Écrire un programme complet en langage C qui permet de

- construire une liste chaînée,
- afficher une liste chaînée,
- calculer la longueur d'une liste chaînée,
- rechercher une valeur dans la liste et de retourner sa position dans la liste.
- effectuer les manipulations suivantes :
 - supprimer une valeur quelconque de la liste,
 - supprimer une valeur en début de liste,
 - supprimer une valeur en fin de liste,
 - supprimer une valeur à une position donnée dans la liste,
 - supprimer une valeur avant ou après une position donnée dans la liste.

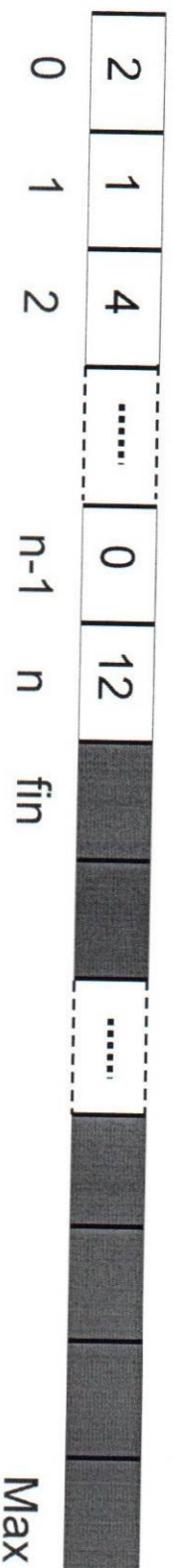
2.1 Les listes simplement chaînées : Implantation par tableau

- Une liste peut être aussi implantée à l'aide d'un tableau.
- Les éléments de la liste seront donc stockés dans les cases contiguës du tableau.
- Comme les listes sont des structures de données qui gèrent des ensembles dynamiques, une taille maximum pour le tableau doit être donc définie lors des déclarations.
- C-à-d, on ne peut pas modifier la taille maximum du tableau lors de l'exécution d'un programme.
- Et si on désire modifier la taille maximum du tableau, il faut modifier aussi le programme.
- Par conséquent, il faut définir un tableau dont la taille sera suffisante pour accueillir la plus grande liste pouvant être utilisée!!
- Mais cette démarche peut encombrer la mémoire de l'ordinateur.
- C'est pourquoi, il est en général préférable d'utiliser l'implantation des listes par des pointeurs.

2.1 Les listes simplement chaînées : Implantation par tableau

Déclaration

- En langage C, pour implanter une liste chaînée à l'aide d'un tableau, on utilise les structures.
- En effet, la liste est un objet constitués :
 - d'un tableau, de taille suffisante pour accueillir la plus grande liste à traiter.
 - d'un entier contenant l'indice de la fin de la liste. Cet entier contient l'indice+1 du dernier élément de la liste, ou la taille de la liste.



- La déclaration correspondante est la suivante, où l'on suppose ici que les valeurs stockées dans la liste sont de type entier.

Code en C

```
typedef int element;
struct listetab
{
    element valeur[taille_MAX];
    int fin;
};
typedef struct listetab listetab, liste;
```

2.1 Les listes simplement chaînées : Implantation par tableau

Création et initialisation

- Pour créer une liste chaînée, on doit l'initialiser à un tableau de taille nulle.
- L'initialisation d'une liste chaînée, ou la création d'une liste vide, est généralement faite par la fonction suivante :

```
int creerListeVide( )  
{  
    return 0;  
}
```

- Pour tester si une liste est vide, on utilise la fonction suivante :

```
int ListeVide(liste l)  
{  
    return l.fin==0;  
}
```

2.1 Les listes simplement chaînées : Implantation par tableau

Ajout d'une valeur à une liste :

- Il est toujours possible d'ajouter des valeurs à une liste chaînée donnée.
- En effet, comme l'implantation par les pointeurs, il y a différentes façons d'ajouter des valeurs à une liste chaînée en particulier on peut
 - ajouter une nouvelle valeur en tête de la liste,
 - ajouter une nouvelle valeur à la fin de la liste.
 - ajouter une nouvelle valeur à une position quelconque de la liste.

Ajout d'un élément en tête de la liste

```

liste ajouterTeteLTab(element val, liste l)
{
    int i;
    l.fin=l.fin+1;
    for(i=l.fin-1;i>=1;i--)
        l.valeur[i]=l.valeur[i-1];
    l.valeur[0]=val;
    return l;
}
  
```

Ajout d'un élément à la fin de la liste

```

liste ajouterFinLTab(element val, liste l)
{
    l.valeur[l.fin]=val;
    l.fin=l.fin+1;
    return l;
}
  
```

2.1 Les listes simplement chaînées : Implantation par tableau

Ajout d'une valeur à une liste :

Ajout d'un élément après une position donnée dans la liste

```
liste ajouterApresPosLTAB(element val, int pos, liste l)
{
    int i;
    l.fin=l.fin+1;
    for(i=l.fin-1; i>=pos+1; i--)
        l.valeur[i]=l.valeur[i-1];
    l.valeur[pos]=val;
    return l;
}
```

Ajout d'un élément après un élément donné dans la liste

```
liste ajouterApresElementLTAB(element val, element elem, liste l)
{
    int i;
    for(i=0; i<l.fin; i++)
        if(l.valeur[i]==elem)
            l=ajouterApresPosLTAB(val, i+1, l);
    return l;
}
```

2.1 Les listes simplement chaînées : Implantation par tableau

Suppression d'une valeur dans une liste :

- Il est possible également de supprimer des valeurs dans une liste donnée.
- En effet, il y a différentes façons de supprimer des valeurs dans des listes :
 - supprimer une valeur en tête de la liste,
 - supprimer une valeur à la fin de la liste,
 - supprimer une valeur quelconque dans une liste.

Suppression d'un élément en tête de la liste

```
liste supprimerTeteTab(liste l)
{
    int i;
    l.fin=l.fin-1;
    for(i=0;i<l.fin;i++)
        l.valeur[i]=l.valeur[i+1];
    return l;
}
```

Suppression d'un élément en fin de la liste

```
liste supprimerFinTab(liste l)
{
    l.fin=l.fin-1;
    return l;
}
```

2.1 Les listes simplement chaînées : Implantation par tableau

Suppression d'une valeur dans une liste :

Suppression d'un élément quelconque dans une liste

```

liste supprimerElementTab(element val, liste l)
{
    int i,j;
    for(i=0;i<l.fin;i++)
        if(l.valeur[i]==val)
            j=i;
    l.fin=l.fin-1;
    for(i=j;i<l.fin;i++)
        l.valeur[i]=l.valeur[i+1];
    return l;
}

```

Suppression d'un élément quelconque à une position donnée d'une liste

```

liste supprimerPoslTab(int pos, liste l)
{
    int i;
    l.fin=l.fin-1;
    for(i=pos-1;i<l.fin;i++)
        l.valeur[i]=l.valeur[i+1];
    return l;
}

```

2.1 Les listes simplement chaînées : Implantation par tableau

Recherche dans une liste :

- L'objectif ici est de rechercher une valeur val dans liste. Pour ceci, on peut envisager deux types de fonction :
 - une fonction qui teste si la valeur recherchée val figure dans la liste.
 - une fonction qui renvoie la position dans la liste de la valeur contenant la valeur recherchée val.

Tester si la valeur recherchée existe dans la liste

```
int rechercherLTab(element val, liste L)
{
    int i;
    for(i=0; i<L.fin; i++)
        if(L.valeur[i]==val)
            return 1;
    return 0;
}
```

Rechercher une position d'un élément d'une liste

```
int rechercherPosLTab(element val, liste L)
{
    int i;
    if(L.fin==0)
        return -1;
    else
        for(i=0; i<L.fin; i++)
            if(L.valeur[i]==val)
                return i+1;
    return 0;
}
```

2.1 Les listes simplement chaînées : Implantation par tableau

Affichage d'une liste :

- La fonction suivante permet d'afficher les valeurs stockées d'une liste.

Affichage d'une liste

```
void afficherLTab(liste l)
{
    int i;
    if(l.fin==0)
        printf("votre liste est vide!!!\n");
    else
    {
        for(i=0;i<l.fin;i++)
            printf("%d ", l.valeur[i]);
        printf("\n\n");
    }
}
```

2.1 Les listes simplement chaînées : Implantation par tableau

Exercice :

Écrire un programme complet en langage C qui permet, en utilisant l'implantation par tableau, de

- saisir une liste chaînée,
- afficher une liste chaînée,
- calculer la longueur d'une liste chaînée,
- rechercher une valeur dans la liste et de retourner sa (ou ses) position(s) dans la liste.
- effectuer les manipulations suivantes :
 - supprimer une valeur quelconque de la liste,
 - supprimer une valeur en début de liste,
 - supprimer une valeur en fin de liste,
 - supprimer une valeur à une position donnée dans la liste,
 - supprimer une valeur avant ou après une position donnée dans la liste.