Chapitre 10

Listes chaînées

1. Structures de données linéaires

Parmi les structures de données linéaires il y a :

- les tableaux,
- les listes chaînées,
- les piles,
- les files.

Les structures de données linéaires induisent une notion de séquence entre les éléments les composant (1er, 2ème, 3ème, suivant, dernier…).

1.1. Les tableaux

Vous connaissez déjà la structure linéaire de type **tableau** pour lequel les éléments de même type le composant sont placés de façon contigüe en mémoire.
Pour créer un tableau, à 1 ou 2 dimensions, il faut connaître sa taille qui ne pourra être modifiée au cours du programme, et lui associer un indice pour parcourir ses éléments. Pour les tableaux la séquence correspond aux numéros des cases du tableau. On accède à un élément du tableau directement grâce à son indice.

Soit le tableau à 1 dimension suivant nommé Tablo :

12	14	10	24	

Pour atteindre la troisième case du tableau il suffit d'écrire Tablo[3] qui contient 10, si les valeurs de l'indice commencent à 1.

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.

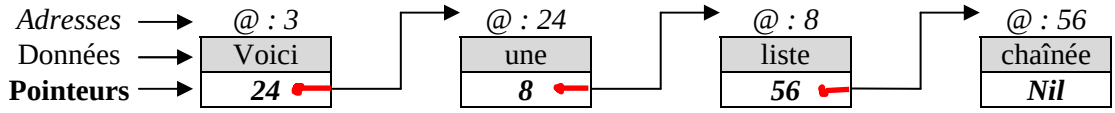
Ce type de stockage de valeurs peut donc être coûteux en temps d'exécution. Il existe une autre structure, appelée liste chaînée, pour stocker des valeurs, cette structure permet plus aisément d'insérer et de supprimer des valeurs dans une liste linéaire d'éléments.

1.2. Les listes chaînées

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.

Pour les listes chaînées la séquence est mise en oeuvre par le pointeur porté par chaque élément qui indique l'emplacement de l'élément suivant. Le dernier élément de la liste ne pointe sur rien (*Nil*). On accède à un élément de la liste en parcourant les éléments grâce à leurs pointeurs.

Soit la liste chaînée suivante (@ indique que le nombre qui le suit représente une adresse) :



Pour accéder au troisième élément de la liste il faut toujours débuter la lecture de la liste par son premier élément dans le pointeur duquel est indiqué la position du deuxième élément. Dans le pointeur du deuxième élément de la liste on trouve la position du troisième élément…
Pour ajouter, supprimer ou déplacer un élément il suffit d'allouer une place en mémoire et de mettre à jour les pointeurs des éléments.

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.

Ces différents types peuvent être mixés selon les besoins.

On utilise une liste chaînée plutôt qu'un tableau lorsque l'on doit traiter des objets représentés par des suites sur lesquelles on doit effectuer de nombreuses suppressions et de nombreux ajouts. Les manipulations sont alors plus rapides qu'avec des tableaux.

Résumé

Structure	Dimension	Position d'une information	Accès à une information
Tableau	Fixe	Par son indice	Directement par l'indice
Liste chaînée	Evolue selon les actions	Par son adresse	Séquentiellement par le pointeur de chaque élément

1.3. Les piles et les files

Les **files** et les **piles** sont des listes chaînées particulières qui permettent l'ajout et la suppression d'éléments uniquement à une des deux extrémités de la liste.

Structures	Ajout	Suppression	Type de Liste
PILE	Tête	Tête	LIFO (Last In First Out)
FILE	Queue	Tête	FIFO (First In First Out)

La pile est une structure de liste similaire à une pile d'assiettes où l'on pose et l'on prend au sommet de la pile.
La file est une structure de liste similaire à une file d'attente à une caisse, le premier client entré dans la file est le premier sorti de celle-ci (aucun resquillage n'est admis).

2. Listes chaînées

2.1. Définitions

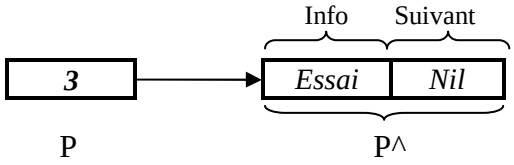
Un **élément** d'une liste est l'ensemble (ou structure) formé :
- d'une donnée ou information,
- d'un pointeur nommé *Suivant* indiquant la position de l'élément le suivant dans la liste.

A chaque élément est associée une adresse mémoire.

Les listes chaînées font appel à la notion de variable dynamique.
Une variable dynamique:
- est déclarée au début de l'exécution d'un programme,
- elle y est créée, c'est-à-dire qu'on lui alloue un espace à occuper à une adresse de la mémoire,
- elle peut y être détruite, c'est-à-dire que l'espace mémoire qu'elle occupait est libéré,
- l'accès à la valeur se fait à l'aide d'un pointeur.

Un **pointeur** est une variable dont la valeur est une adresse mémoire (voir chapitre 9). Un pointeur, noté P, pointe sur une variable dynamique notée P^.
Le **type de base** est le type de la variable pointée.
Le **type du pointeur** est l'ensemble des adresses des variables pointées du type de base. Il est représenté par le symbole ^ suivi de l'identificateur du type de base.

Exemple:



La variable pointeur P pointe sur l'espace mémoire P^ d'adresse 3. Cette cellule mémoire contient la valeur "*Essai*" dans le champ *Info* et la valeur spéciale *Nil* dans le champ *Suivant*. Ce champ servira à indiquer quel est l'élément suivant lorsque la cellule fera partie d'une liste. La valeur Nil indique qu'il n'y a pas d'élément suivant. **P^** est l'objet dont l'adresse est rangée dans P.

Les listes chaînées entraînent l'utilisation de procédures d'allocation et de libération dynamiques de la mémoire. Ces procédures sont les suivantes:
- **Allouer(P)** : réserve un espace mémoire P^ et donne pour valeur à P l'adresse de cet espace mémoire. On alloue un espace mémoire pour un élément sur lequel pointe P.
- **Désallouer(P)** : libère l'espace mémoire qui était occupé par l'élément à supprimer P^ sur lequel pointe P.

Pour définir les variables utilisées dans l'exemple ci-dessus, il faut :
- définir le type des éléments de liste : Type Cellule= Structure
  Info : Chaîne
  Suivant : Liste
  fin Structure
- définir le type du pointeur : Type Liste = ^Cellule
- déclarer une variable pointeur : Var P : Liste
- allouer une cellule mémoire qui réserve un espace en mémoire et donne à P la valeur de l'adresse de l'espace mémoire P^ : Allouer(P)
- affecter des valeur à l'espace mémoire P^: P^.Info ← "Essai" ; P^.Suivant← Nil

Quand P = Nil alors P ne pointe sur rien.

2.2. Listes chaînées simples

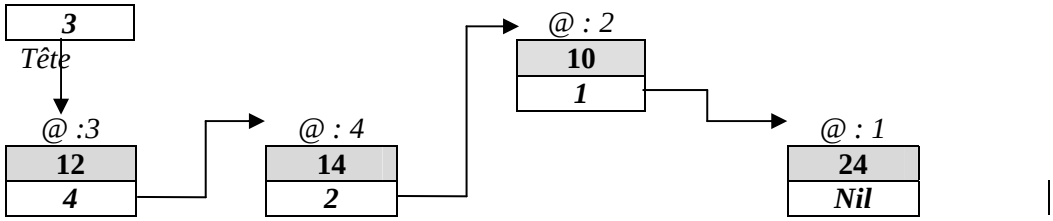
Une liste chaînée simple est composée :

- d'un ensemble d'éléments tel que chacun :
  - est rangé en mémoire à une certaine adresse,
  - contient une donnée (*Info*),
  - contient un pointeur, souvent nommé *Suivant*, qui contient l'adresse de l'élément suivant dans la liste,
- d'une variable, appelée *Tête*, contenant l'adresse du premier élément de la liste chaînée.

Le pointeur du dernier élément contient la valeur *Nil*. Dans le cas d'une liste vide le pointeur de la tête contient la valeur *Nil*. Une liste est définie par l'adresse de son premier élément.

Avant d'écrire des algorithmes manipulant une liste chaînée, il est utile de montrer un schéma représentant graphiquement l'organisation des éléments de la liste chaînée.

Exemple:
Reprenons l'exemple du tableau paragraphe 1.1. page 1. La liste chaînée correspondante pourrait être :



Le 1er élément de la liste vaut 12 à l'adresse 3 (début de la liste chaînée)

Le 2e élément de la liste vaut 14 à l'adresse 4 (car le pointeur de la cellule d'adresse 3 est égal à 4)

Le 3e élément de la liste vaut 10 à l'adresse 2 (car le pointeur de la cellule d'adresse 4 est égal à 2)

Le 4e élément de la liste vaut 24 à l'adresse 1 (car le pointeur de la cellule d'adresse 2 est égal à 1)

Si P a pour valeur 3		***Si P a pour valeur 2***	
P^.Info	a pour valeur 12	P^.Info	a pour valeur 10
P^.Suivant	a pour valeur 4	P^.Suivant	a pour valeur 1

2.3. Traitements de base d'utilisation d'une liste chaînée simple

Il faut commencer par définir un type de variable pour chaque élément de la chaîne. En langage algorithmique ceci se fait comme suit :

```
Type Liste = ^Element
Type Element =  Structure
                    Info :    variant
                    Suivant :       Liste
                Fin structure

Variables Tete, P : Liste
```

Le type de *Info* dépend des valeurs contenues dans la liste : entier, chaîne de caractères, variant pour un type quelconque…

Les traitements des listes sont les suivants :

- Créer une liste.
- Ajouter un élément.
- Supprimer un élément.
- Modifier un élément.
- Parcourir une liste.
- Rechercher une valeur dans une liste.

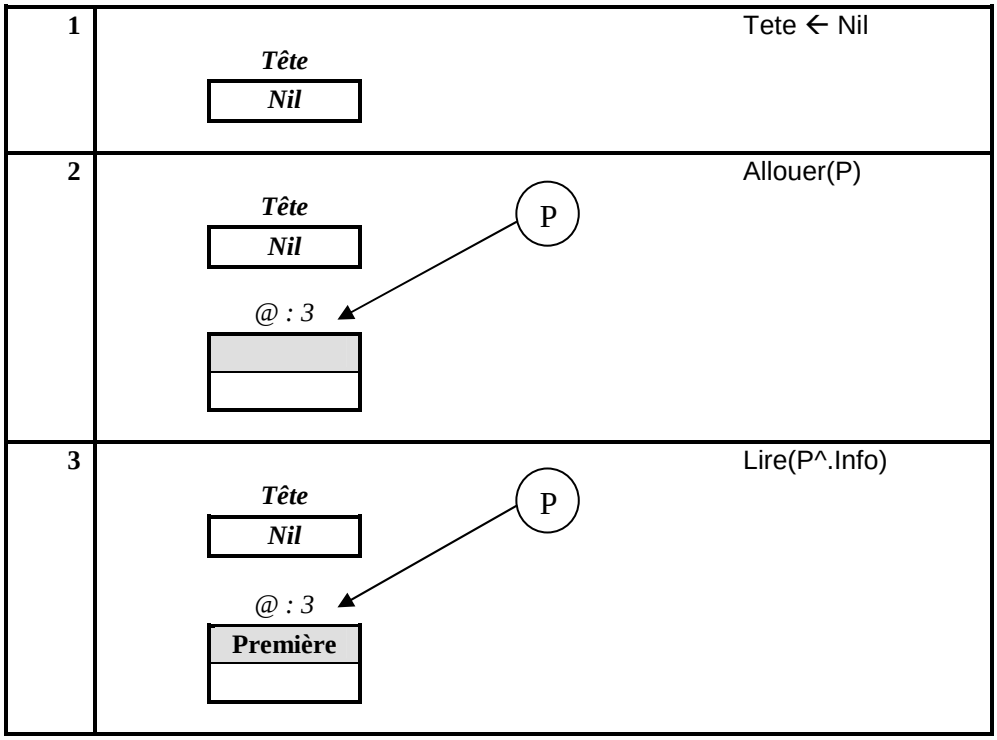
2.3.1 Créer une liste chaînée composée de 2 éléments de type chaîne de caractères

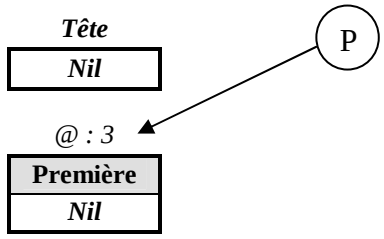
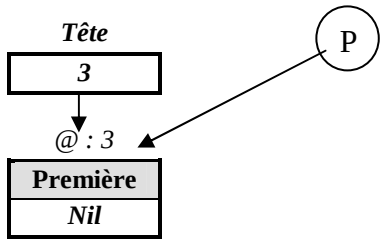
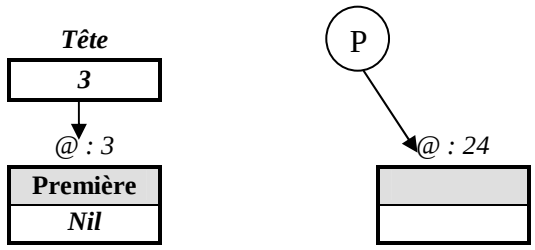
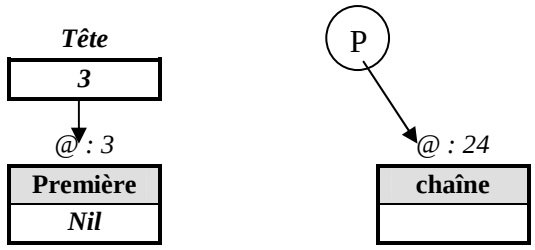
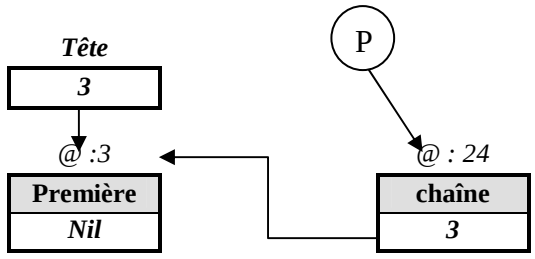
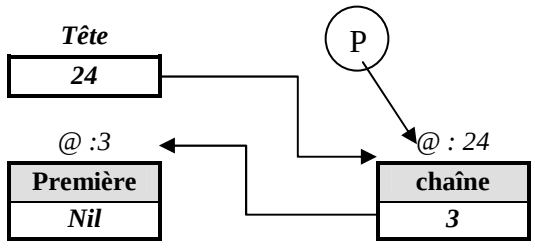
Déclarations des types pour la liste :

```
Type Liste = ^Element
Type Element =  Structure
                    Info :    chaîne de caractères
                    Suivant :       Liste
                Fin structure
```

Algorithme CréationListe2Elements

```
    Tete, P : Liste
    NombreElt : entier
DEBUT
1   Tete ← Nil                      /* pour l'instant la liste est vide*/
2   Allouer(P)                      /* réserve un espace mémoire pour le premier élément */
3   Lire(P^.Info)                   /* stocke dans l'Info de l'élément pointé par P la valeur saisie */
4   P^.Suivant ← Nil                /* il n'y a pas d'élément suivant */
5   Tete ← P                        /* le pointeur Tete pointe maintenant sur P */
    /* Il faut maintenant ajouter le 2e élément, ce qui revient à insérer un élément en tête de liste */
6   Allouer(P)                      /* réserve un espace mémoire pour le second élément */
7   Lire(P^.Info)                   /* stocke dans l'Info de l'élément pointé par P la valeur saisie */
8   P^.Suivant ← Tete               /* élément inséré en tête de liste */
9   Tete ← P
FIN
```



4	P^.Suivant ← Nil 
5	Tete ← P 
6	Allouer(P) 
7	Lire(P^.Info) 
8	P^.Suivant ← Tete 
9	Tete ← P 

2.3.2 Créer une liste chaînée composée de plusieurs éléments de type chaîne de caractères

Déclarations des types pour la liste :

```
Type Liste = ^Element
Type Element =  Structure
                    Info :   chaîne de caractères
                    Suivant :      Liste
                fin Structure
```


Pour créer une liste chaînée contenant un nombre d'éléments à préciser par l'utilisateur il suffit d'introduire deux variables de type Entier *NombreElt* et *Compteur*

- de faire saisir la valeur de *NombreElt* par l'utilisateur dès le début du programme,
- d'écrire une boucle Pour *Compteur allant de 1 à NombreElt* comprenant les instructions 6, 7, 8 et 9.

```
Algorithme CréationListeNombreConnu
    Tete, P : Liste
    NombreElt : entier
    Compteur : entier
DEBUT
    Lire(NombreElt)
    Tete ← Nil
    POUR Compteur DE 1 A NombreElt FAIRE
        Allouer(P)                  /* réserve un espace mémoire pour l'élément à ajouter */
        Lire(P^.Info)               /* stocke dans l'Info de l'élément pointé par P la valeur saisie */
        P^.Suivant ← Tete           /* élément inséré en tête de liste  */
        Tete ← P                    /* le pointeur Tete pointe maintenant sur P */
    FIN POUR
FIN
```

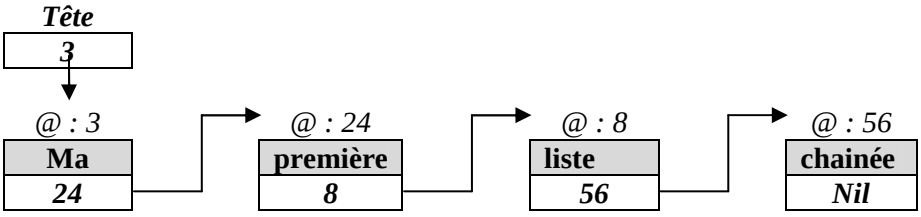
Pour créer une liste chaînée contenant un nombre indéterminé d'éléments il faut :

- déclarer une variable de lecture de même type que celui de l'information portée par la liste,
- déterminer et indiquer à l'utilisateur la valeur qu'il doit saisir pour annoncer qu'il n'y a plus d'autre élément à ajouter dans la chaîne (ici "XXX"),
- écrire une boucle Tant Que permettant d'exécuter les instructions 6, 7, 8 et 9 tant que la valeur saisie par l'utilisateur est différente de la valeur indiquant la fin de l'ajout d'élément dans la chaîne.

```
Algorithme CréationListeNombreInconnu
    Tete, P : Liste
    Valeur : chaîne de caractères
DEBUT
    Tete ← Nil
    Lire (Valeur)
    TANT QUE que Valeur ≠ "XXX" FAIRE
        Allouer(P)                  /* réserve un espace mémoire pour l'élément à ajouter */
        P^.Info ← Valeur            /* stocke dans l'Info de l'élément pointé par P la valeur saisie */
        P^.Suivant ← Tete           /* élément inséré en tête de liste  */
        Tete ← P                    /* le pointeur Tete pointe maintenant sur P */
        Lire (Valeur)
    FIN TANT QUE
FIN
```

2.3.3. Afficher les éléments d'une liste chaînée

Une liste chaînée simple ne peut être parcourue que du premier vers le dernier élément de la liste.



L'algorithme est donné sous forme d'une procédure qui reçoit la tête de liste en paramètre.

```
Procedure AfficherListe  (Entrée      P : Liste)
/* Afficher les éléments d'une liste chaînée passée en paramètre */
  DEBUT
1     P ← Tete                  /* P pointe sur le premier élément de la liste*/
                                /* On parcourt la liste tant que l'adresse de l'élément suivant n'est pas Nil */
      TANT QUE P <> NIL FAIRE   /* si la liste est vide Tete est à Nil */
2           Ecrire(P^.Info)     /* afficher la valeur contenue à l'adresse pointée par P */
3           P ← P^.Suivant      /* On passe à l'élément suivant */
      FIN TANT QUE
  FIN
```

1	P a pour valeur 3
2	"Ma" s'affiche
3	P prend pour valeur 24
2	"première" s'affiche
3	P prend pour valeur 8
2	"liste" s'affiche
3	P prend pour valeur 56
2	"chaînée" s'affiche
3	P prend pour valeur Nil

On s'arrête puisque P a pour valeur Nil et que c'est la condition d'arrêt de la boucle Tant Que.

2.3.4. Rechercher une valeur donnée dans une liste chaînée ordonnée

Dans cet exemple nous reprenons le cas de la liste chaînée contenant des éléments de type chaine de caractères, mais ce pourrait être tout autre type, selon celui déterminé à la création de la liste (rappelons que tous les éléments d'une liste chaînée doivent avoir le même type). La liste va être parcourue à partir de son premier élément (celui pointé par le pointeur de tête). Il a deux cas d'arrêt :

- avoir trouvé la valeur de l'élément,
- avoir atteint la fin de la liste.

L'algorithme est donné sous forme d'une procédure qui reçoit la tête de liste en paramètre. La valeur à chercher est lue dans la procédure.

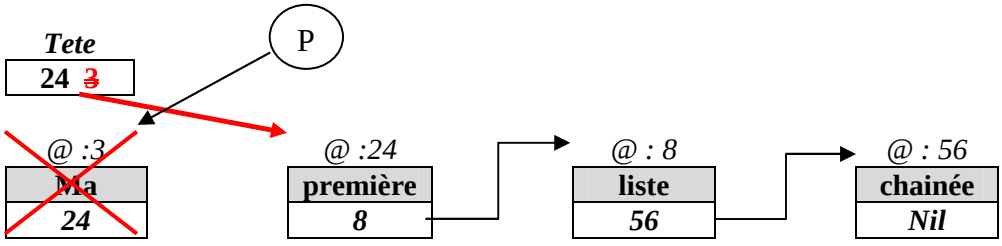
```
Procedure RechercherValeurListe     (Entrée Tete : Liste, Val : variant)
/* Rechercher si une valeur donnée en paramètre est présente dans la liste passée en paramètre */
Variables locales
        P               : Liste     /* pointeur de parcours de la liste */
        Trouve          : booléen   /* indicateur de succès de la recherche */
DEBUT
        SI Tete <> Nil ALORS        /* la liste n'est pas vide on peut donc y chercher une valeur */
                P ← Tete
                Trouve ← Faux
                TANTQUE P <> Nil ET Non Trouve
                        SI P^.Info = Val ALORS      /* L'élément recherché est l'élément courant */
                                Trouve ← Vrai
                        SINON                       /* L'élément courant n'est pas l'élément recherché */
                                P ← P^.Suivant      /* on passe à l'élément suivant dans la liste */
                        FINSI
                FIN TANT QUE
                SI Trouve ALORS
                        Ecrire (" La valeur ", Val, " est dans la liste")
                SINON
                        Ecrire (" La valeur ", Val, " n'est pas dans la liste")
                FINSI
        SINON
                Ecrire("La liste est vide")
        FINSI
FIN
```


2.3.5. Supprimer le premier élément d'une liste chaînée

Il y a deux actions, dans cet ordre, à réaliser :

- faire pointer la tête de liste sur le deuxième élément de la liste,
- libérer l'espace mémoire occupé par l'élément supprimé.

Il est nécessaire de déclarer un pointeur local qui va pointer sur l'élément à supprimer, et permettre de libérer l'espace qu'il occupait.



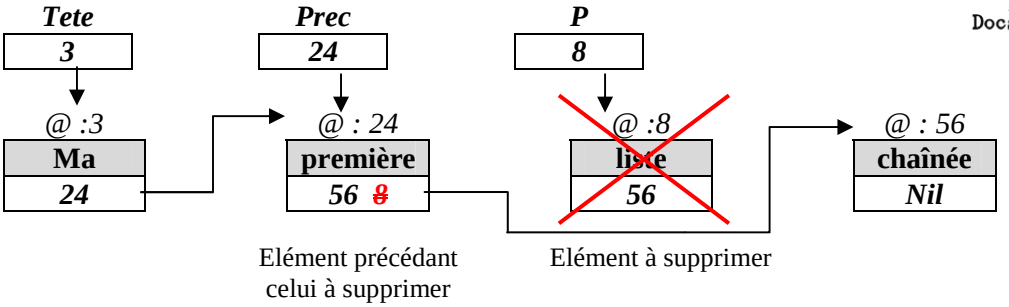

```
Procedure SupprimerPremierElement       (Entrée/Sortie Tete : Liste)
/* Supprime le premier élément de la liste dont le pointeur de tête est passé en paramètre */
Variables locales
        P : Liste                       /* pointeur sur l'élément à supprimer */
DEBUT
        SI Tete <> Nil ALORS            /* la liste n'est pas vide on peut donc supprimer le premier élément */
                P ← Tete                /* P pointe sur le 1er élément de la liste */
                Tete ← P^.Suivant       /* la tête de liste doit pointer sur le deuxième 'élément */
                Desallouer(P)           /* libération de l'espace mémoire qu'occupait le premier élément */
        SINON
                Ecrire("La liste est vide")
        FINSI
FIN
```

2.3.6. Supprimer d'une liste chaînée un élément portant une valeur donnée

Il faut:

- traiter à part la suppression du premier élément car il faut modifier le pointeur de tête,
- trouver l'adresse *P* de l'élément à supprimer,
- sauvegarder l'adresse *Prec* de l'élément précédant l'élément pointé par *P* pour connaître l'adresse de l'élément précédant l'élément à supprimer, puis faire pointer l'élément précédent sur l'élément suivant l'élément à supprimer,
- Libérer l'espace mémoire occupé par l'élément supprimé.

L'exemple considère que l'on souhaite supprimer l'élément contenant la valeur "liste" de la liste ci-dessus.




```
Procedure SupprimerElement  (Entrée/Sortie Tete : Liste, Val : variant)
/* Supprime l'élément dont la valeur est passée en paramètre */
Variables locales
        P       : Liste                 /* pointeur sur l'élément à supprimer */
        Prec    : Liste                 /* pointeur sur l'élément précédant l'élément à supprimer */
        Trouvé : Liste                  /* indique si l'élément à supprimer a été trouvé */
DEBUT
        SI Tete <> Nil ALORS    /* la liste n'est pas vide on peut donc y chercher une valeur à supprimer */
                SI Tete^.info = Val ALORS               /* l'élément à supprimer est le premier */
                        P ← Tete
                        Tete ← Tete^Suivant
                        Desallouer(P)
                SINON                                   /* l'élément à supprimer n'est pas le premier */
                        Trouve ← Faux
                        Prec ← Tete                     /* pointeur précédent */
                        P ← Tete^.Suivant               /* pointeur courant */
                        TANTQUE P <> Nil ET Non Trouve
                                SI P^.Info = Val ALORS  /* L'élément recherché est l'élément courant */
                                        Trouve ← Vrai
                                SINON                   /* L'élément courant n'est pas l'élément cherché */
                                        Prec ← P        /* on garde la position du précédent */
                                        P^ ← P^.Suivant /* on passe à l'élément suivant dans la liste */
                                FINSI
                        FIN TANT QUE
                        SI Trouve ALORS
                                Prec^.Suivant ← P^.Suivant              /* on "saute" l'élément à supprimer "/
                                Desallouer(P)
                        SINON
                                Ecrire ("La valeur ", Val, " n'est pas dans la liste")
                        FINSI
                FINSI
        SINON
                Ecrire("La liste est vide")
        FINSI
FIN
```

2.4. Listes doublement chaînées

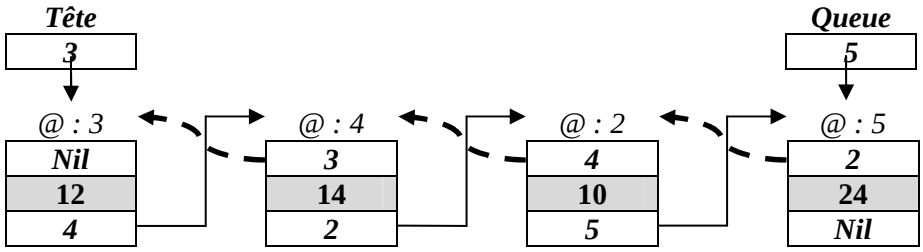
Il existe aussi des liste chaînées, dites bidirectionnelles, qui peuvent être parcourues dans les deux sens, du 1er élément au dernier et inversement.

Une liste chaînée bidirectionnelle est composée :

- d'un ensemble de données,
- de l'ensemble des adresses des éléments de la liste,
- d'un ensemble de pointeurs *Suivant* associés chacun à un élément et qui contient l'adresse de l'élément suivant dans la liste,
- d'un ensemble de pointeurs *Precedent* associés chacun à un élément et qui contient l'adresse de l'élément précédent dans la liste,
- du pointeur sur le premier élément *Tete*, et du pointeur sur le dernier élément, *Queue*,

```
Type ListeDC = ^Element
Type Element =  Structure
                    Precedent : ListeDC
                    Info :      variant
                    Suivant :   ListeDC
                Fin Structure
```

Le pointeur *Precedent* du premier élément ainsi que le pointeur *Suivant* du dernier élément contiennent la valeur *Nil*.



A l'initialisation d'une liste doublement chaînée les pointeurs *Tete* et *Queue* contiennent la valeur *Nil*.

Afficher les éléments d'une liste doublement chaînée

Il est possible de parcourir la liste doublement chaînée du premier élément vers le dernier. Le pointeur de parcours, *P*, est initialisé avec l'adresse contenue dans *Tete*. Il prend les valeurs successives des pointeurs *Suivant* de chaque élément de la liste. Le parcours s'arrête lorsque le pointeur de parcours a la valeur *Nil*. Cet algorithme est analogue à celui du parcours d'une liste simplement chaînée.

```
Procedure AfficherListeAvant   (Entrée       Tete : ListeDC)
Variables locales
        P : ListeDC
  DEBUT
        P ← Tete /
        TANT QUE P <> NIL FAIRE
                Ecrire(P^.Info)
                P ← P^.Suivant
        FIN TANT QUE
  FIN
```

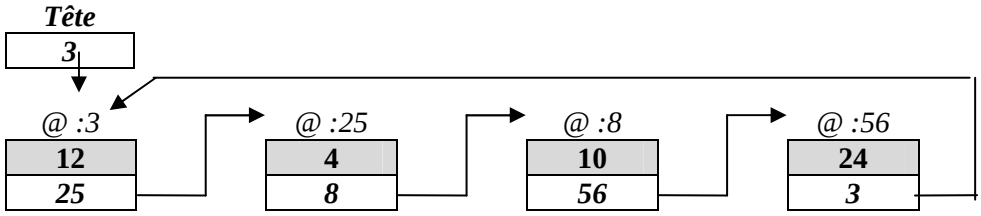
Il est possible de parcourir la liste doublement chaînée du dernier élément vers le premier. Le pointeur de parcours, *P*, est initialisé avec l'adresse contenue dans *Queue*. Il prend les valeurs successives des pointeurs *Precedent* de chaque élément de la liste. Le parcours s'arrête lorsque le pointeur de parcours a la valeur *Nil*.

```
Procedure AfficherListeArriere (Entrée      Queue : ListeDC)
Variables locales
        P : ListeDC
  DEBUT
        P ← Queue
        TANT QUE P <> NIL FAIRE
                Ecrire(P^.Info)
                P ← P^.Precedent
        FIN TANT QUE
  FIN
```

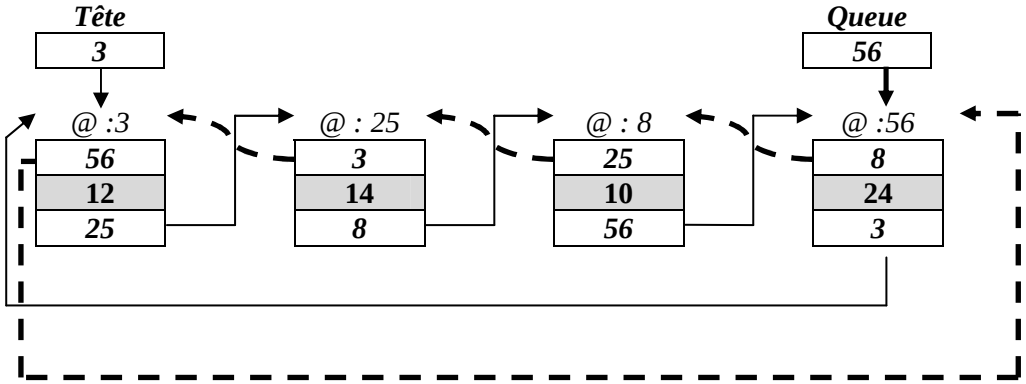

2.5. Listes chaînées circulaires

Une liste chaînée peut être circulaire, c'est à dire que le pointeur du dernier élément contient l'adresse du premier.

Ci-dessous l'exemple d'une liste simplement chaînée circulaire : le dernier élément pointe sur le premier.



Puis l'exemple d'une liste doublement chaînée circulaire. : Le dernier élément pointe sur le premier, et le premier élément pointe sur le dernier.



WEBOGRAPHIE

http://www.siteduzero.com/tutoriel-3-36245-les-listes-chainees.html
http://liris.cnrs.fr/pierre-antoine.champin/enseignement/algo/listes_chainees/
http://deptinfo.cnam.fr/Enseignement/CycleA/SD/cours/structuress%E9quentielleschain%E9es.pdf
http://pauillac.inria.fr/~maranget/X/421/poly/listes.html#toc2
http://wwwens.uqac.ca/~rebaine/8INF805/courslistespilesetfiles.pdf