Listes chainées

Fait par : Mme Nouzri Sana
sana.nouzri@gmail.com

Chapitre II

Listes chaînées

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.

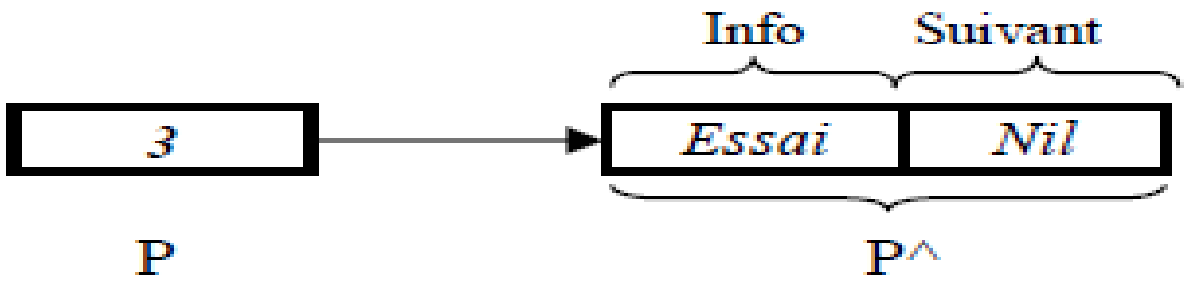
Les listes chaînées font appel à la notion de variable dynamique.
Une variable dynamique:

- 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.

Listes chaînées

Définitions

- Un **pointeur** est une variable dont la valeur est une adresse mémoire.
- 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.



Procédures d’allocation

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.

Déclaration et définition

Définir le type des éléments de liste

```
Type Element= Structure
Champs_1: Chaîne
Suivant : ^ Element
fin Structure
```

Déclarer une variable pointeur

```
Var P : ^ Element
```

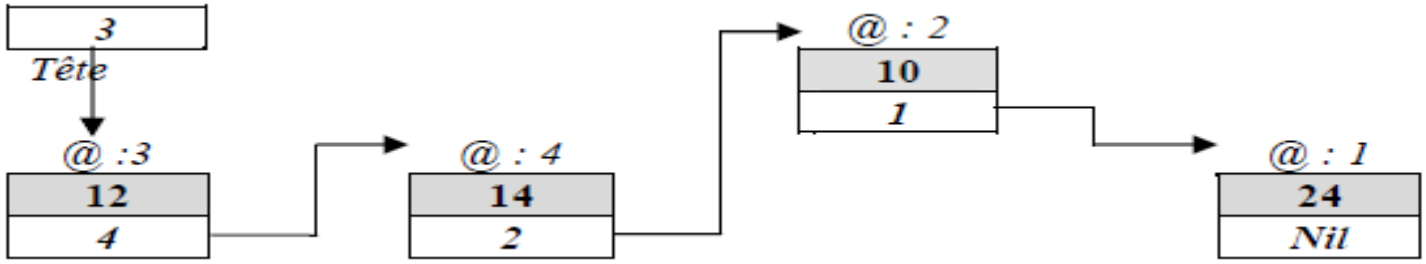
- Allouer une cellule mémoire qui réserve un espace en mémoire et donne à P la valeur de l'adresse de l'espace mémoire **P^ : Allouer(P)**
- Affecter des valeur à l'espace mémoire **P^: P^.Champs_1<-"Essai" ;**
  P^.Suivant<- Nil

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.



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

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.

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

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

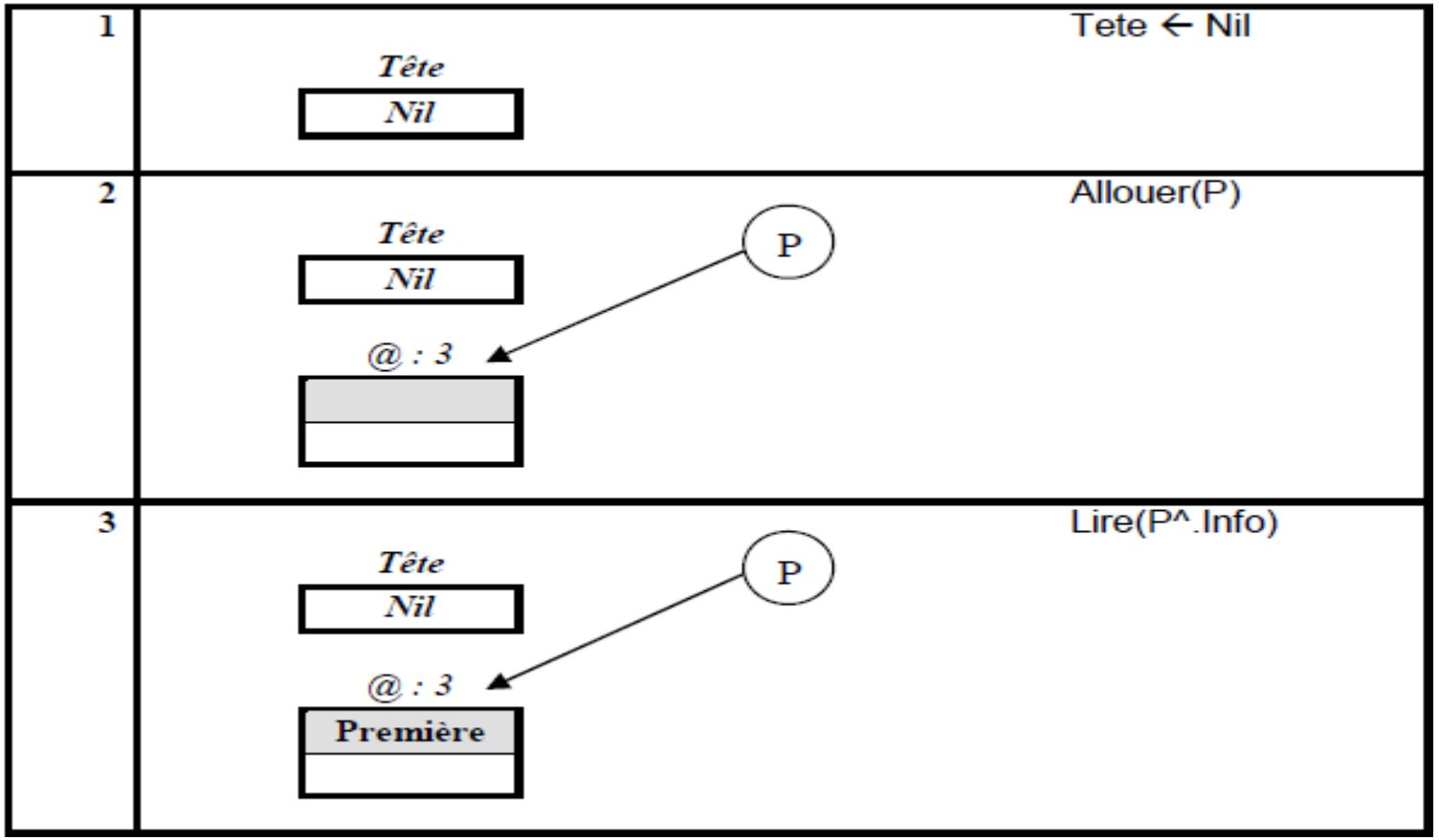
Déclarations des types pour la liste :
Type Liste = ^Element
Type Element = Structure
Info : chaîne de caractères
Suivant : ^Element
Fin structure

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

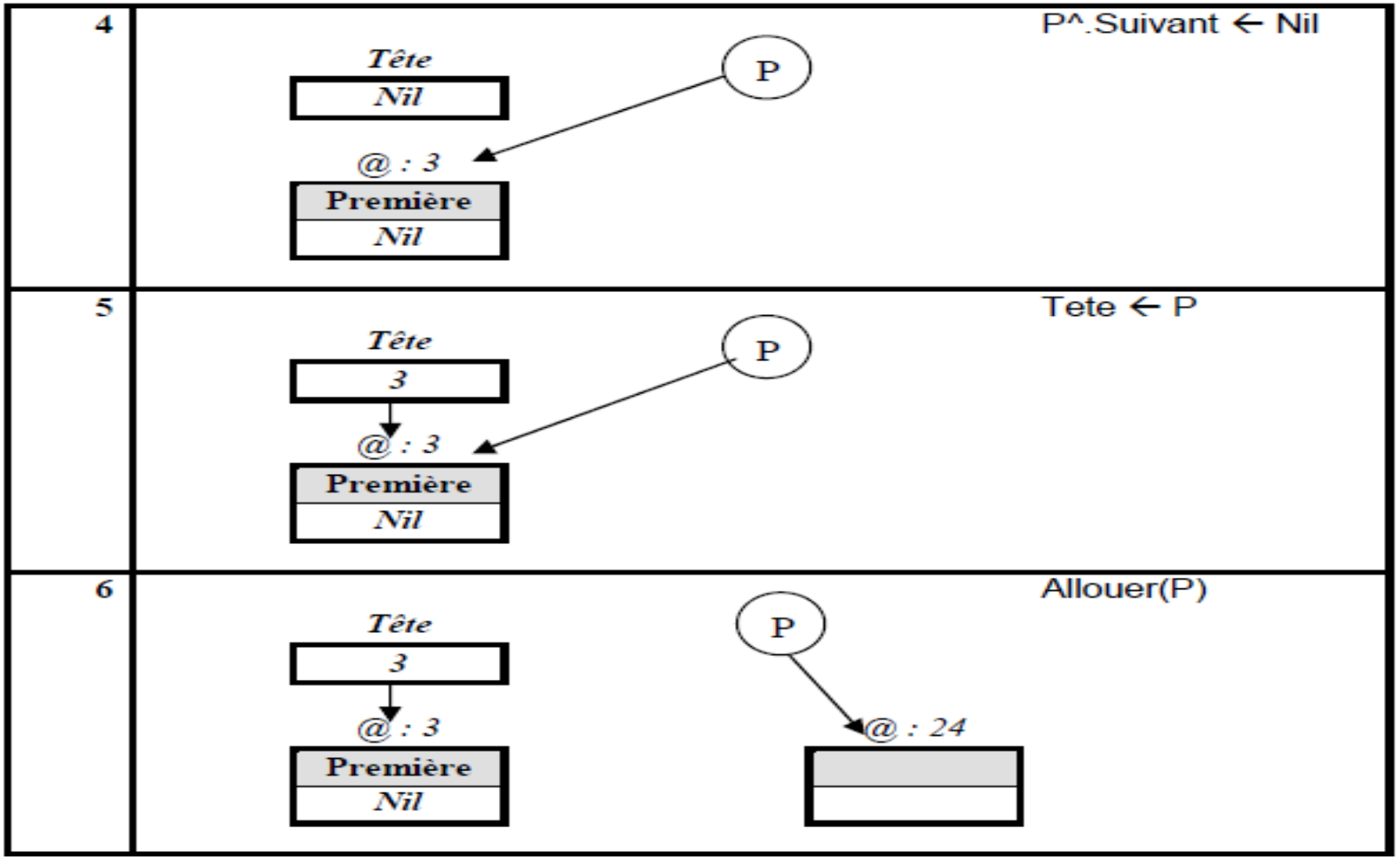
(suite)

```
Algorithme CréationListe2Elements
Tete, P : ^Element
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
```

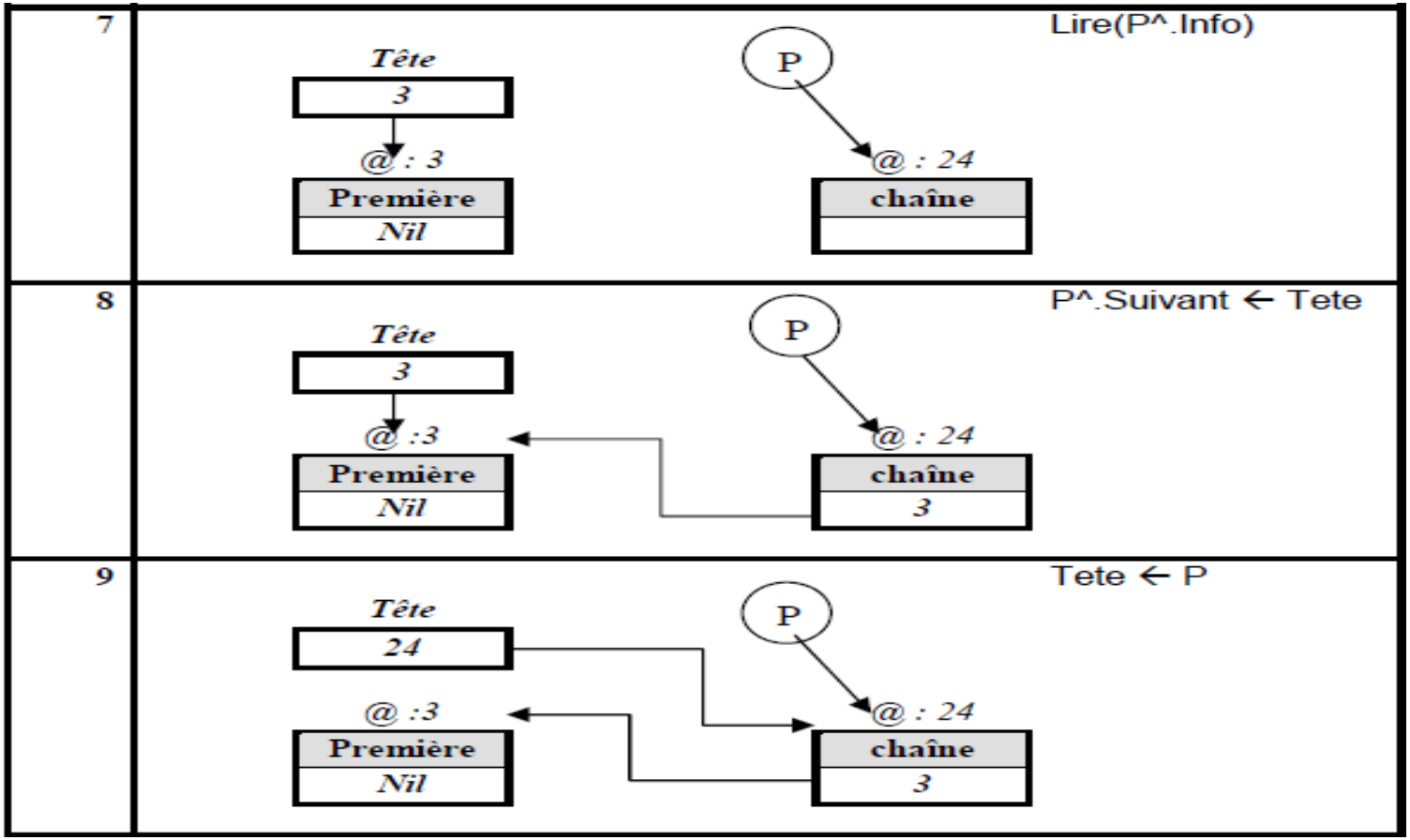
Traitements de base d'utilisation d'une liste chaînée simple



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



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



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

Créer une liste chaînée contenant un nombre d'éléments à préciser par l'utilisateur

```
Algorithme CréationListeNombreConnu
Tete, P : ^Element
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
```

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

Créer une liste chaînée contenant un nombre indéterminé d'éléments

```
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
```

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.

Procedure AfficherListe (*Entrée P : Liste)*

++/* Afficher les éléments d'une liste chaînée passée en paramètre */
DEBUT
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 */
Ecrire(P^.Info) /* afficher la valeur contenue à l'adresse pointée par P */
P <-P^.Suivant /* On passe à l'élément suivant */
FIN TANT QUE
FIN

Rechercher une valeur donnée dans une liste chaînée ordonnée

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.

(Suite)

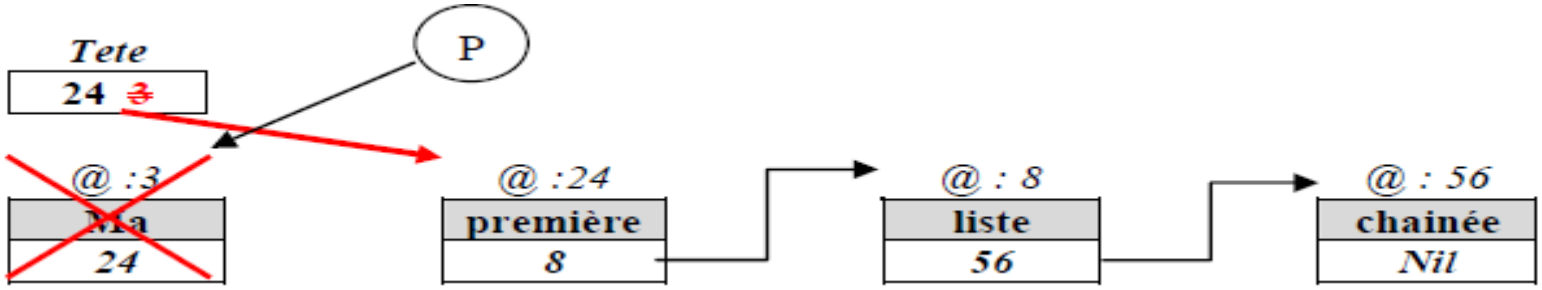
```
Procedure RechercherValeurListe (Entrée Tete : ^Element, Val : variant)
/* Rechercher si une valeur donnée en paramètre est présente dans la liste passée en paramètre */
Variables locales
P : ^Element          /* pointeur de parcours de la liste */
Trouve : booléen    /* indicateur de succès de la 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
```

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.



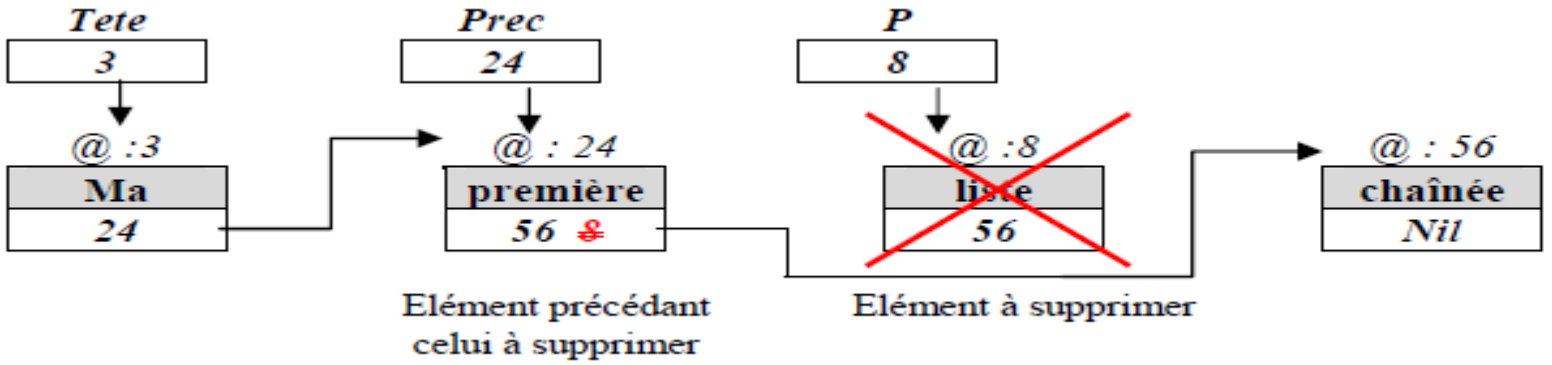
Supprimer le premier élément d'une liste chaînée

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 : ^Êlement /* 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

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.



Procedure SupprimerElement (*Entrée/Sortie Tete : ^Element, Val : variant)*

/* Supprime l'élément dont la valeur est passée en paramètre */

Variables locales

P : ^ ***Element*** /* pointeur sur l'élément à supprimer */

Prec : ^ ***Element*** /* pointeur sur l'élément précédant l'élément à supprimer */

Trouvé : ^ ***Element*** /* 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

(Suite)

```
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
```

Listes doublement chaînées

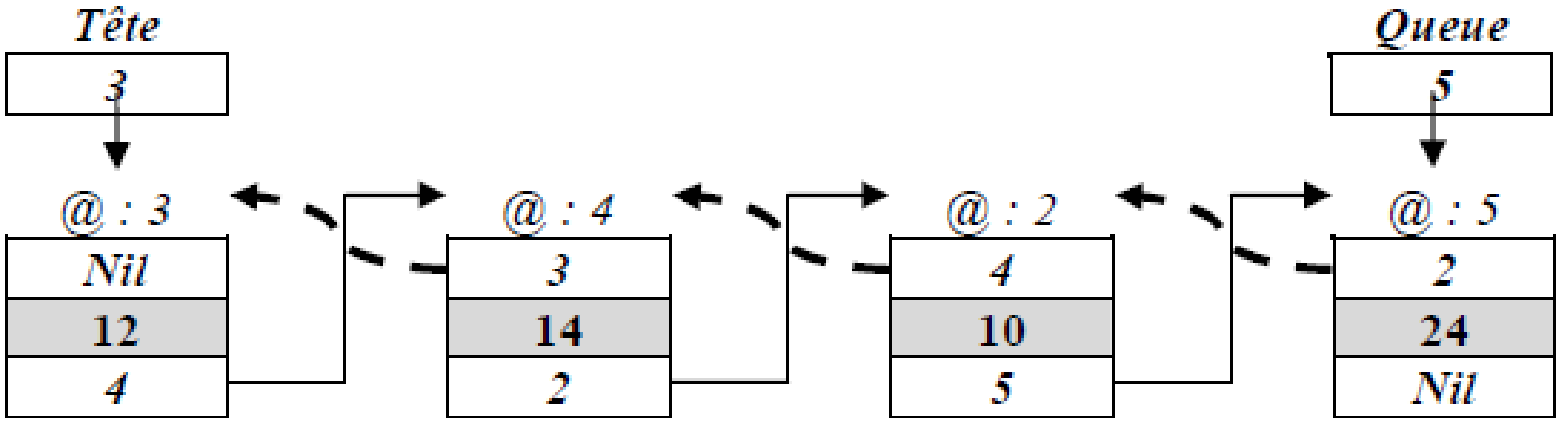
Il existe aussi des liste chaînées, dites **bidirectionnelles**, qui peuvent être parcourues dans les deux sens, du 1er élément au dernier et inversement.
Une liste chaînée bidirectionnelle est composée :

- 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**,

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

Listes doublement chaînées

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



Listes doublement chaînées

Déclaration et définition

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

Listes doublement chaînées

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 : ^Element)
Variables locales
P : ^ Element
DEBUT
P <-Tete /
TANT QUE P <> NIL FAIRE
Ecrire(P^.Info)
P <-P^.Suivant
FIN TANT QUE
FIN
```

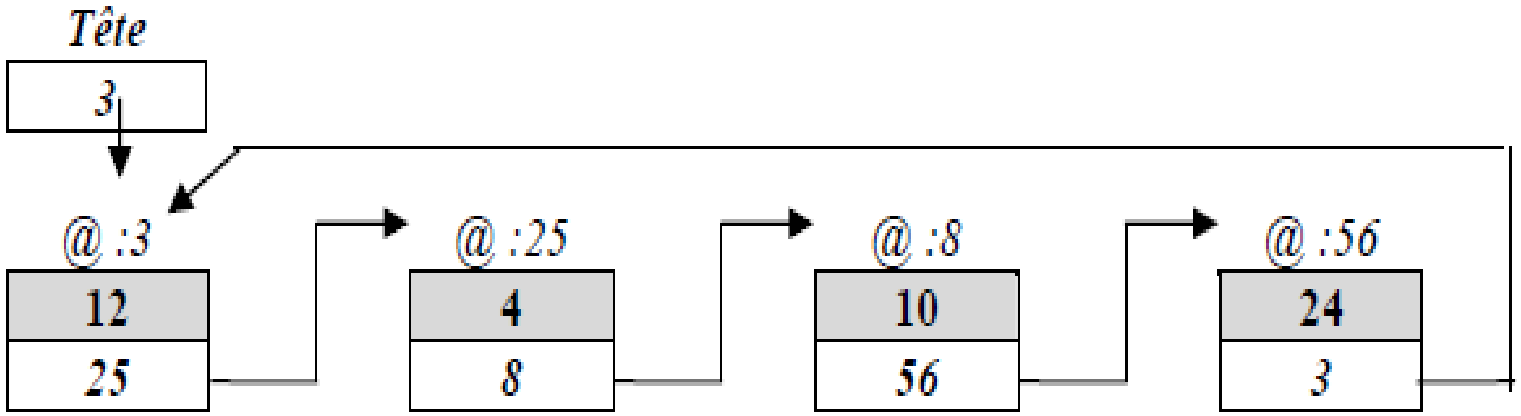
Listes doublement chaînées

- 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 : ^Element)
Variables locales
P : ^ Element
DEBUT
P ^<-Queue
TANT QUE P <> NIL FAIRE
Ecrire(P^.Info)
P <-P^.Precedent
FIN TANT QUE
FIN
```

Listes chaînées circulaires

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



Liste doublement chaînée circulaire

Liste doublement chaînée circulaire. : Le dernier élément pointe sur le premier, et le premier élément pointe sur le dernier.