Exercice 01 *:*

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

// **Déclarations des types pour la liste :**

Programme CréationListe2Elements

Type Liste = ^Element

Element = Enregistrement

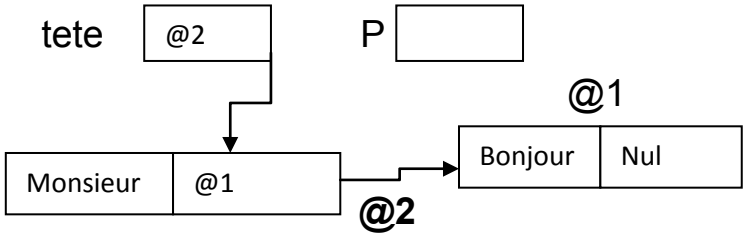
Info : chaîne de caractères ;

Suivant : Liste

Fin ;

Variable Tete , P : Liste ;

NombreElt : entier ;



DEBUT

Tete := Nil ; // pour l'instant la liste est vide

Allouer(P) ; // réserve un espace mémoire pour le premier élément

ecrire ('Donnez la valeur de info') ;

Lire (P^.Info) ; // stocke dans l'Info de l'élément pointé par P la valeur saisie

P^.Suivant := Nil ; // il n'y a pas d'élément suivant

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

Allouer(P) ; // réserve un espace mémoire pour le second élément

Ecrire (' donnez la valeur de l''information') ;

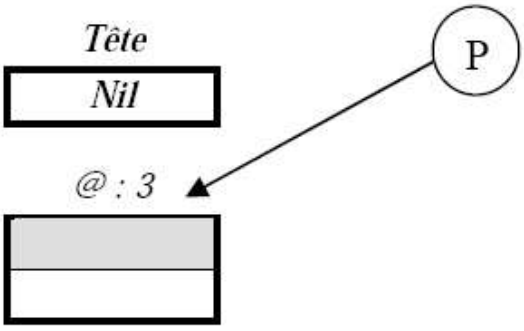
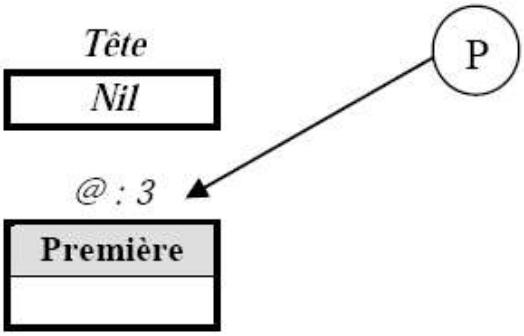
Lire(P^.Info) ; // stocke dans l'Info de l'élément pointé par P la valeur saisie

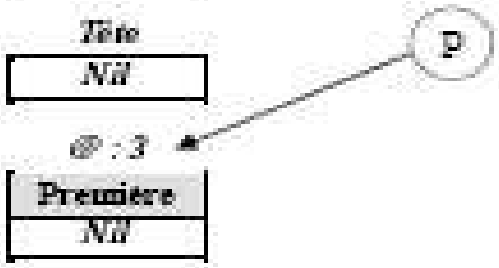
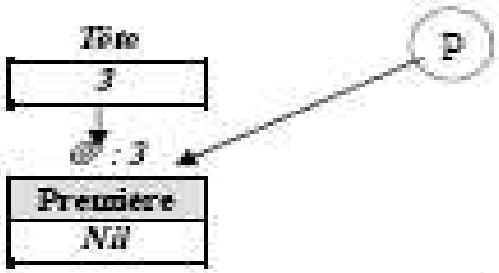
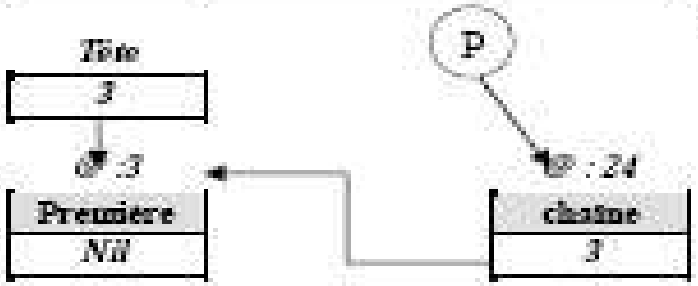
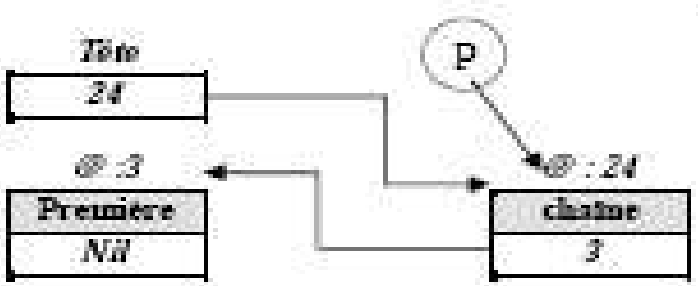
P^.Suivant := Tete ; // élément inséré en tête de liste

Tete := P ;

Liberer (P)

FIN.

1	Tete ← Nil 
2	Allouer(P) 
3	Lire(P^.Info) 

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

Exercice 02 **:**
Créer une liste chaînée composée de plusieurs éléments de type chaîne de caractères

Déclarations des types pour la liste :

```
Type Liste = ^Element
     Element = Enregistrement
               Info : chaîne de caractères ;
               Suivant : Liste
Fin ;
```

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,
- Ecrire une boucle Pour ***Compteur*** *allant de 1 à* ***NombreElt*** comprenant les instructions 6, 7, 8 et 9.

Programme CréationListeNombreConnu

```
Type Liste = ^Element
     Element = Enregistrement
               Info : chaîne de caractères ;
               Suivant : Liste
Fin.

Variable Tete, P : Liste ;
         NombreElt : entier ;
         i : entier ;

DEBUT
     Ecrire ('donnez le nombre d''éléments à saisir') ;

     Lire(NombreElt) ;

     Tete := Nil ;

     POUR i := 1 A NombreElt FAIRE
          Allouer(P) ;
                                   //  Réserve un espace mémoire pour l'élément à ajouter
          Ecrire ('donnez l''information') ;
          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 Tête pointe maintenant sur P
     FIN
FIN.
```

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.

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

```
Variable Tete, P : Liste ;
         Valeur : chaîne de caractères ;

DEBUT
        Tete := Nil ;
        Ecrire ('donnez la valeur sachant que la condition de sortie = stop') ;
        Lire (Valeur)

        TANT QUE Valeur ≠ 'stop' FAIRE
                debut
                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 */

                Ecrire (' introduire une nouvelle chaine de valeur') ;
                Lire (Valeur)
                fin

FIN.
```

Algorithme CréationListeNombreInconnu

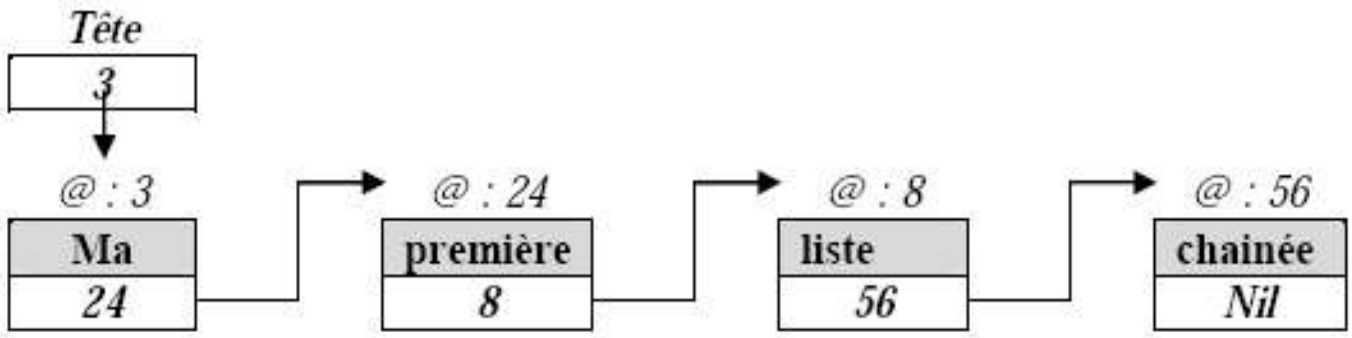
```
Variable Tete, P : Liste ;
         Valeur : chaîne de caractères ;
        Bool : booleen ;

DEBUT
        Tete := Nil ;
        Ecrire ('donnez la valeur sachant que la condition de sortie = stop') ;
        Lire (Valeur)
        Debut
        Bool := oui ;

        Repeter
                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 */

                Ecrire (' voulez vous continuer  oui / non') ;
                Lire(bool)
                Si bool = oui alors
                        debut
                        Ecrire (' donnez la valeur de la chaine à saisir ') ;
                        Lire(valeur)
                        Fin
                Sinon
```

Ecrire(‘ merci pour votre visite’)

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.



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
        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 */
                debut
                Ecrire(P^.Info) ;                    /* afficher la valeur contenue à l'adresse pointée par P */
                 P :=   P^.Suivant                    /* On passe à l'élément suivant */
                fin
FIN
```

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

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.

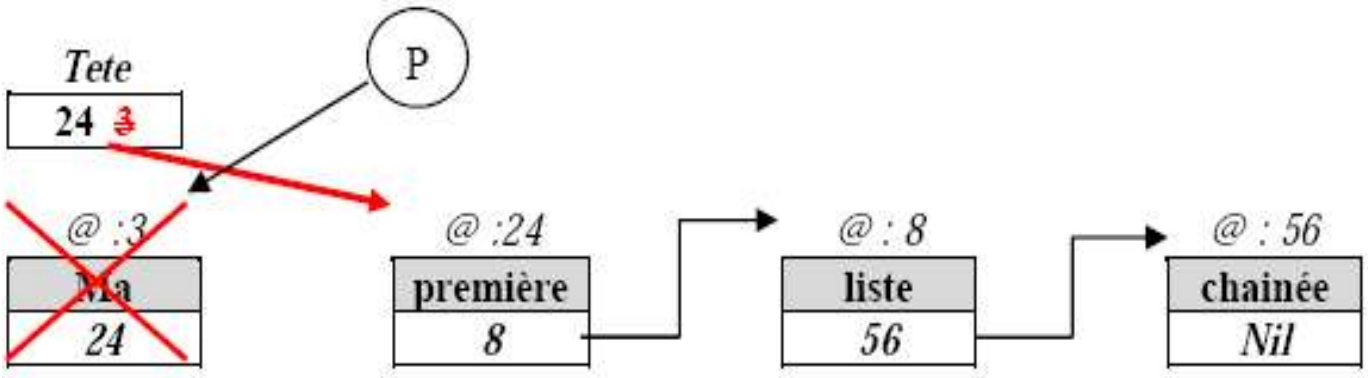
```
Procedure RechercherValeurListe (Entrée Tete : Liste, Val : variant)

/* Rechercher si une valeur donnée en paramètre est présente dans la liste passée en paramètre */


Variable  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 faire
                        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.



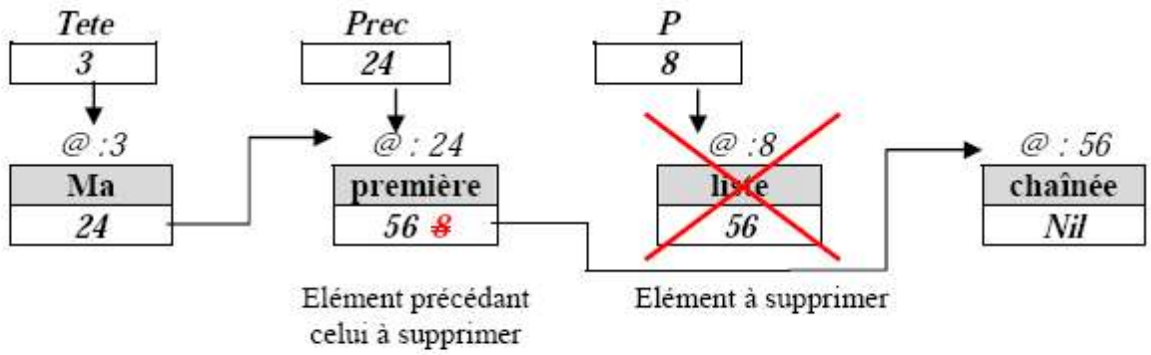
```
Procedure SupprimerPremierElement (Tete : Liste)

/* Supprime le premier élément de la liste dont le pointeur de tête est passé en paramètre */
Variables
        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.
```

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 de 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 (Tete : Liste, Val : variant)

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

Listes doublement chaînées

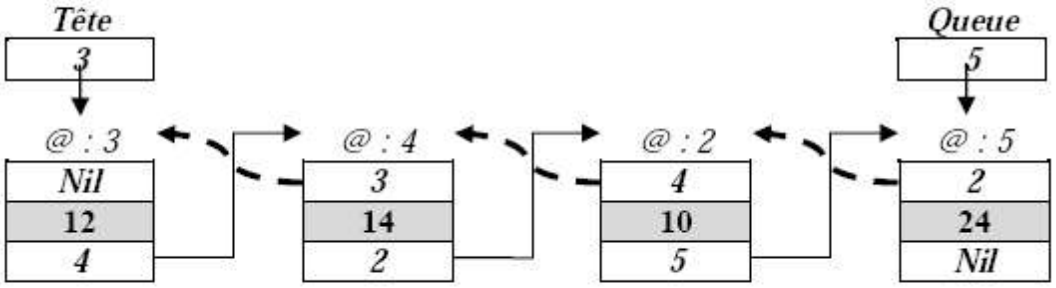
Il existe aussi des liste chaînées, dites bidirectionnelles, qui peuvent être parcourues dans les deux sens, du 1er élément au dernier et inversement.

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 = enregistrement
                Precedent : ListeDC
                Info : variant
                Suivant : ListeDC
Fin
```

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 (Tete : ListeDC)

Variables  P : ListeDC ;
DEBUT
        P :=    Tete ;

        TANT QUE P <> NIL FAIRE
                Ecrire(P^.Info)
                P :=    P^.Suivant
        FIN TANT QUE
FIN.
```

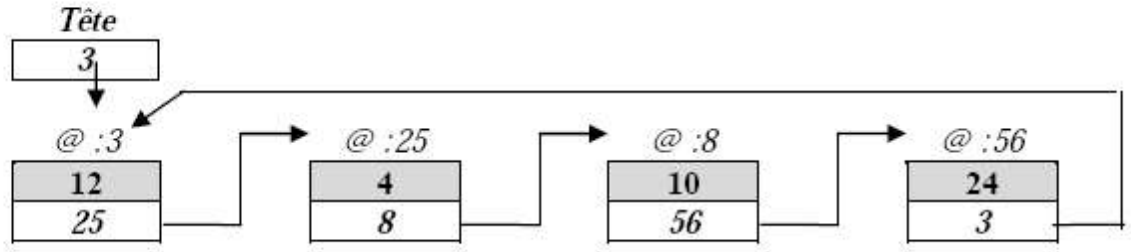
Remarque :

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 (Queue : ListeDC)

Variables  P : ListeDC ;
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. 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.