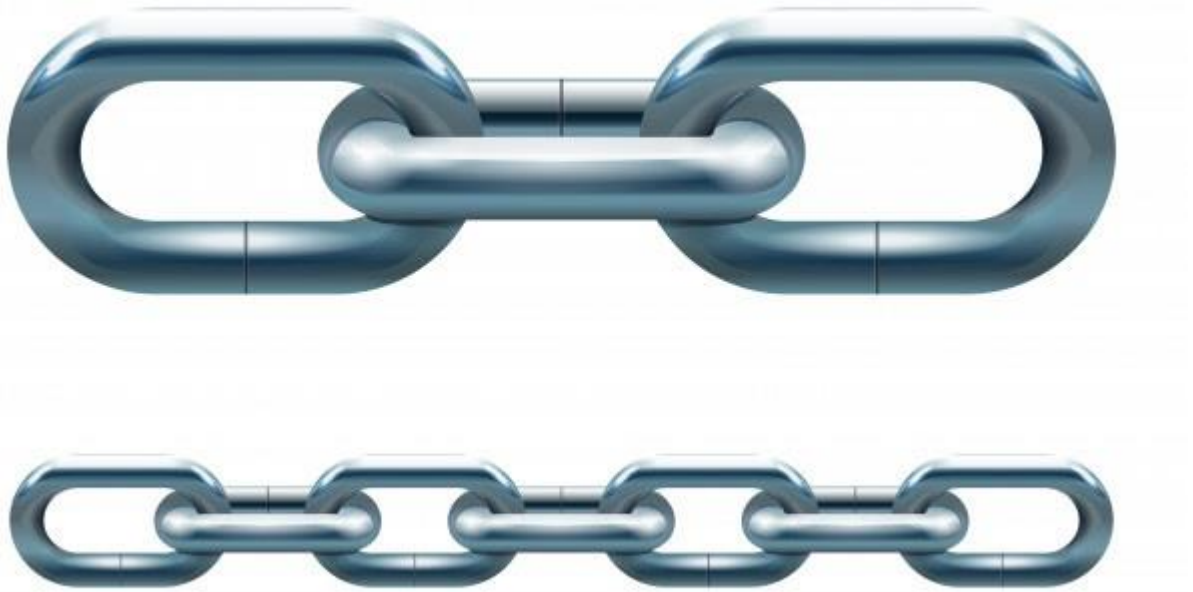
CFI-CIRAS

ALGORITHME AVANCE

Chapitre 3:

Les listes linéaires chaînées

Semestre III
Licence 2 Info



Sommaire

1. Introduction

Dans la première partie de ce module (Initiation à l'algorithmique), nous avons vu et manipulé les types simples de données (entier, réel, caractère, booléen), le type Tableau et le type Chaîne de caractères. Ce sont les types prédéfinis d'un langage de programmation (ils existent tels quels).

Tous les langages de programmation offrent à l'utilisateur la possibilité de définir de nouveaux types de données plus sophistiqués, permettant d'imaginer des traitements à la fois plus performants et plus souples. C'est ce que nous étudierons dans ce chapitre.

Nous avons déjà vu comment le tableau permettait de désigner sous un seul nom un ensemble de valeurs de même type, chacune d'entre elles étant repérée par un indice.

L'enregistrement, quant à elle, nous a permis de désigner sous un seul nom un ensemble de valeurs pouvant être de types différents. L'accès à chaque élément de la structure (nommé Champ ou membre) s'effectue, cette fois, non plus par une indication de position, mais par son nom au sein de la structure.

2.Rappel sur les pointeurs

Lorsqu'on déclare une variable et ce, quel que soit le langage de programmation, le compilateur réserve en mémoire l'espace nécessaire au contenu de cette variable à une donnée. Toute variable possède donc une adresse en mémoire. La plupart du temps on ne s'intéresse pas à ces adresses. Mais quelque fois, ce type de renseignement peut s'avérer fort utile.

Un **pointeur** est une **variable** qui au lieu de contenir l'information (donnée, valeur) proprement dite, contient l'**adresse** mémoire d'une autre entité informatique.

Illustration :

Mémoire RAM

1000	1001	1002	1003	1004	1005
1006	1007	1008	1009	1010	1011
1012	1013	1014 71 X	1015	1016	1017
1018	1019	1020	1021	1022	1023
1024	1025	1026	1027	1028 1014 Y	1029
1030	1031	1032	1033	1034	1035

Variables X classique

Variables Y déclarée comme pointeur

A la différence de la variable **X** classique qui contient directement l'information « **71** », la variable **Y** déclarer comme pointeur contient son adresse. On dit que Y pointe vers l'information « 71 » grâce à son contenu, qui est l'adresse de cette information.

2.Rappel sur les pointeurs

Syntaxe de déclaration d'une variable pointeur :

<NomVariable> : ↑ <Type de donnée pointée>

Exemple : **age : ↑ Entier** **taille : ↑ Réel**

Note : Pour affecter une valeur à une variable pointeur, on utilise le symbole **& (Et commercial)** qui est appelé **opérateur d'adressage.**

Illustration :

```
a : Entier            // a variable de type entier
taille : ↑ Entier     // taille variable pointeur sur un entier
a ← 17                // affectation d'une valeur à la variable a
taille ← &a           // affectation d'une valeur à la variable pointeur taille
```

3. Gestion dynamique de la mémoire

Deux procédures permettent de gérer dynamiquement la mémoire :

- La procédure **nouveau (p : pointeur)** qui permet à chaque appel d'obtenir (**allouer** ou **réserver**) un espace mémoire dont l'adresse sera retournée dans la variable **pointeur p**.
- La procédure **liberer (p : pointeur)** qui permet de libérer (**laisser** ou **abandonner**) un espace mémoire **d'adresse p** dont on a plus besoin.

Ces deux procédures permettent d'obtenir et de rendre un espace mémoire au fur et à mesure des besoins de l'algorithme. On parle de gestion **dynamique de la mémoire** contrairement à la gestion statique des tableaux (taille ou dimension fixe).

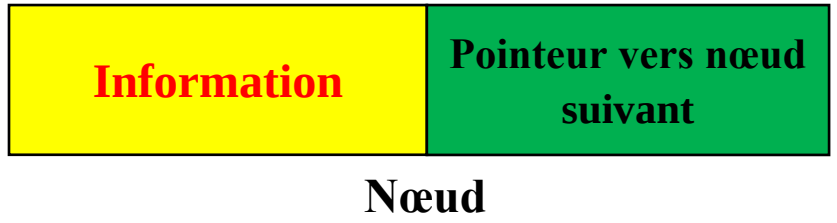
4. Définition d'une liste chaînée

Une **liste chaînée** est un ensemble **d'élément** d'information appelés **nœuds** (ou **maillon**). En plus de l'information dont il est porteur, un nœud possède un champ **pointeur**.

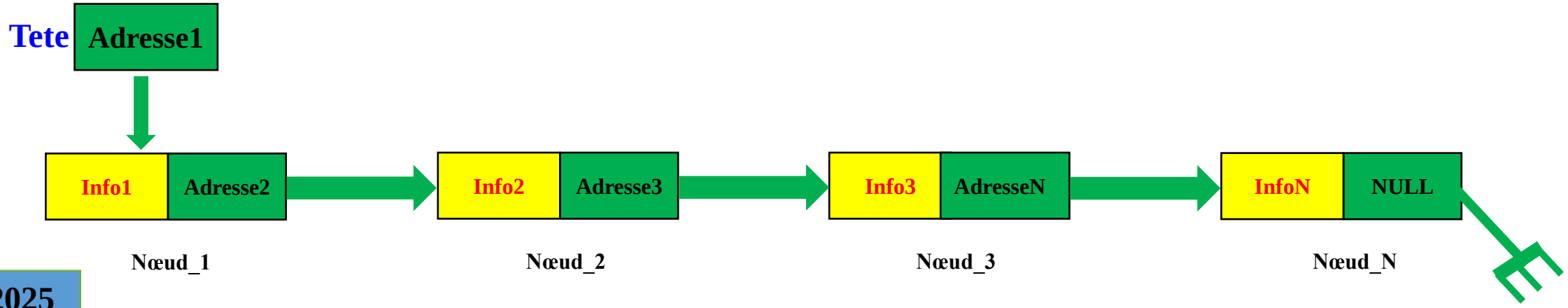
Ce pointeur contient l'adresse du nœud suivant dans la liste.

Une liste chaînée est déterminée grâce à l'adresse de son **premier nœud**. Cette adresse doit être sauvegarder dans une variable pointeur que nous appellerons très souvent **début** ou **tête**.

- **Représentation graphique d'un nœud :**



- **Représentation graphique d'une liste chaînée simple :**



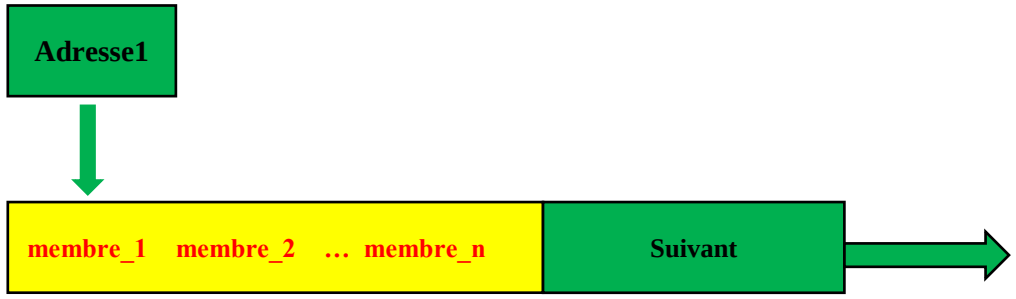
4. Définition d'une liste chaînée

Une liste chainée n'est **pas limitée à un nombre fixe d'élément**. On peut insérer et supprimer des éléments de la liste autant que nécessaire, et ce, sans avoir besoin de connaitre, à priori, le nombre d'élément de la liste : c'est la **gestion dynamique de la mémoire**. Elle apporte donc une réponse optimale à bon nombre de problèmes d'implantation de l'information dans un algorithmes

➢ **Syntaxe de déclaration d'un nœud :**

```
TYPE pointeur = ↑ nœud
        nœud = structure
                membre_1 : Type1
                membre_2 : Type2
                    .....
                membre_n : TypeN
                suivant : pointeur
        Fin_structure
```

On définit un type nommé **pointeur** sur une variables de type nœud, puis on définit le type nœud. Une variables P de type pointeur contient l'adresse d'une variable de type nœud. La variable P peut être représentée schématiquement par :



L'adresse du nœud est rangée dans la variable P. par abus de langage, on dira : nœud d'adresse P ou nœud pointé par P. Le nœud contient deux parties : La partie information contenant les variables membre1, membre2, … et le pointeur suivant contenant une adresse.

4. Définition d’une liste chaînée

On accède aux variables membres du nœud d’adresse P grâce à l’opérateur **point (.)** comme suit :

P↑. membre1, P↑.membre2, …

On accède de la même façon à l’adresse contenue dans le nœud d’adresse P :

P↑.suivant

5. Opération sur les listes chaînées

Dans ce qui suit, nous allons créer, consulter, insérer et supprimer des éléments d'un liste chaînée. Pour ce faire, considérons que chaque nœud « Personne » de la liste contient deux informations : un entier « numéro » et une chaîne de caractère « nom » de 20 caractères.

```
TYPE pointeur = ↑ Personne
    Personne = structure
        Numero : entier
        Nom : chaîne de 20 caractères
        Suivant : pointeur
    Fin_structure
```

5. Opération sur les listes chaînées

5.1. Création d'une liste chaînée simple

Principe :

1. Déclarer et Initialiser la Tete à vide (**Nil**) ;
2. Allouer un espace mémoire P pour le premier nœud grâce à la procédure allouer ou nouveau;
3. Stocke la valeur dans le champ Information du nœud pointé par P ;
4. P suivant est **Nil**, s'il n'y a pas d'élément suivant ;
5. Le pointeur **Tete** pointe maintenant sur P.

```
Procédure CreerListe(Tete : ^Liste, Elt : entier) : Vide
        Variables :
                P : ^Liste
        Début
                Tete←Nil
                Allouer(P)
                P^.Valeur←Elt
                P^.Suivant ←Nil
                Tete←P
        Fin
FinProcedure
```

Supposons que nous voulions créer la liste chaînée qui soit la représentation des nœuds de type « Personne ». On déclare pour cela deux pointeurs « tete » et « courant » de type « Personne » :

- « tete » pour contenir l'adresse du 1er élément de la liste ;
- « courant » pour contenir l'adresse de l'élément en cours (celui qui subit un traitement).

5. Opération sur les listes chaînées

5.1. Création d'une liste chaînée simple

```
Algorithme Création_Liste
    TYPE pointeur = ↑ Personne
        Personne = structure
            Numero : entier
            Nom : chaîne de 20 caractères
            suivant : pointeur
        Fin_structure
    Variables :
        Tete, courant : pointeur
        Reponse : Car
Début
    Nouveau(Tete)
    courant ← Tete
    Répéter
        Ecrire (″Entrez le numéro de la personne ″)
        Lire (courant↑.Numero)
        Ecrire (″ Entrez le nom de la personne ″)
        Lire (courant↑.Nom)
        Ecrire(″ Autre élément o/n ? ″)
        Lire(Reponse)
        Si (Reponse == ′o′) alors
            Nouveau(courant↑.suivant)
            Courant ← courant↑.suivant
        Sinon
            Courant ↑.suivant ← nil
        Fsi
    Jusqu'à (Reponse == ′n′)
Fin
FinAlgorithme
```

Le contenu de la variable pointeur « tete » est fixe : elle contient la même valeur tout le long du programme, l'adresse du 1er nœud de la liste.

Le contenu de la variable pointeur « courant » change tout au long du programme : elle contient l'adresse du nœud en cours de traitement.

5. Opération sur les listes chaînées

5.2. Consultation (affichage) d'une liste chaînée simple

Pour consulter la liste chaînée que nous avons crée précédemment, il faut se repositionner au début (tête) de la liste, puis la parcourir nœud par nœud jusqu'à la fin.

```
Algorithme Consultation_Liste
    Variables :
        Tete, courant : pointeur
    Début
        Courant ← Tete
        Tant que (courant<>Nil) faire
            Ecrire (courant↑.numero)
            Ecrire (courant↑.nom)
            Courant ← courant↑.suivant
        Fin tant que
    Fin
FinAlgorithme
```

5. Opération sur les listes chaînées

5.3. Insertion d'un élément dans une liste chaînée

Dans une liste chaînée, on peut insérer autant d'éléments qu'on souhaite avec seulement la modification d'un ou deux pointeurs sans recopie ni décalage des éléments non par la mise à jour. L'insertion d'un élément peut se faire en tête, en queue ou dans une position k de la liste.

5.3.1. Insertion d'un élément en tête de liste chaînée

Insérer un nouvel élément en tête de la liste se fait en trois étapes :

a) On réserve un nouvel espace mémoire et on récupère son adresse dans un pointeur de type « Personne » appelé, par exemple « NouvElmt » ;
b) On saisit les informations véhiculées par le nouveau nœud ;
c) On effectue le chaînage du nouveau nœud avec la liste existante. Ce nœud devient le 1er élément de la nouvelle liste.

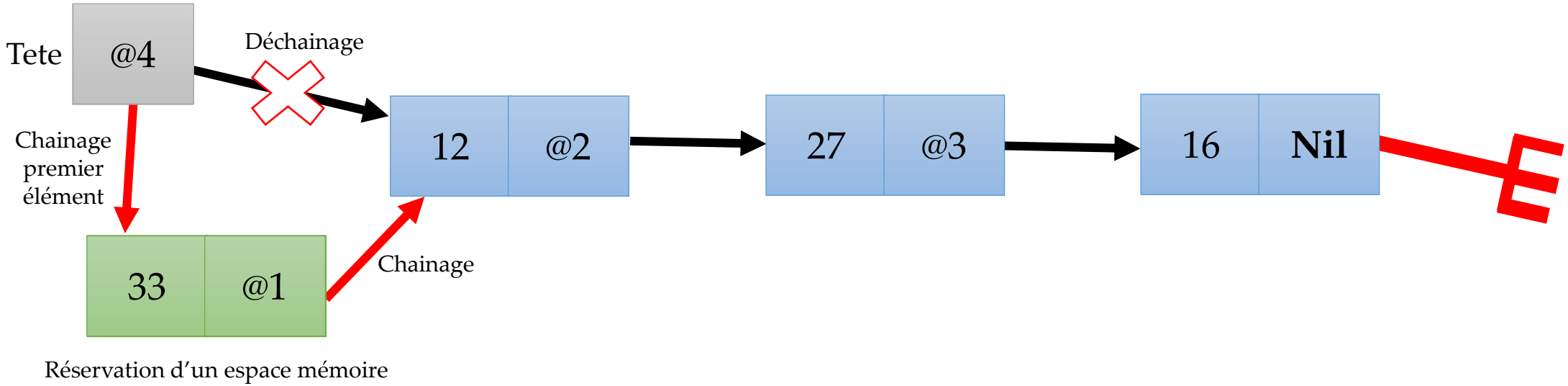
5. Opération sur les listes chaînées

5.3. Insertion d'un élément dans une liste chaînée

5.3.1. Insertion d'un élément en tête de liste chaînée

Simulation sur une liste d'entier :

On souhaite insérer un nouvel élément en tête de liste.



5. Opération sur les listes chaînées

5.3. Insertion d'un élément dans une liste chaînée

5.3.1. Insertion d'un élément en tête de liste chaînée

```
Algorithme Insertion_En_Tete_De_Liste
    Variables :
        Tete, NouvElmt : pointeur
    Début
        Nouveau (NouvElmt)
        Lire (NouvElmt ↑.numero)
        Lire (NouvElmt ↑.nom)
        NouvElmt↑.suivant ← Tete
        Tete ← NouvElmt
    Fin
FinAlgorithme
```

5. Opération sur les listes chaînées

5.3. Insertion d'un élément dans une liste chaînée

5.3.1. Insertion d'un élément en queue de liste chaînée

Insérer un nouvel élément en queue de la liste se fait en quatre étapes :

a) On réserve un nouvel espace mémoire et on récupère son adresse dans un pointeur de type « Personne » appelé, par exemple « NouvElmt » ;

b) On saisit les informations véhiculées par le nouveau nœud ;

c) On se positionne sur le dernier élément de la liste. Pour ce faire on la parcourt du début à la fin. Le dernier élément de la liste est celui pour lequel le « suivant » est égal à « Nil »

d) On effectue le chaînage du nouveau nœud avec la liste existante. Ce nœud devient le dernier élément de la nouvelle liste.

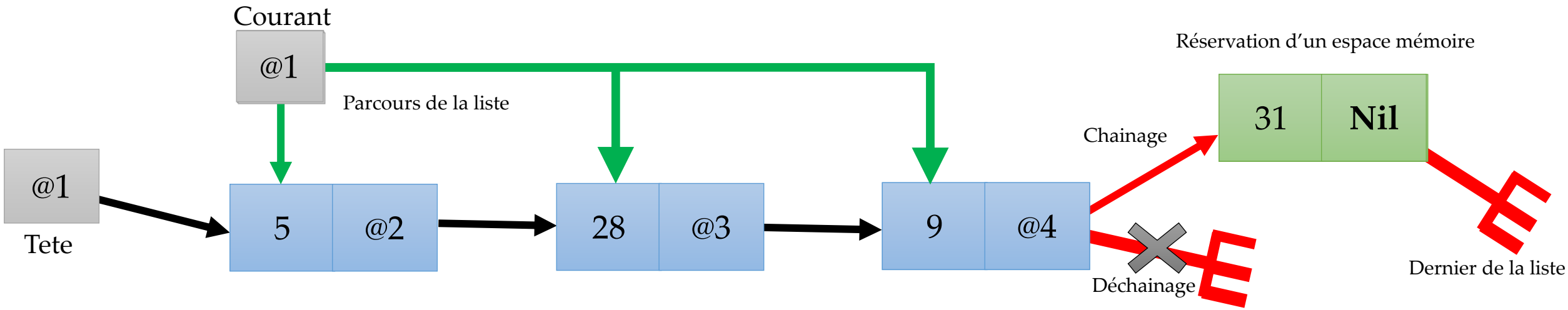
5. Opération sur les listes chaînées

5.3. Insertion d'un élément dans une liste chaînée

5.3.1. Insertion d'un élément en queue de liste chaînée

Simulation sur une liste d'entier :

On souhaite insérer un nouvel élément à la queue de liste.



5. Opération sur les listes chaînées

5.3. Insertion d'un élément dans une liste chaînée

5.3.1. Insertion d'un élément en queue de liste chaînée

```
Algorithme Insertion_En_Queue_De_Liste
    Variables :
        Tete, Courant, NouvElmt : pointeur
    Début
        Nouveau (NouvElmt)
        Lire (NouvElmt ↑.numero)
        Lire (NouvElmt ↑.nom)
        Courant ← Tete
        Tant que (Courant↑.suivant <>Nil) faire
            Courant ← Courant↑.suivant
        Fin tantque
        Courant↑.suivant ← NouvElmt
        NouvElmt↑.suivant ← Nil
    Fin
FinAlgorithme
```

5. Opération sur les listes chaînées

5.3. Insertion d'un élément dans une liste chaînée

5.3.1. Insertion d'un élément à une position k dans liste chaînée

Insérer un nouvel élément en queue de la liste se fait en quatre étapes :

a) On réserve un nouvel espace mémoire et on récupère son adresse dans un pointeur de type « Personne » appelé, par exemple « NouvElmt » ;

b) On saisit les informations véhiculées par le nouveau nœud ;

c) On effectue la recherche du nœud après lequel insérer le nouvel élément.

d) On effectue le chaînage du nouveau nœud avec la liste existante. Ce nœud prend sa place après celui recherché.

5. Opération sur les listes chaînées

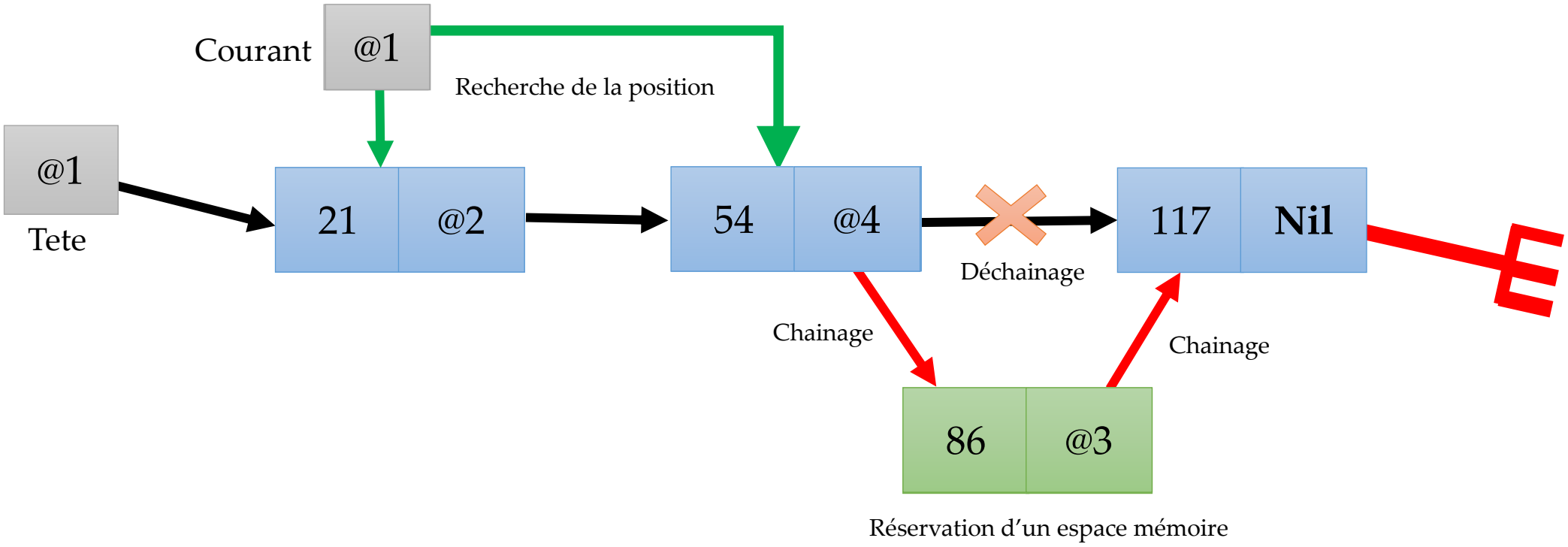
5.3. Insertion d'un élément dans une liste chaînée

5.3.1. Insertion d'un élément à une position k dans liste chaînée

Simulation sur une liste d'entier :

On souhaite insérer un nouvel élément à la position : 3

position



5. Opération sur les listes chaînées

5.3. Insertion d'un élément

5.3.1. Insertion d'un élément à un

```
Algorithme Insertion_Position_k_Dans_Liste
    Variables :
        Tete, Courant, NouvElmt : pointeur
        Num : entier

    Début
        Nouveau (NouvElmt)
        Lire (NouvElmt↑.numero)
        Lire (nom)
        Ecrire ("Entrer le numéro de la personne après lequel insérer")
        Lire (Num)
        Courant ← Tete
        Tant que (Courant <>Nil) faire
            Si (Num == Courant↑.numero ) alors
                NouvElmt↑.suivant ← Courant↑.suivant
                Courant↑.suivant ← NouvElmt
            Fsi
            Courant ← Courant↑.suivant
        Fin tant que
    Fin
FinAlgorithme
```

5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

Dans une liste chaînée, on peut supprimer un ou plusieurs éléments avec mise à jour d'un ou deux pointeurs et restitution des espaces mémoires occupés par ces éléments. La suppression d'un élément peut s'effectuer en tête, en queue ou à une position k donnée.

5.3.1. Suppression d'un élément en tête de liste chaînée

Supprimer le premier élément de la liste se fait en deux étapes :

a) On libère l'espace mémoire occupé par le premier nœud de la liste en utilisant la procédure « **Liberer** ».
b) On met à jour l'adresse de la nouvelle tête de la liste.

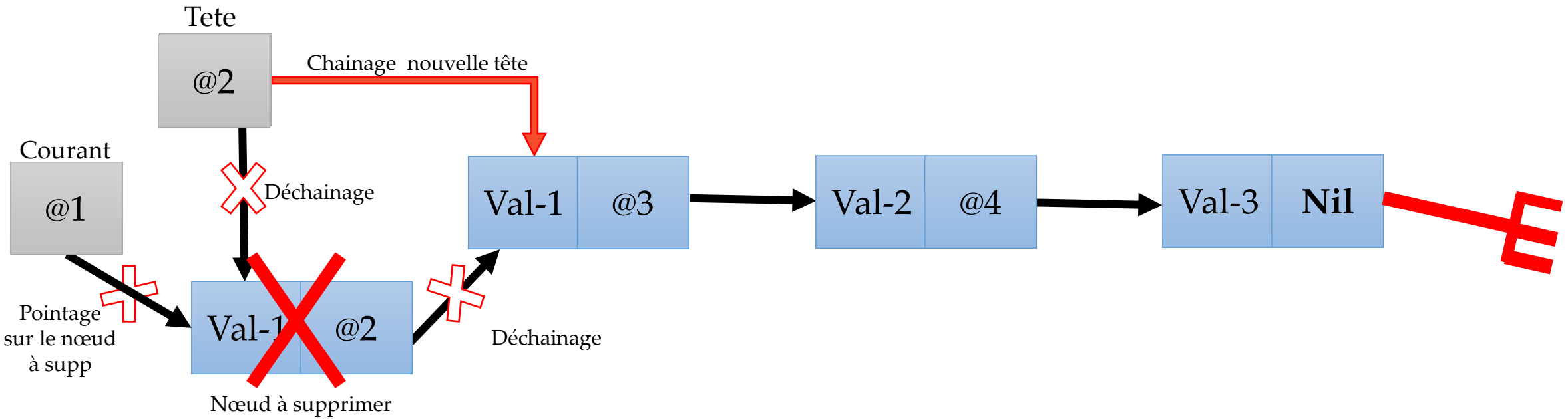
5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

5.3.1. Suppression d'un élément en tête de liste chaînée

Simulation sur une liste d'entier :

On souhaite supprimer un élément à la tête de la liste.



5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

5.3.1. Suppression d'un élément en tête de liste chaînée

```
Algorithme Suppression_En_Tete_De_Liste
    Variables :
        Tete, Courant : pointeur
    Début
        Courant ← Tete
        Tete ← Tete↑.suivant
        Liberer (Courant)
    Fin
FinAlgorithme
```

5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

5.3.1. Suppression d'un élément en queue de liste chaînée

La suppression du dernier élément d'une liste se fait en trois étapes :

a) On se positionne sur l'avant dernier élément de la liste. Pour ce faire on la parcourt à partir du début, élément par élément jusqu'à trouver l'avant dernier. L'avant dernier élément d'une liste est celui dont le pointeur « suivant » du « suivant » est égal à « Nil ».
b) On libère l'espace mémoire occupé par le dernier élément de la liste (qui se trouve être le suivant de l'élément courant).
c) On fait en sorte que l'avant dernier élément de la liste devienne le dernier.

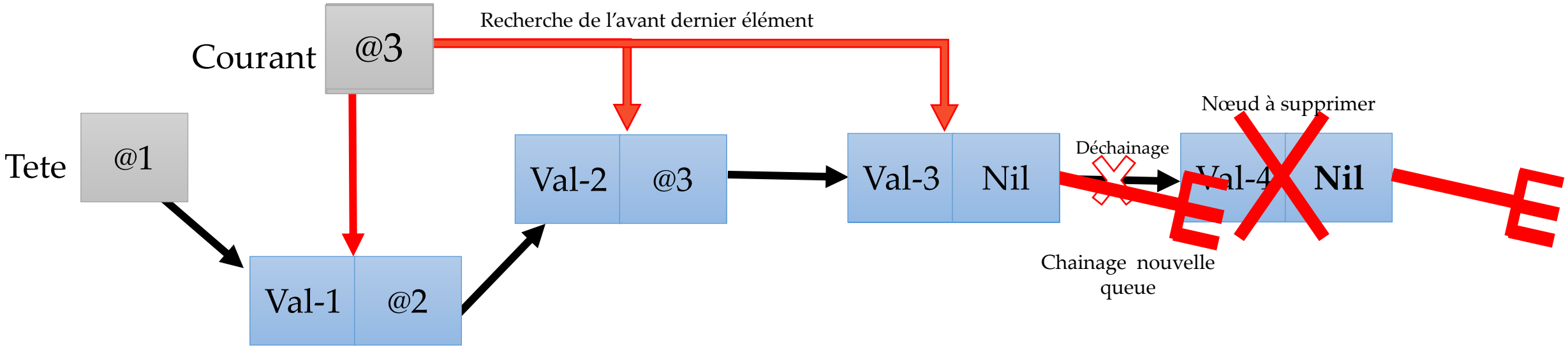
5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

5.3.1. Suppression d'un élément en queue de liste chaînée

Simulation sur une liste d'entier :

On souhaite supprimer un élément en queue de la liste.



5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

5.3.1. Suppression d'un élément en queue de liste chaînée

```
Algorithme Suppression_En_Queue_De_Liste
    Variables :
        Tete, Courant : pointeur
    Début
        Courant ←Tete
        Tant que (Courant↑.suivant↑.suivant <>Nil) faire
            Courant ← Courant↑.suivant
        Fin tantque
        Liberer (Courant↑.suivant)
        Courant↑.suivant ← Nil
    Fin
FinAlgorithme
```

5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

5.3.1. Suppression d'un élément à une position k dans liste chaînée

La suppression d'un élément dans une position donnée dans une liste se fait en trois étapes :

a) On effectue la recherche du nœud à supprimer. On peut faire la recherche avec le numéro de la personne en commençant par le 2nd élément de la liste.
b) On libère l'espace occuper par l'élément à supprimer.
c) On met à jour le pointeur « suivant » de l'élément **précédent** celui à supprimer.

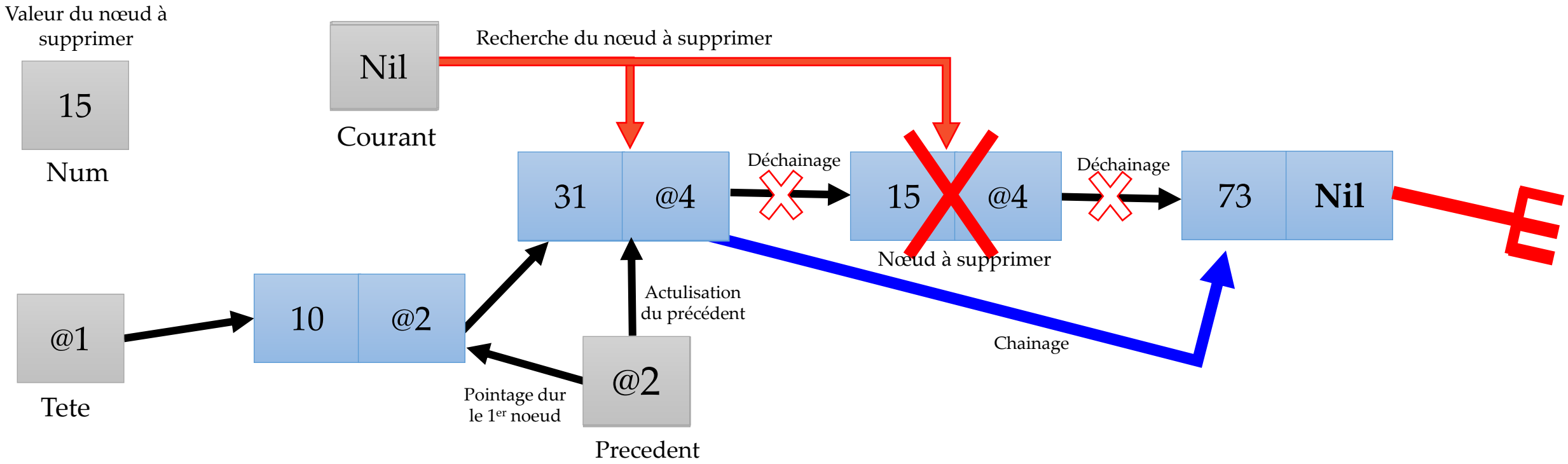
5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

5.3.1. Suppression d'un élément à une position k dans liste chaînée

Simulation sur une liste d'entier :

On souhaite supprimer un élément à une position donnée dans la liste.



5. Opération sur les listes chaînées

5.3. Suppression d'un élément dans une liste chaînée

5.3.1. Suppression d'un élément à une position k dans liste chaînée

```
Algorithme Suppression_ Position_k_Dans_Liste
    Variables :
        Tete, Courant, Precedent : pointeur
        Num :entier
    Début
        Ecrire ("Entrer le numéro de la personne  a supprimer")
        Lire (Num)
        Precedent ←Tete
        Courant ←Tete↑.suivant
        Tant que (Courant <>Nil) faire
            Si (Num  = = Courant↑.numero) alors
                Precedent↑.suivant ← Courant↑.suivant
                Liberer (Courant)
                Courant ← nil    // pour quitter la boucle
            Sinon
                Precedent ← Courant
                Courant ← Courant↑.suivant
            Fsi
        Fin tantque
    Fin
FinAlgorithme
```

6. Les listes chaînées particulières

6.1. Listes chaînées bidirectionnelles (listes doublement chaînées)

Les listes chaînées que nous avons précédemment peuvent être qualifier de simples ou monodirectionnelles car on ne peut les parcourir que dans un seul sens : de gauche à droite.

Si on veut pouvoir effectuer un parcours de droite à gauche, il faut ajouter un pointeur permettant l'accès au nœud précèdent. On qualifie alors la liste de **bidirectionnelle**. Compte tenu de la place supplémentaire occupée en mémoire. On utilise cette possibilité que dans le cas où on a très souvent besoin d'effectuer un retour en arrière vers la tête de la liste.

Pour faciliter le parcours de droite à gauche, on mémorise l'adresse du dernier nœud dans une variable pointeur que nous appellerons très souvent « **fin** ».

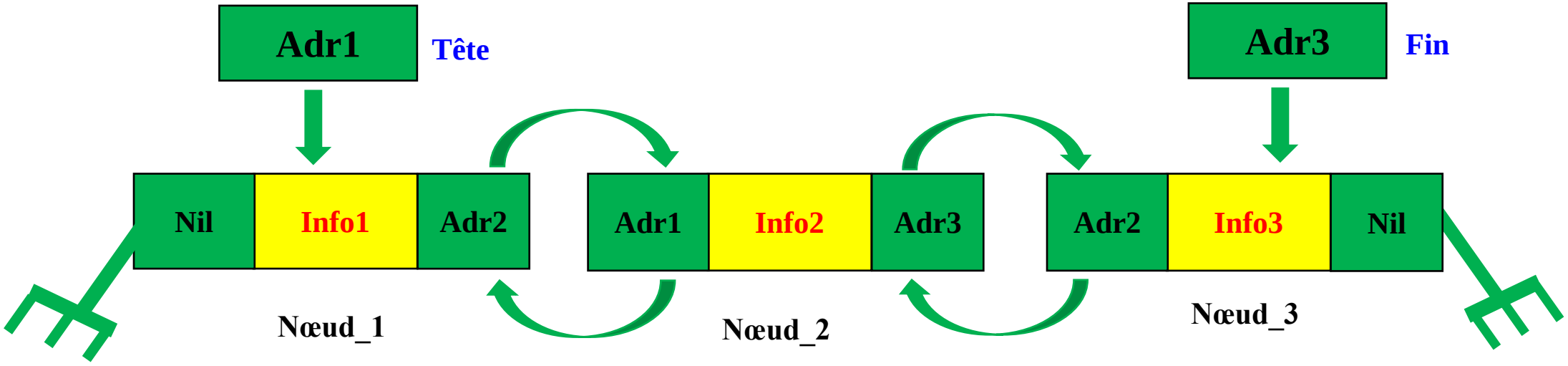
6. Les listes chaînées particulières

6.1. Listes chaînées bidirectionnelles (listes doublement chaînées)

- **Représentation graphique d'un nœud :**

Nœud

- **Représentation graphique d'une liste chaînée bidirectionnelle :**



6. Les listes chaînées particulières

6.1. Listes chaînées bidirectionnelles (listes doublement chaînées)

- **Syntaxe de déclaration d'un nœud :**

```
TYPE pointeur = ↑ nœud
    nœud = structure
        membre_1 : Type1
        membre_2 : Type2
            .....
        membre_n : TypeN
        Suivant, Precedent : pointeur
    Fin_structure
```

6. Les listes chaînées particulières

6.1. Listes chaînées bidirectionnelles (listes doublement chaînées)

6.1.1. Création d'une listes chaînées bidirectionnelles

- Le principe reste assez similaire à celui d'une liste simple, sauf qu'il faut gérer le pointeur « **precedent** » de chaque nœud créer.
- Le pointeur « **precedent** » du premier nœud ainsi que le pointeur « **suivant** » du dernier nœud contiennent la valeur « **Nil** ».

6. Les listes chaînées particulières

6.1. Listes chaînées bidirectionnelles (listes doublement chaînées)

6.1.1. Création d'une listes chaînées bidirectionnelles

```
Algorithme Création_Liste_Bidirectionnelle
    TYPE pointeur = ↑ Personne
        Personne = structure
            Numéro : entier
            Nom : chaîne de 20 caractères
            suivant, precedent : pointeur
        Fin_structure
    Variables :
        Tete, courant, fin : pointeur
        Reponse : Car
    Début
        Nouveau(Tete)
        courant ← Tete
        courant↑.suivant ← Nil
            Répéter
            Ecrire (Entrez le numéro et le nom de la personne)
            Lire (courant↑.numero, courant↑.nom)
            Ecrire(Autre élément o/n ?)
            Lire(Reponse)
            Si (Reponse == 'o') alors
                Nouveau(courant↑.suivant)
                courant↑.suivant↑.precedent ← courant
                courant ← courant↑.suivant
            Sinon
                Courant ↑.suivant ← nil
                Fin ← courant
            FinSi
        Jusqu'à Reponse = = 'n'
    Fin
FinAlgorithme
```

6. Les listes chaînées particulières

6.1. Listes chaînées bidirectionnelles (listes doublen

6.1.2. Consultation d'une listes chaînées bidirect

Une fois la liste doublement chaînée créée, on peut la parcourir de la **tête à la fin** (gauche-droite) ou de la **fin à la tête** (droite-gauche). Le principe est le même que pour une liste chaînée simple.

```
Algorithme Consultation_Liste_Bidirectionnelle
    Variables :
                    tete, courant, fin : pointeur
                    choix : entier
    Début
        Ecrire ("1-Consultation par la gauche   2-Consultation par la droite ")
        Ecrire ("Entrez votre choix : ")
        Lire (choix)
        Si (choix == 1) alors
            courant ← tete
            Tant que (courant<>Nil) faire
                Ecrire (courant↑.numero, "   ", courant↑.nom)
                Courant ← courant↑.suivant
            Fin tantque
        SinonSi (choix == 2) alors
            courant ← fin
            Tant que (courant<>Nil) faire
                Ecrire (courant↑.numero, "   ", courant↑.nom)
                Courant ← courant↑.precedent
            Fin tant que
        FinSi
    Fin
FinAlgorithme
```

6. Les listes chaînées particulières

6.1. Listes chaînées bidirectionnelles (listes doublement chaînées)

6.1.3. Insertion d'un élément en tête d'une listes chaînées bidirectionnelles

L'insertion en tête de la liste d'un nouvel élément se fait comme suit :

a) On réserve un nouvel espace mémoire à l'adresse « NouvElmt » ;
b) On saisit les informations véhiculées par le nouveau nœud ;
c) On initialise le pointeur « precedent » du nouveau nœud à Nil.
d) On effectue le chaînage du nouveau nœud avec la liste existante.
e) On met à jour l'adresse de la nouvelle tete de liste.

```
Algorithme Insertion_En_Tete_De_Liste_Bidrectionnelle
    Variables :
        tete, NouvElmt : pointeur
    Début
        Nouveau (NouvElmt)
        Lire (NouvElmt ↑.numero, NouvElmt ↑.nom)
        NouvElmt↑.precedent ← Nil
        tete↑.suivant ← NouvElmt
        NouvElmt↑.suivant ← tete
        tete ← NouvElmt
    Fin
FinAlgorithme
```

6. Les listes chaînées particulières

6.1. Listes chaînées bidirectionnelles (listes doublement chaînées)

6.1.3. Suppression d'un élément en tête d'une listes chaînées bidirectionnelles

La suppression du 1er élément de la liste se fait comme suit :

a) On libère l'espace mémoire occupé par le 1er nœud.
b) On met à jour l'adresse de la nouvelle tête de la liste.
c) La nouvelle tête de liste n'a pas de précédent.

```
Algorithme Suppression_En_Queue_De_Liste_Bidirectionnelle
    Variables :
        tete : pointeur
    Début
        Liberer (tete)
        Tete ← tete↑. suivant
        tete↑.precedent ← Nil
    Fin
FinAlgorithme
```

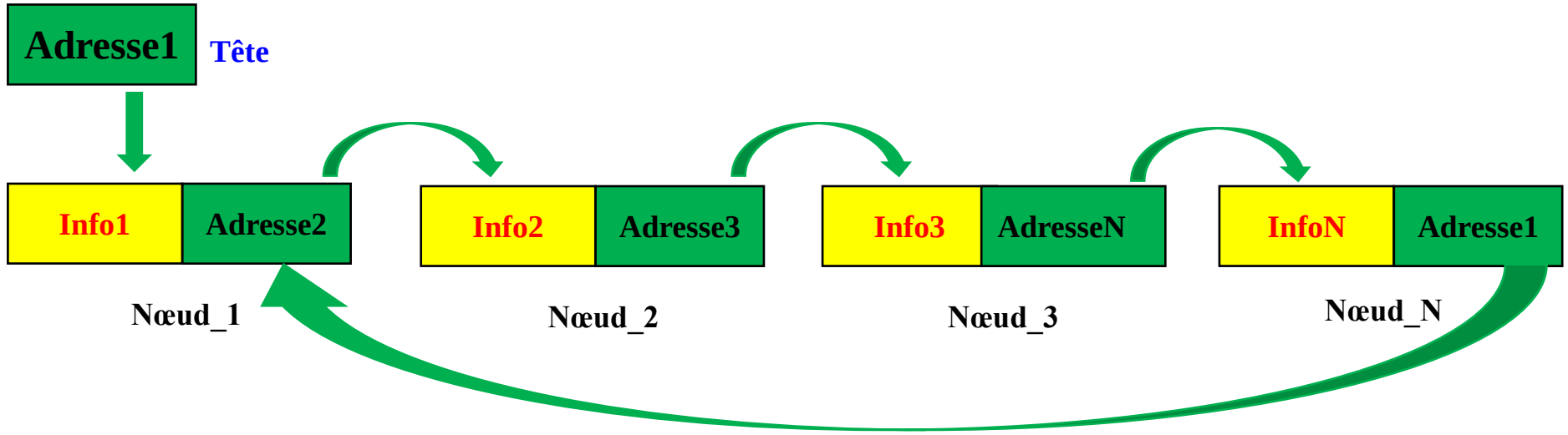
6. Les listes chaînées particulières

6.2. Listes chaînées circulaires

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

Dans une liste circulaire tous les maillons sont accessibles à partir de n'importe quel autre maillon. Une liste circulaire n'a pas de premier et de dernier maillon. Par convention, on peut prendre le pointeur externe de la liste vers le dernier élément et le suivant serait le premier élément de la liste.

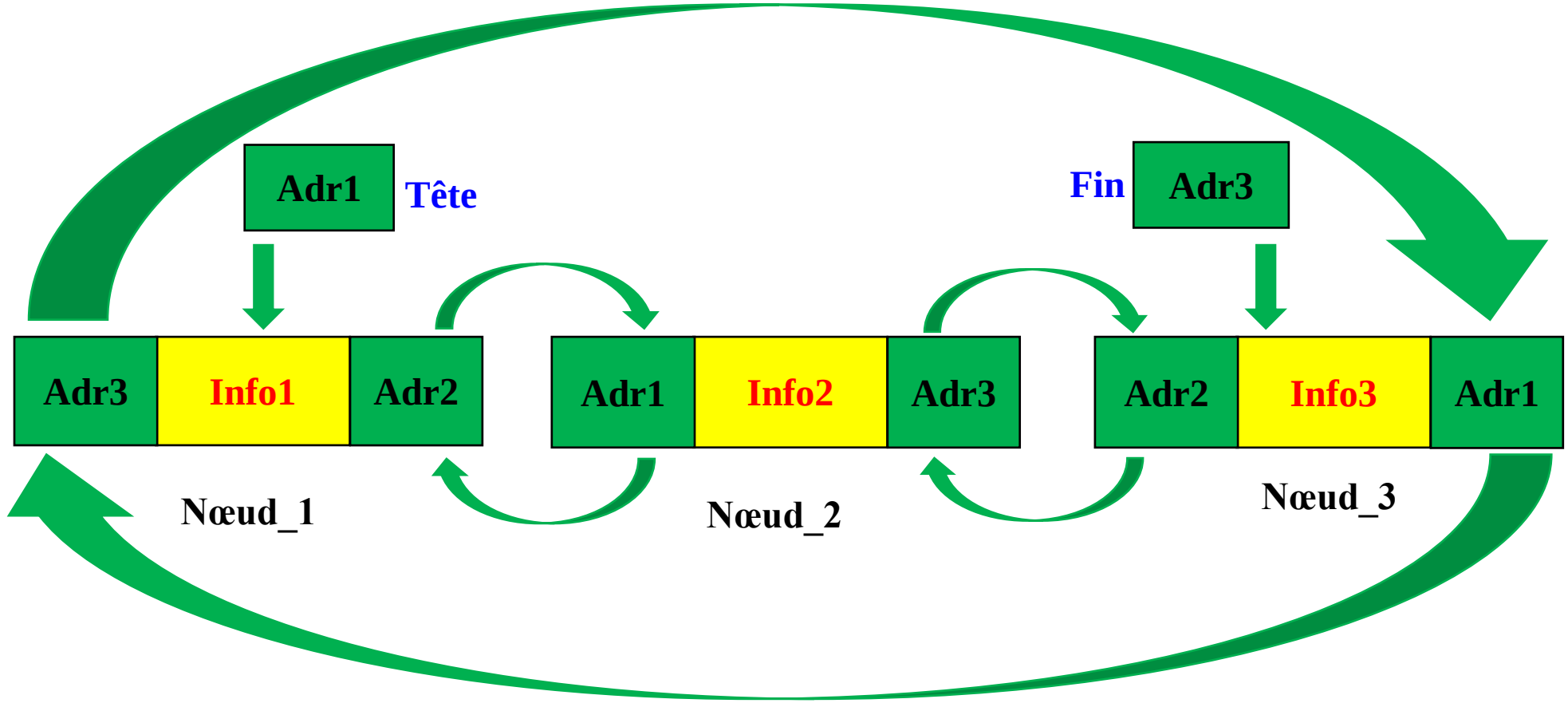
Ci-dessous l'exemple d'une **liste simplement chaînée circulaire** : le dernier élément pointe sur le premier.



6. Les listes chaînées particulières

6.2. Listes chaînées circulaires

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



7. Conclusion

L'avantage des listes chaînées est donc de ne plus utiliser des tableaux de pointeurs (**Tableau dynamique**) pour pointer les éléments instanciés. Chaque élément est simplement relié au suivant par un **pointeur**, voire également au précédent si nécessaire par un second pointeur.

Car les tableaux de pointeurs déclarés grâce à l'opérateur **new** sont toujours de longueur fixe.