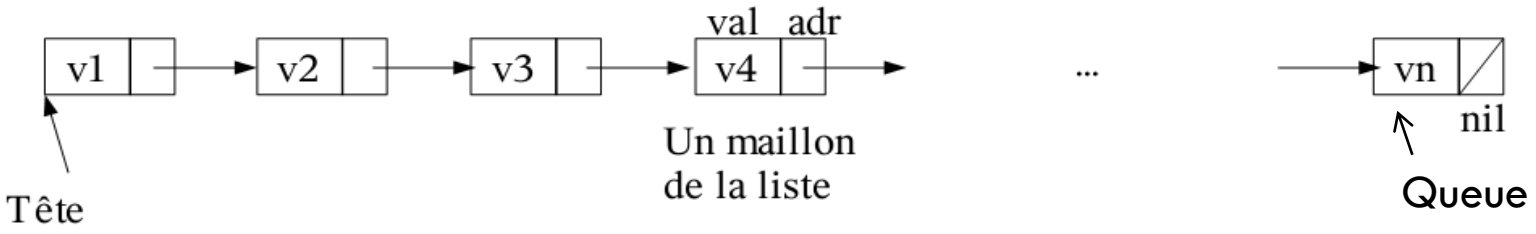
Chapitre 4 :
Les listes linéaires chainées

Liste linéaire chainée [1]

- ***Définition:*** une ***liste linéaire chainée*** est constituée d'un ensemble de cellules (généralement des vars dynamique) **chaînées** entre elles.
- C'est **l'adresse de la première** de ces cellules qui **détermine la liste**.
- Cette adresse doit être stockée dans une variable appelée souvent : **liste**, **tête**…
- **Propriétés**:
  - Chaque élément est relié à son successeur par un pointeur.
  - Le premier élément est appelé **Tête** & le dernier **Queue**
  - La valeur du pointeur du **dernier** élément est **Nil**.

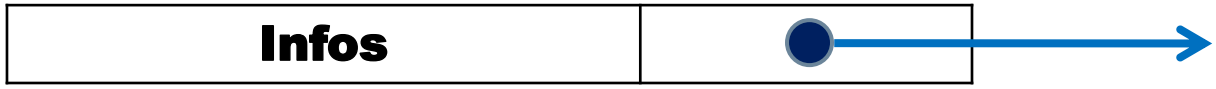


Liste linéaire chainée [2]

- **Mais pourquoi les LLC ?**
- C'est une alternative aux vecteurs (Tab à 1 Dimension)
  - Où on a à réserver à l'avance la taille (espace mémoire) à utiliser
    - **Exemple:** `var tab: array [1..1000] of etudiant;`
  - On réserve le max et on ne sait pas combien de cases seront réellement utilisées.
  - Si on aura besoin de plus de cases; on a à modifier le code et réserver d'avantage de cases.
- **Solution**: les **LLC**
  - Variables à taille dynamique (**+** / **-**)
  - Espace mémoire **réservé** = Espace mémoire **utilisé**.

Définition d'une Liste chainée [1]

- La définition d'une **liste linéaire chainée** est très semblable aux définitions des structures de données de type enregistrement.
- La définition d'une **LLC** peut être décomposée en deux (**02**) parties:
  - **Partie champs d'un élément de LLC (Info):** qui détermine le type des éléments de **LLC** (**Ex:** liste d'entiers, d'étudiants …).
  - **Partie pointeur d'élément (Suivant):** contenant un champ adresse (pointeur) de l'élément suivant dans la liste.



- **Remarque:** **Chaque élément dans la liste est un enregistrement.**

Définition d'une Liste chainée [3]

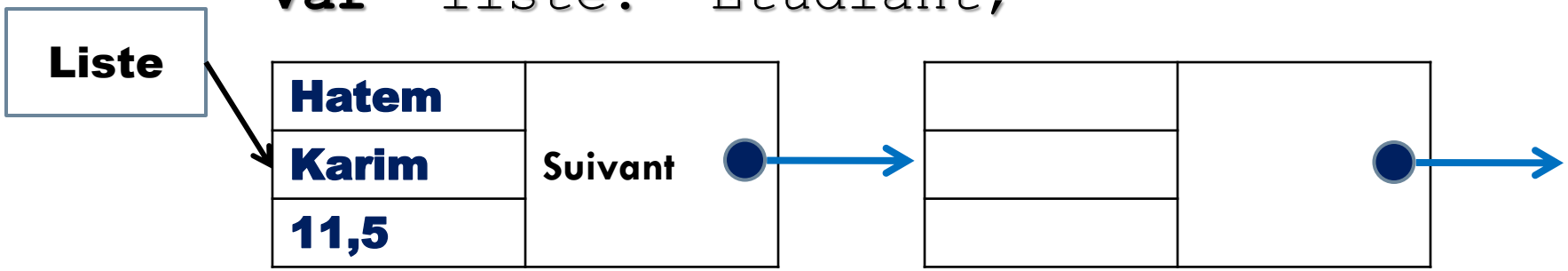
- En pascal:
- Exemple1: (LLC d'entiers)

```
Type liste_entier = record
        nbr: integer;
        suivant: ^liste_entier;
End;
Var  liste: ^liste_entier;
```

Définition d'une Liste chainée [4]

- En pascal:
- Exemple2: (LLC d'étudiants)

```
Type Etudiant = record
        nom, prenom: string[20];
        Moyenne: real;
        suivant: ^Etudiant;
End;
Var  liste: ^Etudiant;
```



Le modèle des LLC

Algorithmes de Création et de manipulation des LLC

Primitives d'accès à une LLC

- **Les primitives d'accès à une LLC sont:**

- Le prédicat **VIDE** qui teste la **vacuité** d'une LLC
- La fonction **PREMIER** qui fournit **le premier élément** de LLC (**Tête**)
- La procédure **INSERER** pour **l'ajout d'un élément** à LLC
- La procédure **SUPPRIMER** pour **le retrait d'un élément** de LLC

- D'autres procédures et fonctions peuvent être ajoutees à celles-ci, ça dépendra du cas;
  - **Exemple**: AFFICHER (P) …

Le prédicat « VIDE »

- Le prédicat **VIDE:** prend deux valeurs « **vrai** » ou « **faux** ».
- Si le pointeur de tête de liste pointe sur un élément de la liste;
  - Vide ← **faux**
- Si le pointeur de tête de liste pointe sur **NIL**
  - Vide ← **vrai**
- **Implémentation**

```
Vide (TETE)
VIDE ← TETE=NIL
```

  - **Ou**

```
Si (TETE = NIL) alors VIDE ← Vrai
sinon VIDE ← Faux
```

La primitive « INSERER » - (1)

- Dans tous les cas, l'***insertion*** se fait en **2 temps** :

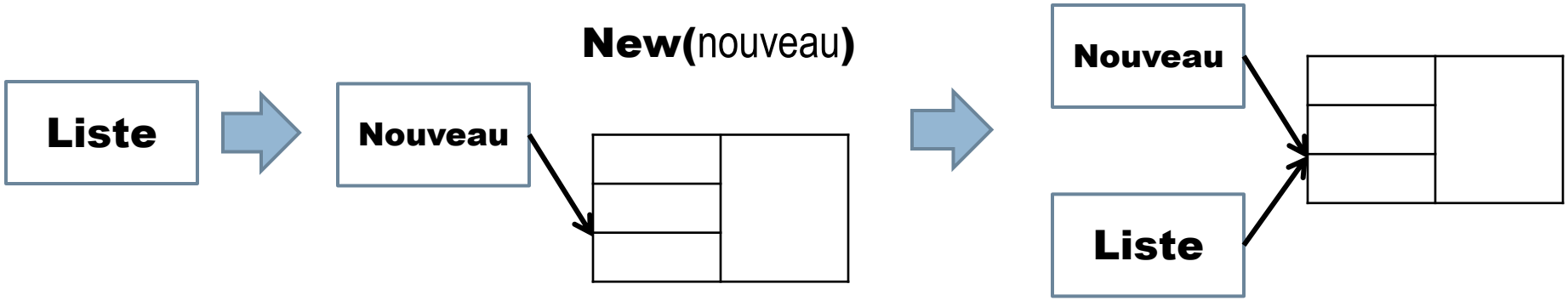
1. On ***crée une nouvelle cellule*** avec l'instruction **new(Nouveau);** et on initialise le champ valeur de cette cellule avec la valeur voulue.

2. On la **place dans la liste**, pour cela il faut distinguer « **plusieurs cas** ».

La primitive « INSERER » - (2)

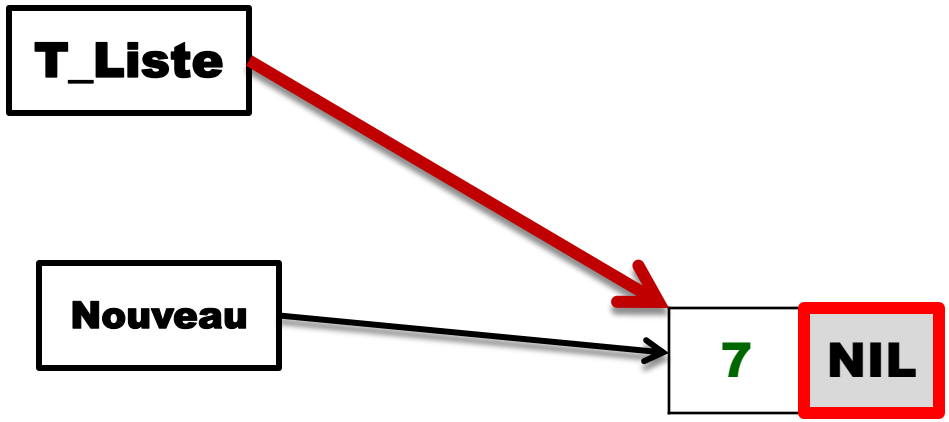
a) **Première insertion:** Si la liste est **vide**, le pointeur qui la désigne contient la valeur **NIL** : il suffit donc de lui affecter « **Nouveau** ».

```
Liste_Chainee := Nouveau;
```



Remarque: puisque nouveau est en même temps le premier et le dernier élément, nouveau^.suivant:=nil;

Exemple – 1ère Insertion



PRG 1ère insertion

Exemple ETUDIANT

```
Type  etudiant = record
    Num: integer;
    Nom: string[20];
    Suivant: ^etudiant;
End;
Var tete, nouveau: ^etudiant;
Begin
    Tete:=nil;
    New (nouveau);
    nouveau^.suivant:=nil;
    Tete:=nouveau;
    Nouveau:=nil;
```

La primitive « INSERER » - (3)

b) **Insertion en tête de la liste:** Il s'agit ici de modifier le pointeur de tête de liste et d'établir un lien entre la nouvelle cellule et celle qui se trouvait en tête :

{ On établit le lien }

```
Nouveau^.Suivant := Liste_Chainee;
```

{ On modifie le pointeur de tête }

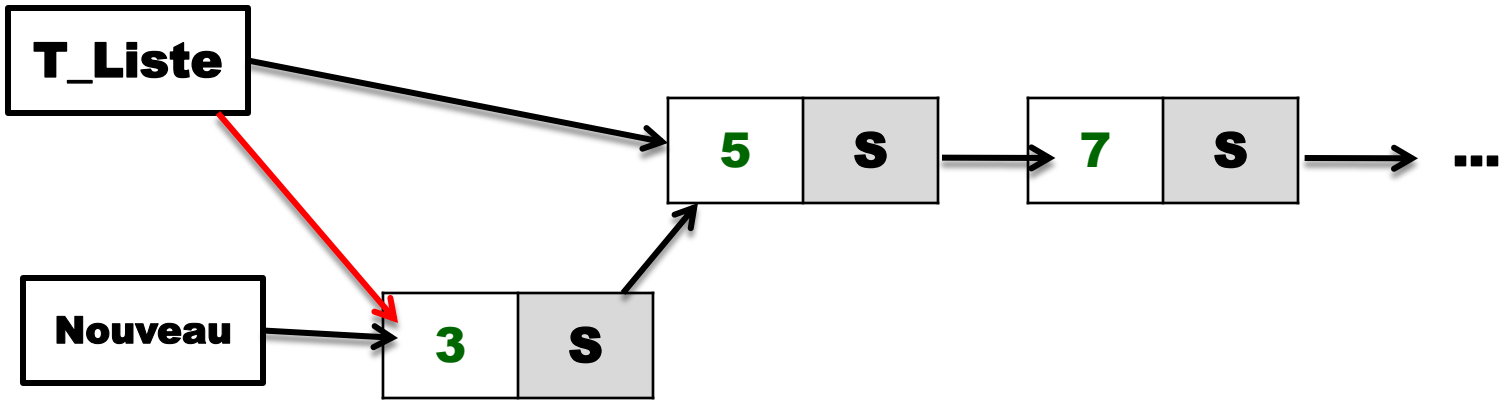
```
Liste_Chainee := Nouveau;
```

PRG insertion en tête de liste

Exemple ETUDIANT

```
New(nouveau);

Nouveau^.ch1:=23;

Nouveau^.suivant:=tete_liste;

Tete_liste:=nouveau;

Nouveau:=NIL;
```

Exemple – Insertion tête



La primitive « INSERER » - (4)

c) **Insertion au milieu de la liste:**

Dans une liste simplement chaînée, le sens de parcours est toujours d'une cellule vers sa suivante : on ne peut revenir en arrière. Pour insérer entre deux cellules, **on insèrera** donc toujours **après une cellule donnée**.

La primitive « INSERER » - (5)

Insertion au milieu (Suite)

Soit **`Courant`** la cellule **après** laquelle **on insère** :

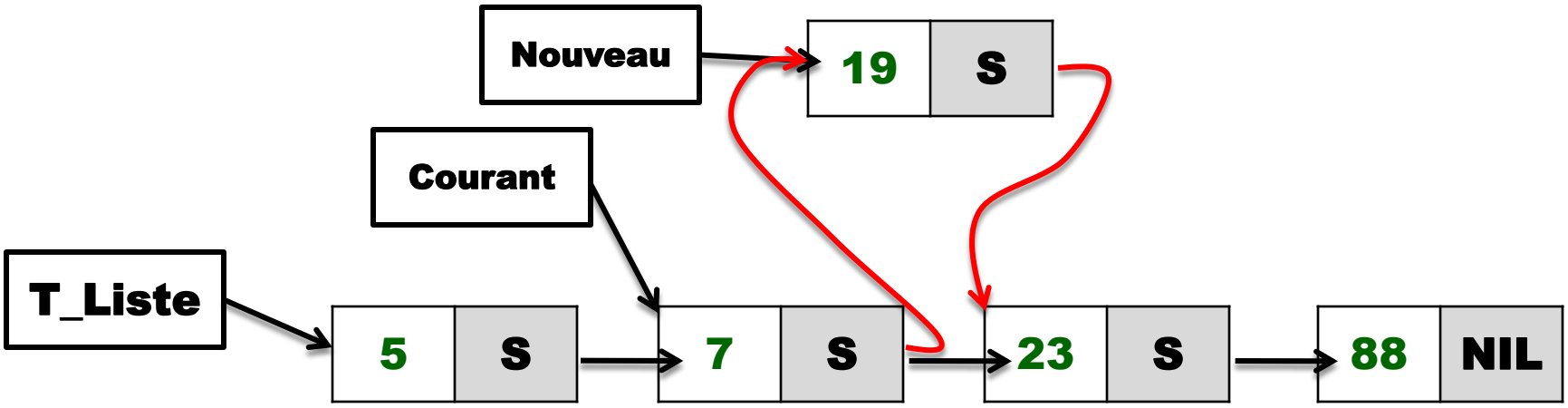
- on mémorise d'abord la cellule suivante de Courant dans le champ Suivant de Nouveau :

```
Nouveau^.Suivant := Courant^.Suivant;
```

- on peut donc maintenant faire pointer le champ Suivant de Courant sur Nouveau :

```
Courant^.Suivant := Nouveau;
```

Exemple – Insertion Milieu



La primitive « INSERER » - (6)

c) **Insertion en FIN de liste:**

Il s'agit d'un **cas particulier** du précédent : insérer en fin de liste, c'est ***insérer après la dernière cellule***. Dans ce cas, le champ **`Suivant`** de **`Courant`** vaut **NIL**.

Après insertion, le champ **`Suivant`** de **`Nouveau`** vaudra **NIL** et celui de **`Courant`** pointera sur **`Nouveau`**.

PRG insertion en fin de liste

Exemple ETUDIANT

La suppression- (1)

Dans tous les cas, on suppose que **`Temp`** ***pointe*** sur la cellule à **supprimer**.

Le sens de parcours étant **unidirectionnel** : on ***supprimera*** donc ***toujours*** **`Temp`** en utilisant la cellule qui la précède (**`Courant`**), sauf s'il s'agit de la première cellule de la liste.

Remarque : Comme pour l'insertion, on traitera à part la suppression de la première cellule car c'est le seul cas où on modifie directement **`Liste_Chainee (Tête)`**.

La suppression- (2)

a) **Suppression en tête de liste**

On libère le premier emplacement ***après avoir mémorisé le suivant*** :

① `Temp := Liste_Chainee^.suivant;`

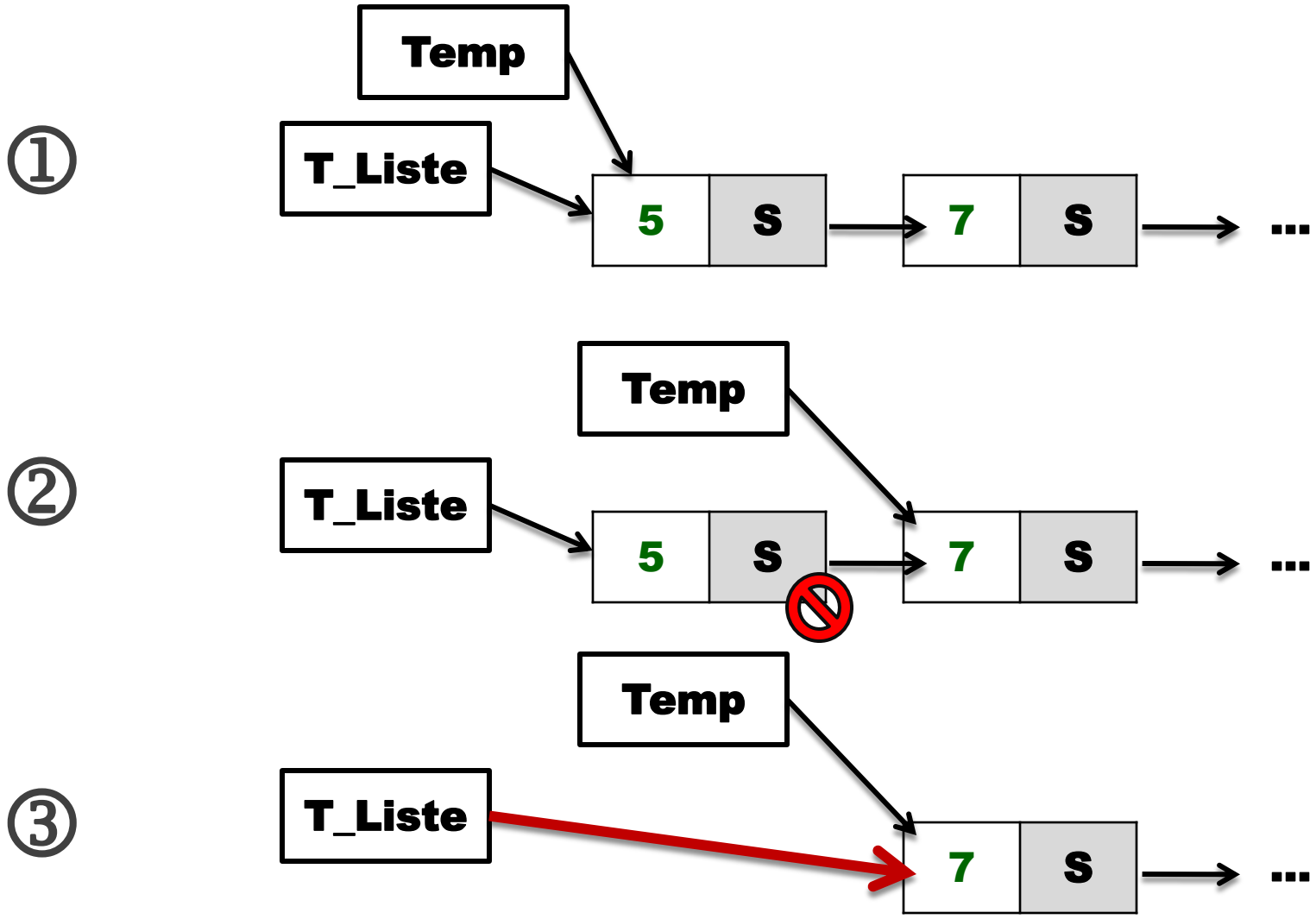
② `dispose(Liste_Chainee);`

③ `Liste_Chainee := Temp;`

Remarque : On notera que ce cas traite la situation où **`Liste_Chainee`** est constituée **d'une seule cellule**, **`Temp`** vaudrait alors **nil**.

Après la suppression, **`Liste_Chainee`** vaudrait également **nil** : **la liste serait** considérée comme **vide**.

Exemple – Suppression tête



La suppression- (3)

b) **Suppression au milieu de la liste**

① On **mémorise l'emplacement** de la cellule à ***supprimer*** :

```
Temp := Courant^.Suivant;
```

② On **établit le lien** entre **`Courant`** et la ***cellule suivante*** de celle que l'on va ***supprimer*** :

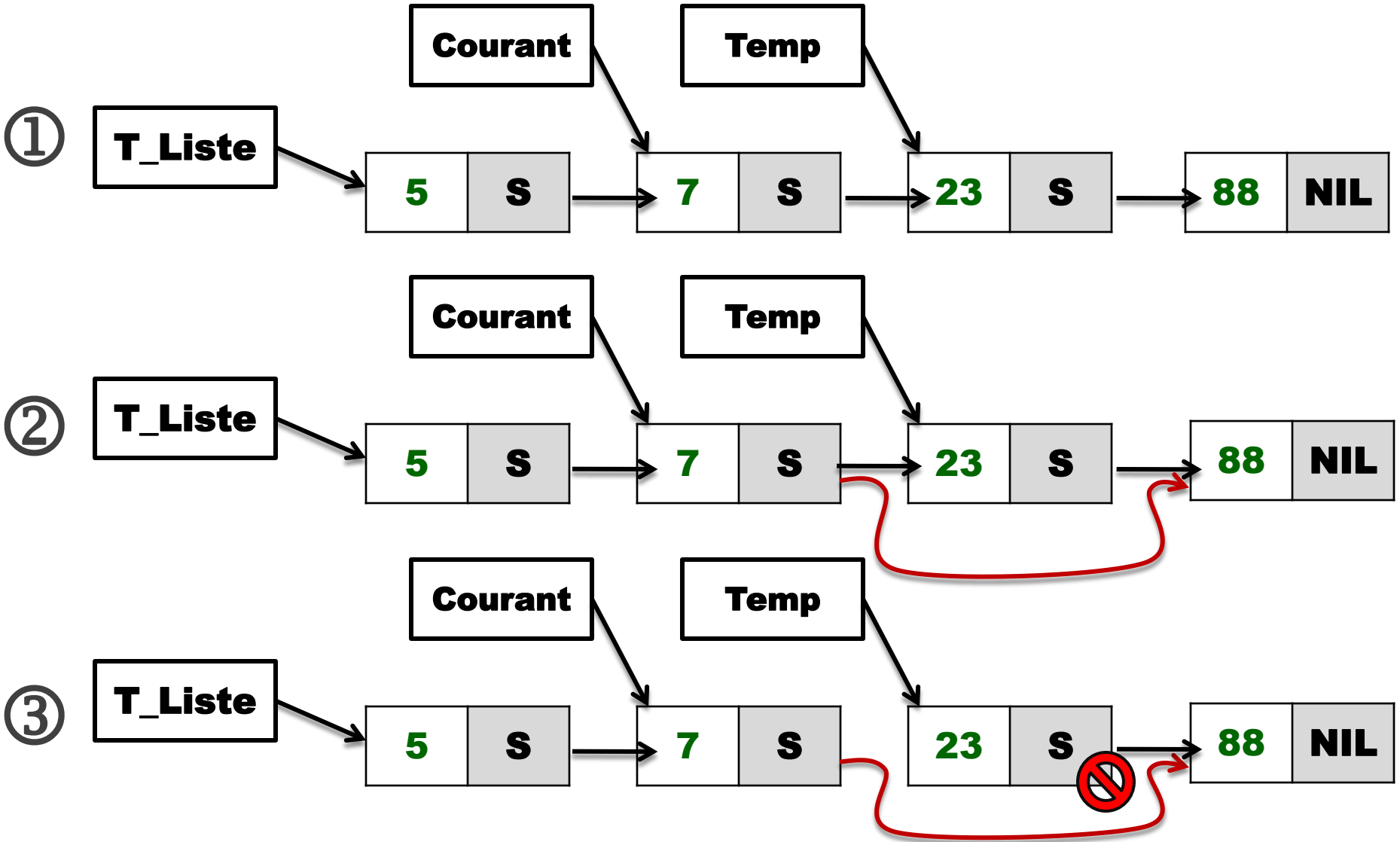
```
Courant^.Suivant := Temp^.Suivant;
```

{**ou** `Courant^.Suivant := Courant^.Suivant^.Suivant`}

③ **On libère l'espace alloué** à la cellule à supprimer :

```
dispose(Temp);
{impossible si on a pas mémorisé Temp}
```

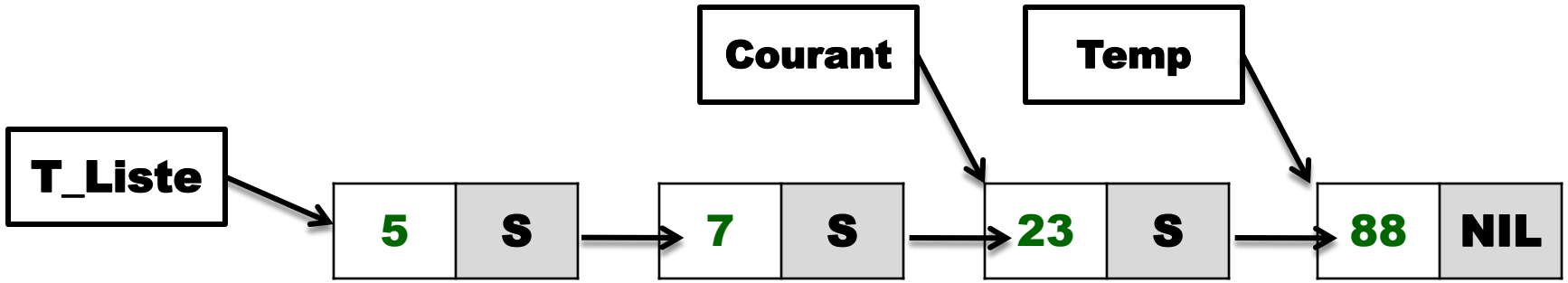
Exemple – Suppression milieu



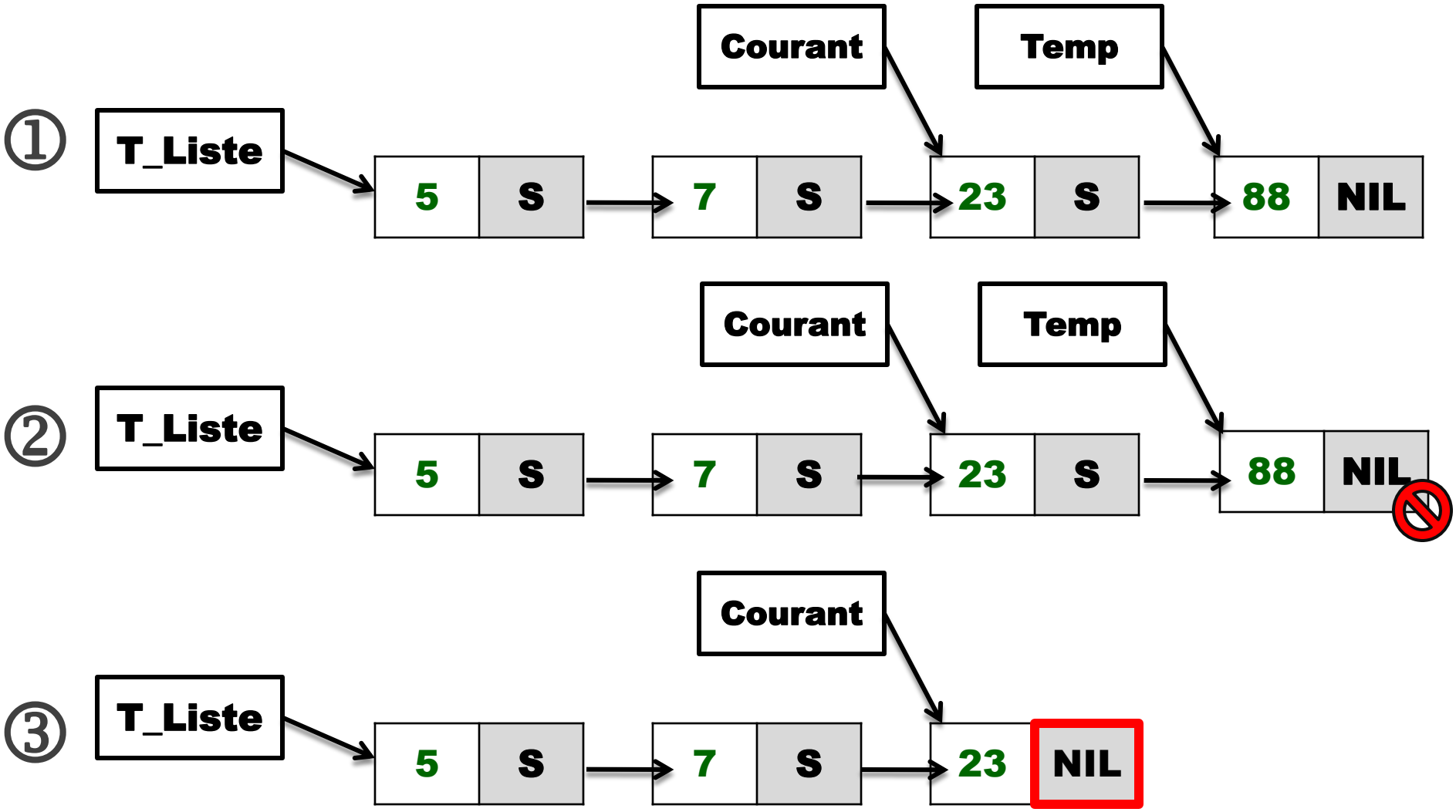
La suppression- (4)

c) Suppression en FIN de liste

Ici, `Temp^.Suivant = nil;` après suppression, `Courant^.Suivant` vaudra donc `nil`, ce qui indiquera que Courant sera la dernière cellule de la liste.



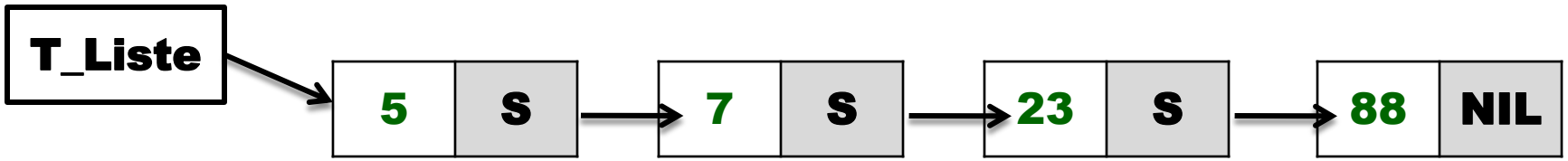
Exemple – Suppression FIN



La suppression- (4)

d) Suppression d'une liste entière

La suppression d'une liste complète passe par la suppression de tous ses éléments, **dans le bon ordre**. À l'issue de cette opération, la liste doit être vide **et** l'ensemble **de l'espace mémoire qu'elle occupait «est libéré »**.



Exercice 3 : Ecrire les procédures et les fonctions (en Pascal) sur les Llc :

1. Construire une liste chainée à partir de n données lues (Ex : liste d'entiers).
2. Longueur d'une Llc.
3. Rechercher dans une Llc l'élément qui a le plus grand nombre d'occurrences.
4. Accès par valeur dans une Llc
5. Accès par position dans une Llc
6. Suppression par valeur dans Llc
7. Insertion par position dans Llc
8. Trier une Llc par la méthode des Bulles

Autres opérations (1)

- Afin d'afficher la partie ***INFO*** d'un des éléments d'une LLC, on peut utiliser une procédure **AFFICHER (P).**
- La procédure permettra d'afficher la partie ***INFO*** d'***un élément pointé par P***.

```
Procedure AFFICHER (P:^type);
Begin
    Writeln (P^.Info);
End;
```

Autres opérations (2)

- Afin de pointer sur le premier élément d'une LLC, on peut utiliser une procédure **PREMIER (P).**

```
Procedure PREMIER (var P:^type);
Begin
    P:=TETE;
End;
```

Autres opérations (3)

- Afin de ***pointer sur le Dernier élément*** d'une LLC, on peut utiliser une procédure **DERNIER (P).**

```
Procedure DERNIER (P:^type);
Begin
  If (TETE <> NIL) then
  begin
  P:=TETE;
  While (p^.suivant <> NIL) do
      p:= p^.suivant;
  End
  else Writeln ('Liste vide !');
End;
```

Autres opérations (4)

- On peut aussi utiliser une procédure RECHERCHE qui permet de chercher une valeur dans une LLC.

```
Procedure RECHERCHE (X:type_val, TETE :^type, P:^type);
VAR trouve:boolean;
Begin
Trouve:=false;
   If (TETE <> NIL) then
   begin
      P:=TETE;
      While (P <> NIL) and (trouve = false) do
         if (P^.Info = X) then trouve := true
         else p:= p^.suivant;
      if trouve = false then writeln ('Aucun élément correspondant')
      else writeln ('Trouvé & pointé par P !');
   End
   else Writeln ('Liste vide !');
End;
```

Autres opérations (4)

- Compter le nombre d'éléments dans une LLC.
- Accéder au $K^{ième}$ élément de la liste.
- Concaténer deux listes.
- Trier une Liste.
- …

Bien programmer avec les listes

- Pour chaque méthode transformant des listes :
  - *Dessinez les listes et leurs pointeurs,*
  - *Dessinez les déplacements de pointeurs,*
  - *Traduisez les déplacements de pointeurs en affectations,*
  - ***Ordonnez les affectations dans le temps.***
  - *Enfin, programmez les affectations dans l'ordre*

Conclusion

- Ce chapitre a pour objectif dans un premier temps d'introduire le concept des pointeurs et des VARs dynamiques (Allocation Dynamique).

- Les listes linéaires chaînées sont un outil incontournable à tout programmeur où il est essentiel de bien avoir compris leur fonctionnement et les principes des structures de données récursives qui sont elles-mêmes basées sur les pointeurs.