

Structures chaînées

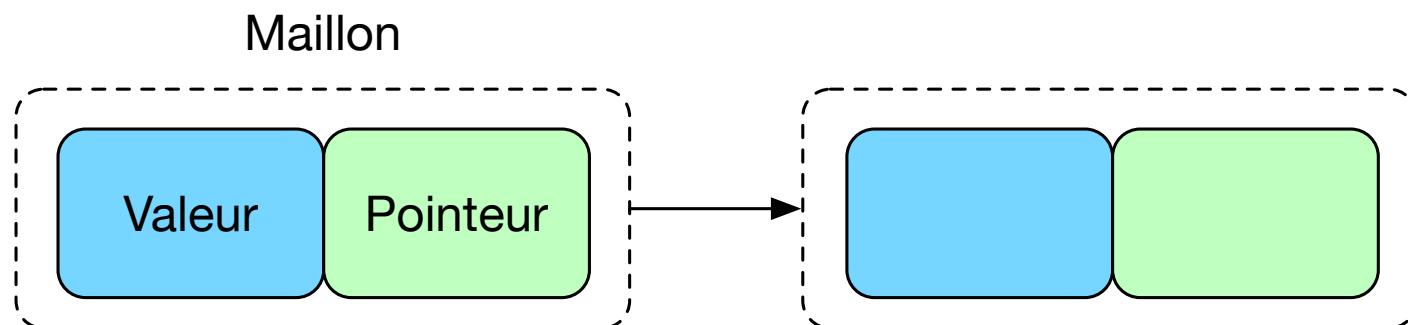
Sébastien Jean

IUT de Valence
Département Informatique

v1.0, 5 janvier 2026

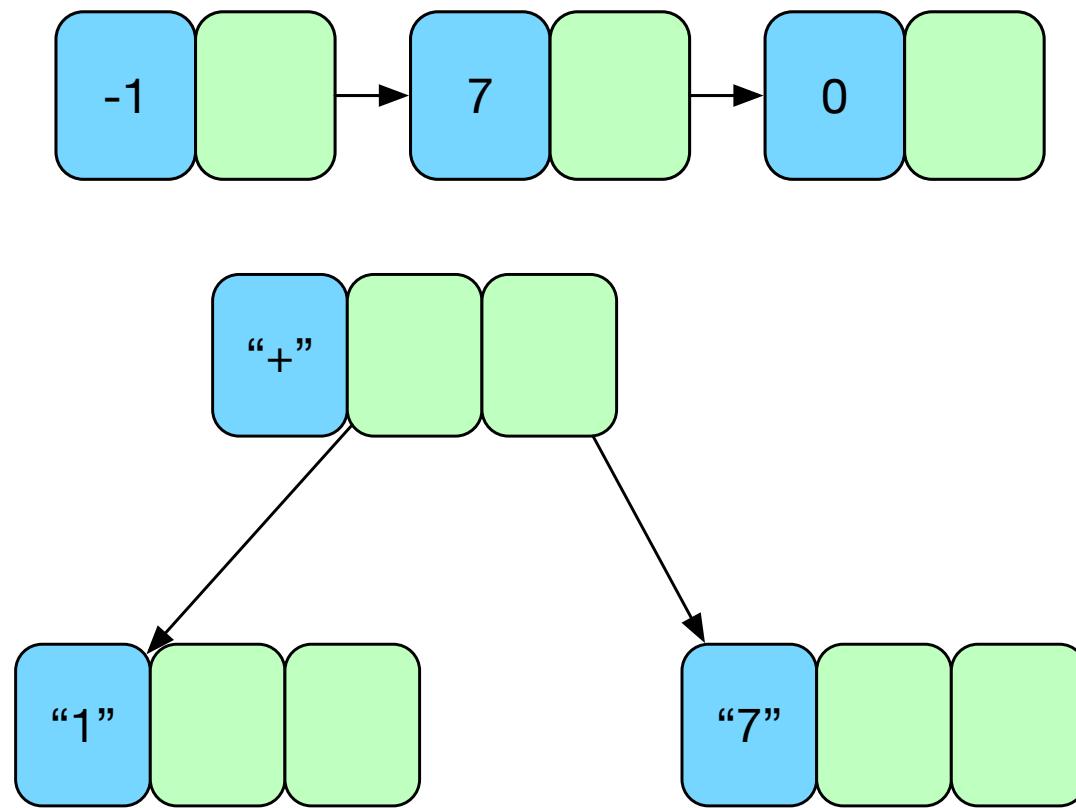
Structure chaînée

- Une **structure chaînée** est un moyen d'organiser des données **dynamiquement**, en établissant des **liaisons** exprimant des **relations**
 - Prédécesseur/ successeur , ...
- L'**élément de base** d'une structure chaînée est un **maillon**, composé :
 - d'une **valeur** (valeur simple, enregistrement, collection)
 - d'un (ou plusieurs) **pointeur**(s), matérialisant la ou les liaisons avec d'autres maillons



Structure chaînée

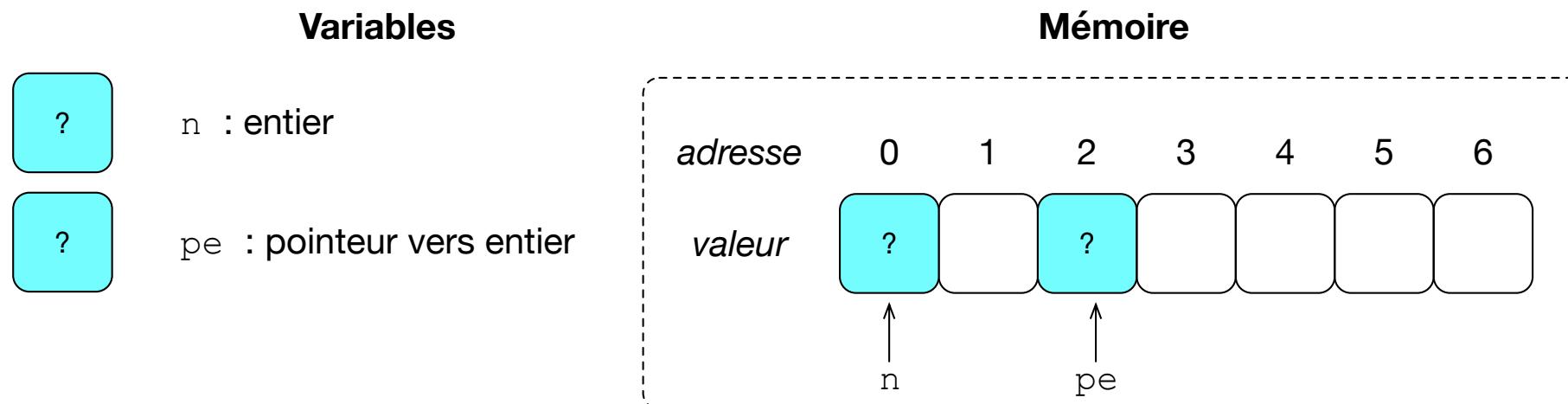
- Les structures chaînées permettent de mettre en œuvre des **vecteurs**, des **piles**, des **files**, des **arbres**, ...



Rappels : variable de type pointeur

- Une **variable de type *pointeur vers un type T*** est une **variable dont la valeur est l'adresse d'un emplacement mémoire** où est stockée une valeur de type T

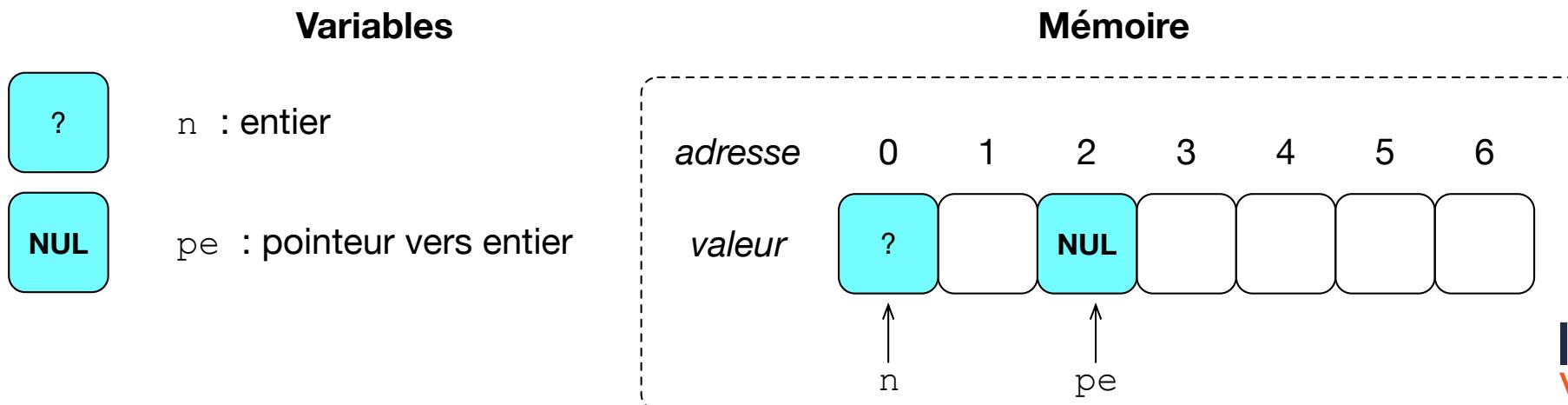
VARIABLE pe : pointeur vers entier



Rappels : initialisation d'une variable de type pointeur

- Une variable de type *pointeur vers un type T* ne doit pas être lue si elle n'est pas initialisée
- On peut initialiser une variable de type pointeur avec la valeur **NUL** pour indiquer qu'elle ne désigne pas encore d'emplacement
 - Elle peut alors être lue, on saura qu'elle ne désigne pas encore d'emplacement

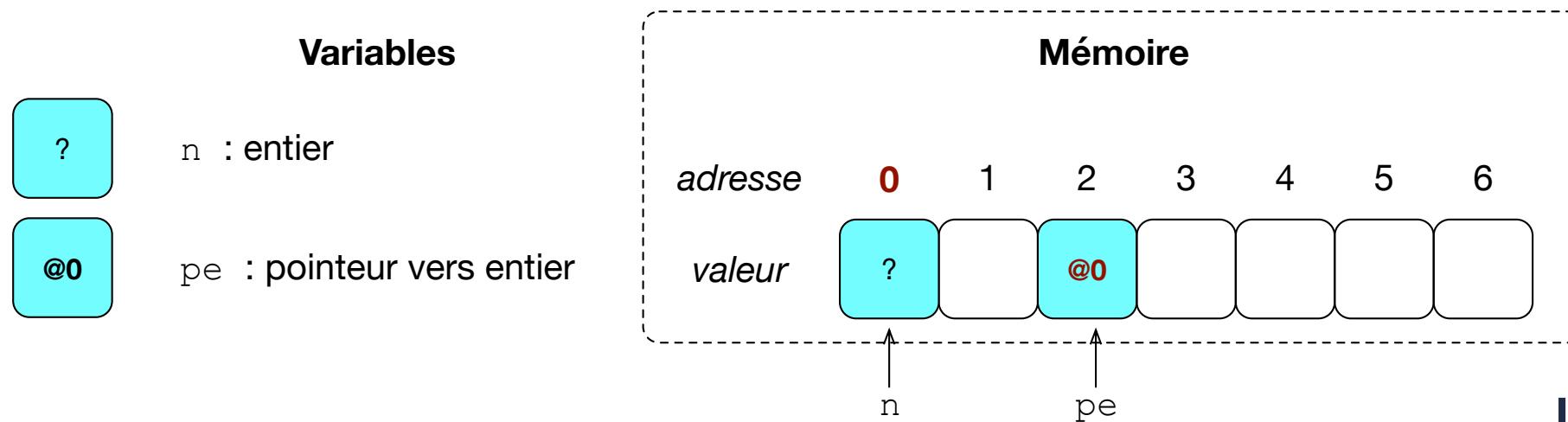
pe \leftarrow NUL



Rappels : initialisation d'une variable de type pointeur

- On ne peut pas « fabriquer d'adresse », **on ne peut affecter comme valeur à une variable de type pointeur que l'adresse d'une variable (ou d'un paramètre) du type attendu**
- l'opérateur **&** désigne l'**adresse d'une variable**

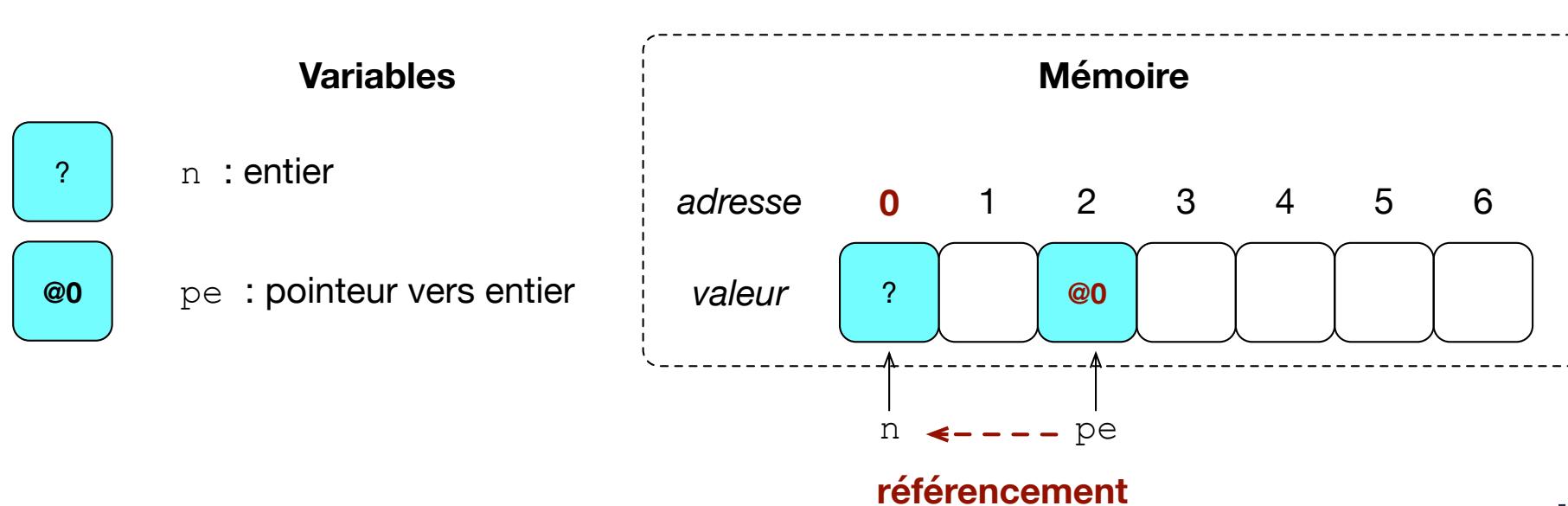
```
pe ← &n
```



Rappels : initialisation d'une variable de type pointeur

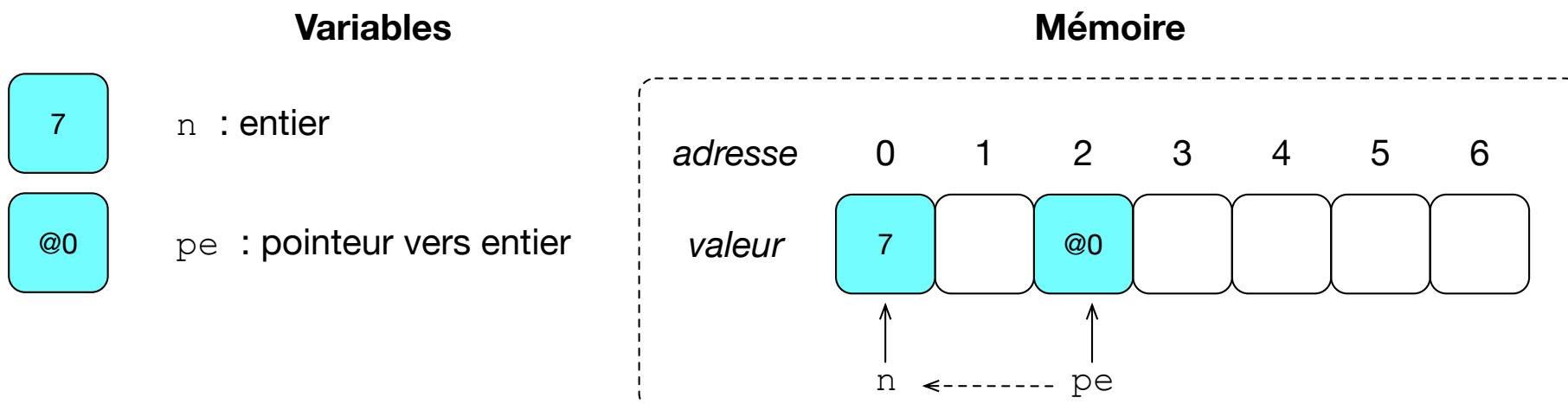
- La valeur d'une variable de type pointeur vers le type T est l'**adresse de l'emplacement mémoire d'une valeur de type T**
 - On dit que le pointeur **référence** la variable

```
pe ← &n
```



Rappels : déréferencement d'une variable de type pointeur

- L'opérateur \uparrow exprime le **déréférencement**, c'est à dire la **désignation de l'emplacement mémoire dont l'adresse est contenue dans le pointeur**

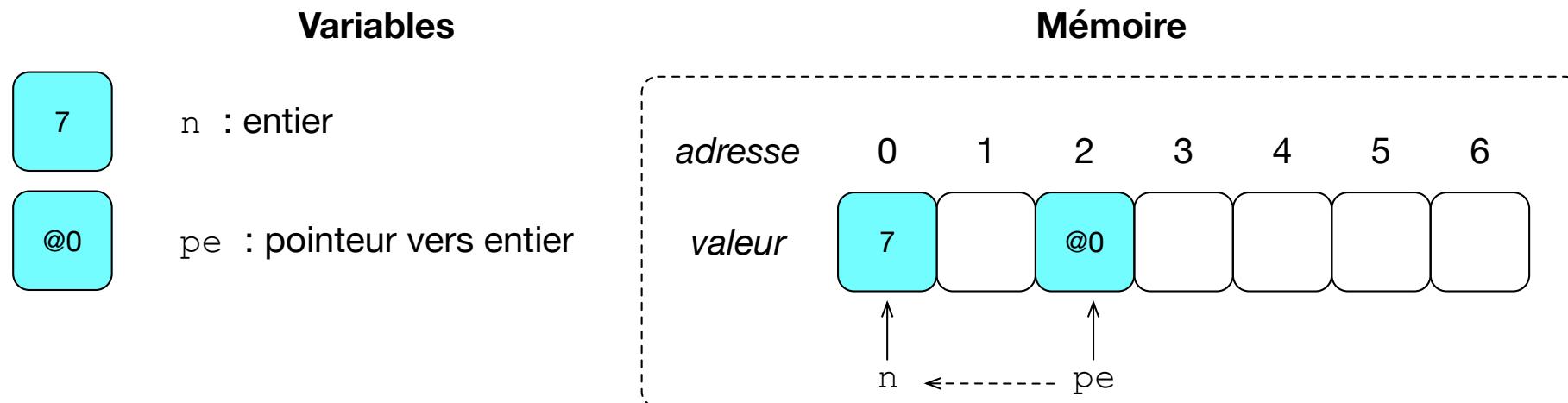


- Un pointeur dont la valeur est NUL ou dont la valeur désigne un emplacement non initialisé ne doit pas être déréférencé

Rappels : déréférencement d'une variable de type pointeur

- L'opérateur \uparrow peut être utilisé dans une **expression**
 - Dans ce cas l'**expression vaut la valeur contenue dans l'emplacement dont l'adresse est contenue dans le pointeur**

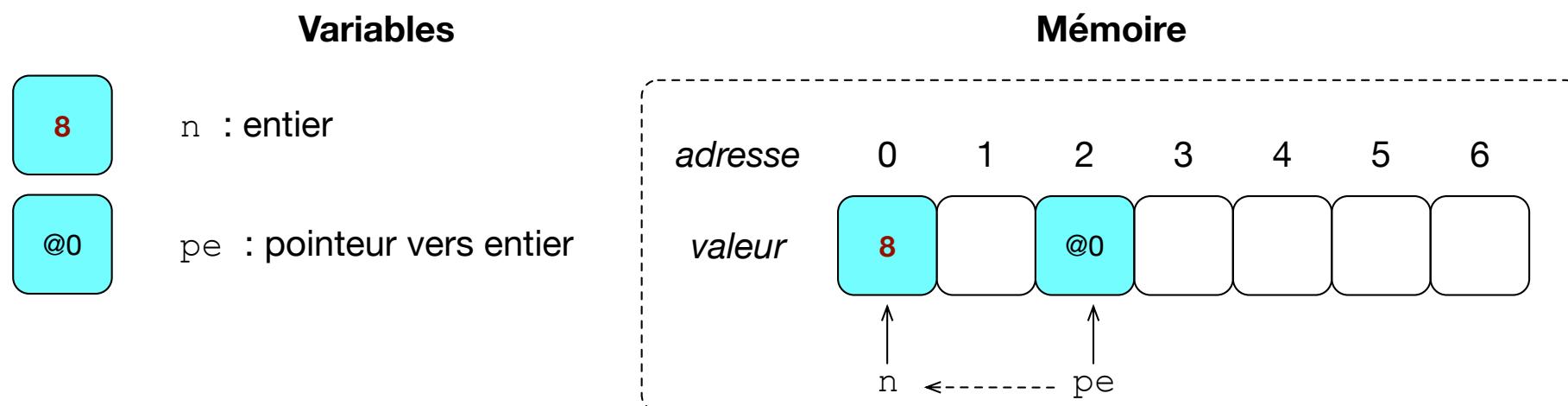
afficher_entier(pe \uparrow) *// affiche 7*



Rappels : déréférencement d'une variable de type pointeur

- L'opérateur \uparrow peut être utilisé dans une **affectation**
 - Dans ce cas l'**emplacement dont l'adresse est contenue dans le pointeur est réaffecté avec la nouvelle valeur**

$pe \uparrow \leftarrow 8$



$pe \uparrow \leftarrow pe \uparrow +1$ // ?

Rappels : allocation dynamique de mémoire

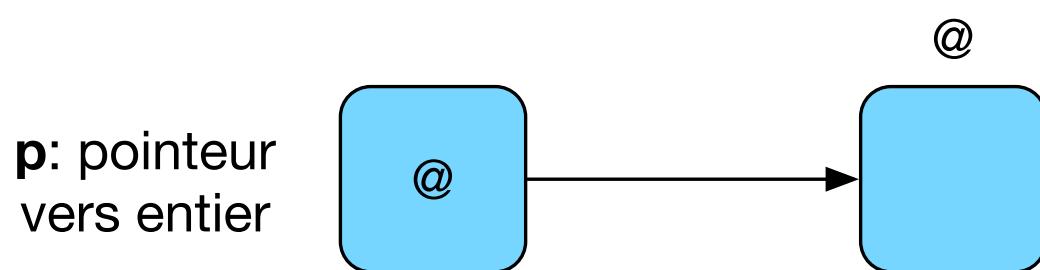
- **Allocation dynamique**

- Réservation de zone mémoire *à la volée*, sans variable

```
VARIABLE p : pointeur vers entier
```

```
p ← ALLOUER entier
```

- ALLOUER réserve une **nouvelle zone mémoire** de **taille adaptée** au stockage d'une valeur de type donné et **retourne l'adresse de début de la zone mémoire**



Rappels : allocation dynamique de mémoire

● Libération de mémoire

- En théorie : pas de contrainte sur la quantité de mémoire disponible
- En pratique : ressources limitées, judicieux d'économiser.

```
VARIABLE p : pointeur vers entier
p ← ALLOUER entier
...
LIBERER p
```

- LIBERER libère la zone mémoire située à partir de l'adresse contenue dans le pointeur et de la taille correspondant au type de valeur référencée par le pointeur

p: pointeur
vers entier



@



Rappels : enregistrements

- Un **enregistrement** est **composé de plusieurs valeurs** appelées **champs** ou **membres**
 - Nombre de champs **fixe**, champs **nommés** et de **type quelconque**
 - Les **opérations** se limitent à la **lecture et l'affectation des champs**
 - Accès aux champs via la **notation pointée** (`variable.champs`)

ENREGISTREMENT Point

CHAMPS x : réel

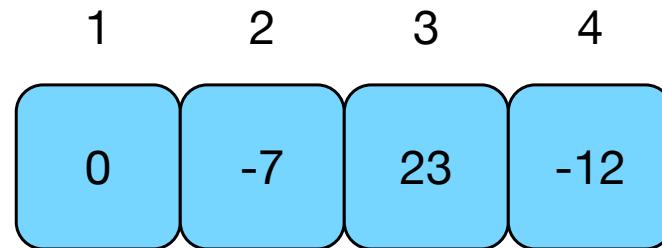
CHAMPS y : réel

FIN ENREGISTREMENT

VARIABLE p : Point

`p.x ← 0.0`

`p.y ← p.x + 1`



- Type de données **linéaire** et **dynamique**, collection de valeurs (de type T) stockées dans des **cases contigües** identifiées par un **indice**
 - Indices allant de 1 au **nombre d'éléments**
 - Possibilité de **lire** et **écrire** l'élément à un indice donné
 - Possibilité d'**insérer** et **retirer** un élément à un indice donné
 - Possibilité d'**obtenir la taille** (nombre d'éléments)

Rappels : TDA : Vecteur

- **Nom** : Vecteur (de T)
- **Dépendances** : Entier, T (type des éléments)
- **Opérations** :
 - **Constructeurs** :
 - `vecteur_vide` : \rightarrow Vecteur
 - **Transformateurs** :
 - `ecrire` : Vecteur \times Entier \times T \rightarrow Vecteur
 - `inserer` : Vecteur \times Entier \times T \rightarrow Vecteur
 - `retirer` : Vecteur \times Entier \rightarrow Vecteur
 - **Observateurs** :
 - `lire` : Vecteur \times Entier \rightarrow T
 - `taille` : Vecteur \rightarrow Entier
- *A suivre ...*

● Pré-conditions

- `ecrire(v, n, e)` $\rightarrow 1 \leq n \leq \text{taille}(v)$
- `inserer(v, n, e)` $\rightarrow 1 \leq n \leq \text{taille}(v) + 1$
- `retirer(v, n)` $\rightarrow 1 \leq n \leq \text{taille}(v)$
- `lire(v, n)` $\rightarrow 1 \leq n \leq \text{taille}(v)$

• Axiomes

- `taille(vecteur_vide()) = 0`
- `taille(ecrire(v, n, e)) = taille(v)`
- `taille(inserer(v, n, e)) = taille(v) + 1`
- `taille(retirer(v, n)) = taille(v) - 1`
- `lire(ecrire(vecteur_vide(), 0, e), 0) = e`
- `lire(ecrire(v, 0, e), n) = lire(v, n) si n ≥ 1`
- `retirer(inserer(v, n, e), n) = v`
- `lire(inserer(v, n, e), n) = e`
- `lire(inserer(v, n, e), p) = lire(v, p) si p < n`
- `lire(inserer(v, n, e), p) = lire(v, p-1) si p ≥ n`
- `lire(retirer(v, n), p) = lire(v, p) si p < n`
- `lire(retirer(v, n), p) = lire(v, p+1) si p ≥ n`

Exercice

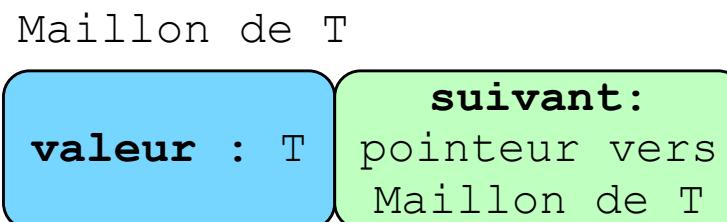
- On définit un enregistrement **Maillon de T** permettant de représenter un **maillon de structure simplement chaînée** composé :
 - d'une **valeur** (de type **T**)
 - d'un **pointeur** vers **Maillon de T**

ENREGISTREMENT **Maillon de T**

CHAMPS **valeur** : **T**

CHAMPS **suivant** : **pointeur vers Maillon de T**

FIN ENREGISTREMENT



Exercice

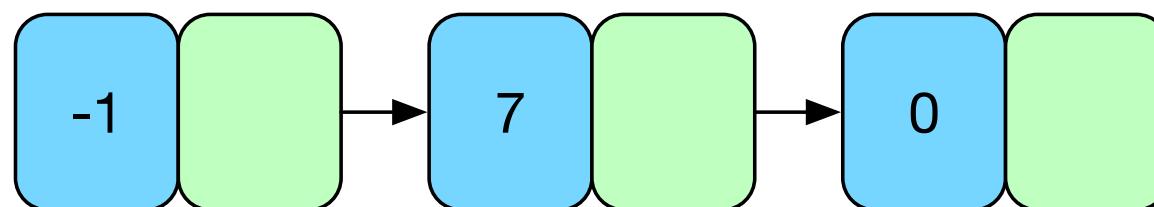
- On définit un enregistrement **Vecteur de T** permettant de manipuler un **Vecteur (dynamique)** en s'appuyant sur une **structure chaînée**

ENREGISTREMENT Vecteur de T

CHAMPS `taille` : entier

CHAMPS `debut` : pointeur vers **Maillon de T**

FIN ENREGISTREMENT

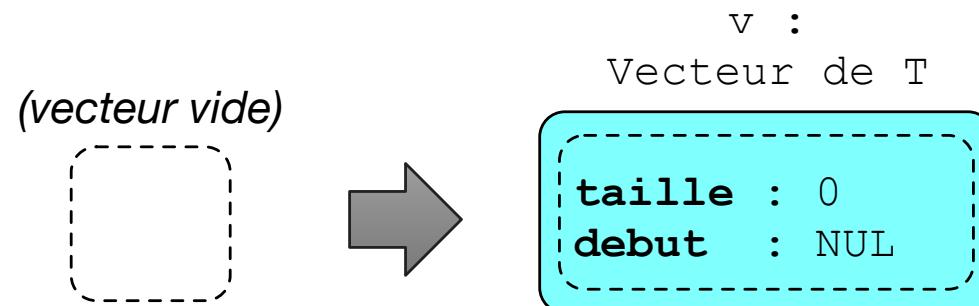


Exercice

- On considère que :
 - La déclaration d'une variable Maillon de T ou Vecteur de T **n'initialise pas les enregistrements**
 - Les **transformateurs** de Vecteur de T sont des **mutateurs**
 - Les **enregistrements** sont passés par copie
- Schématiser et écrire les opérations :
 - vecteur_vide, taille,
 - lire, ecrire,
 - retirer et inserer



vecteur_vide



FONCTION **vecteur_vide**

(p : pointeur vers Vecteur de T) : (aucun)

$(p \uparrow).taille \leftarrow 0$
 $(p \uparrow).debut \leftarrow NUL$

FIN FONCTION

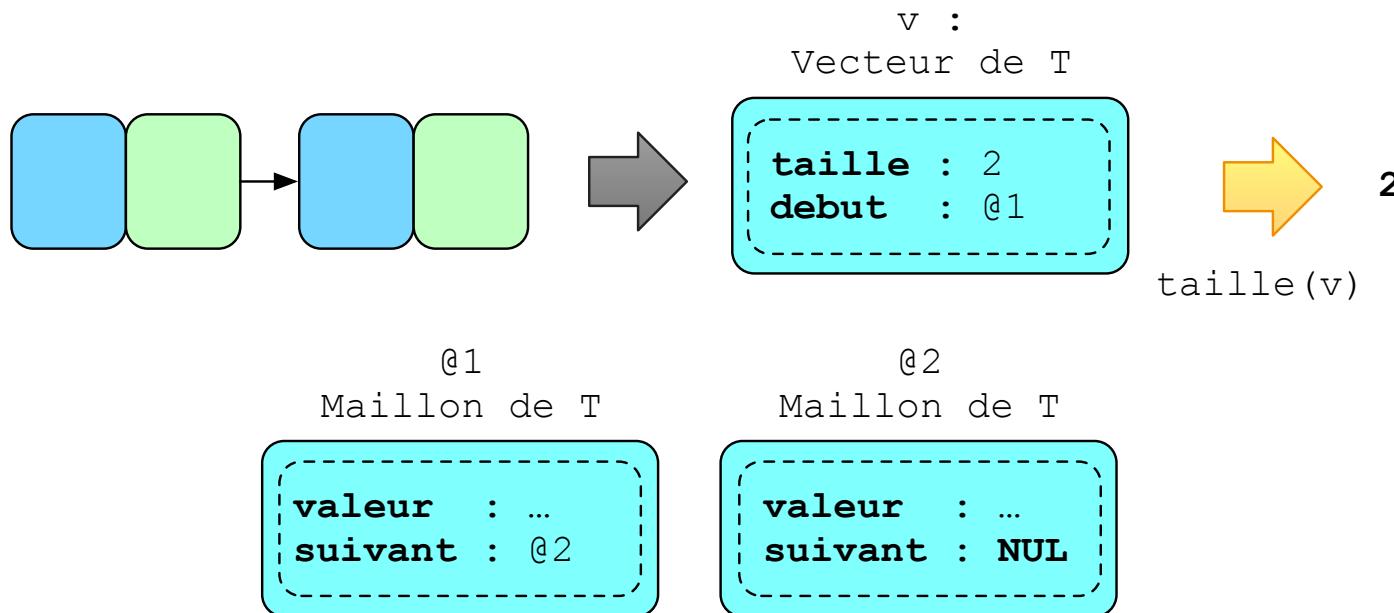
taille

FONCTION `taille (v : Vecteur de T) : entier`

RETOURNER `v.taille`

FIN FONCTION

- Cas d'un vecteur avec 2 éléments



lire

```
FONCTION lire (v : Vecteur de T, index : entier) : T
```

```
VARIABLE i : entier
```

```
VARIABLE p : pointeur vers Maillon de T
```

```
p ← v.debut
```

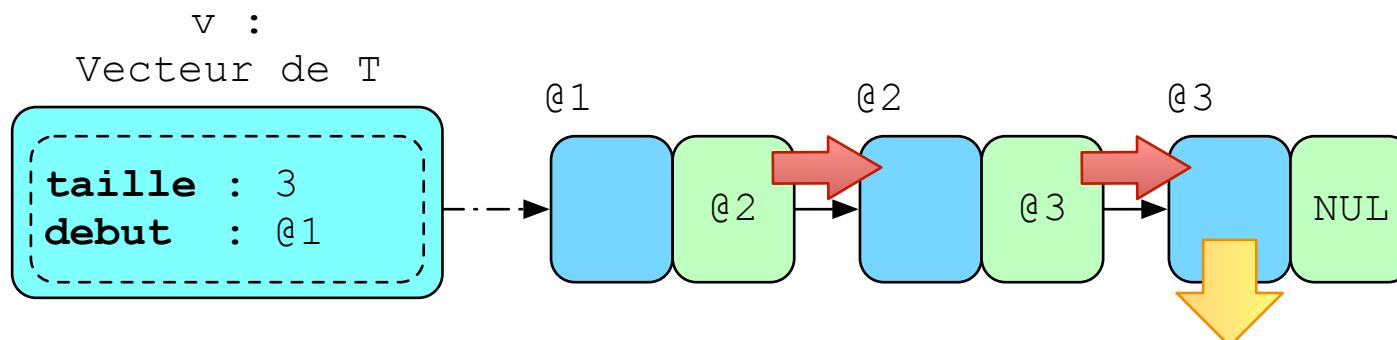
```
POUR i DE 1 à index - 1 PAR PAS DE 1
```

```
    p ← (p↑).suivant
```

```
FIN POUR
```

```
RETOURNER (p↑).valeur
```

```
FIN FONCTION
```



écrire

```
FONCTION écrire (v : Vecteur de T, index : entier,  
                  t : T) : (aucun)
```

```
VARIABLE i : entier
```

```
VARIABLE p : pointeur vers Maillon de T
```

```
p ← v.debut
```

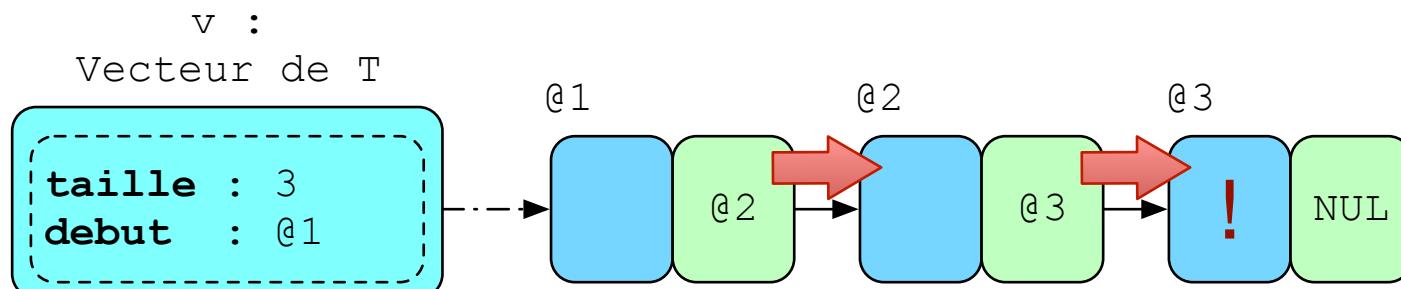
```
POUR i DE 1 à index - 1 PAR PAS DE 1
```

```
    p ← (p↑).suivant
```

```
FIN POUR
```

```
(p↑).valeur ← t
```

```
FIN FONCTION
```



inserer

```
FONCTION inserer (pv : pointeur vers Vecteur de T,
                   index : entier, t : T) : (aucun)
  VARIABLE i : entier
  VARIABLE p : pointeur vers Maillon de T
  VARIABLE np : pointeur vers Maillon de T

  // A suivre ...

FIN FONCTION
```

inserer

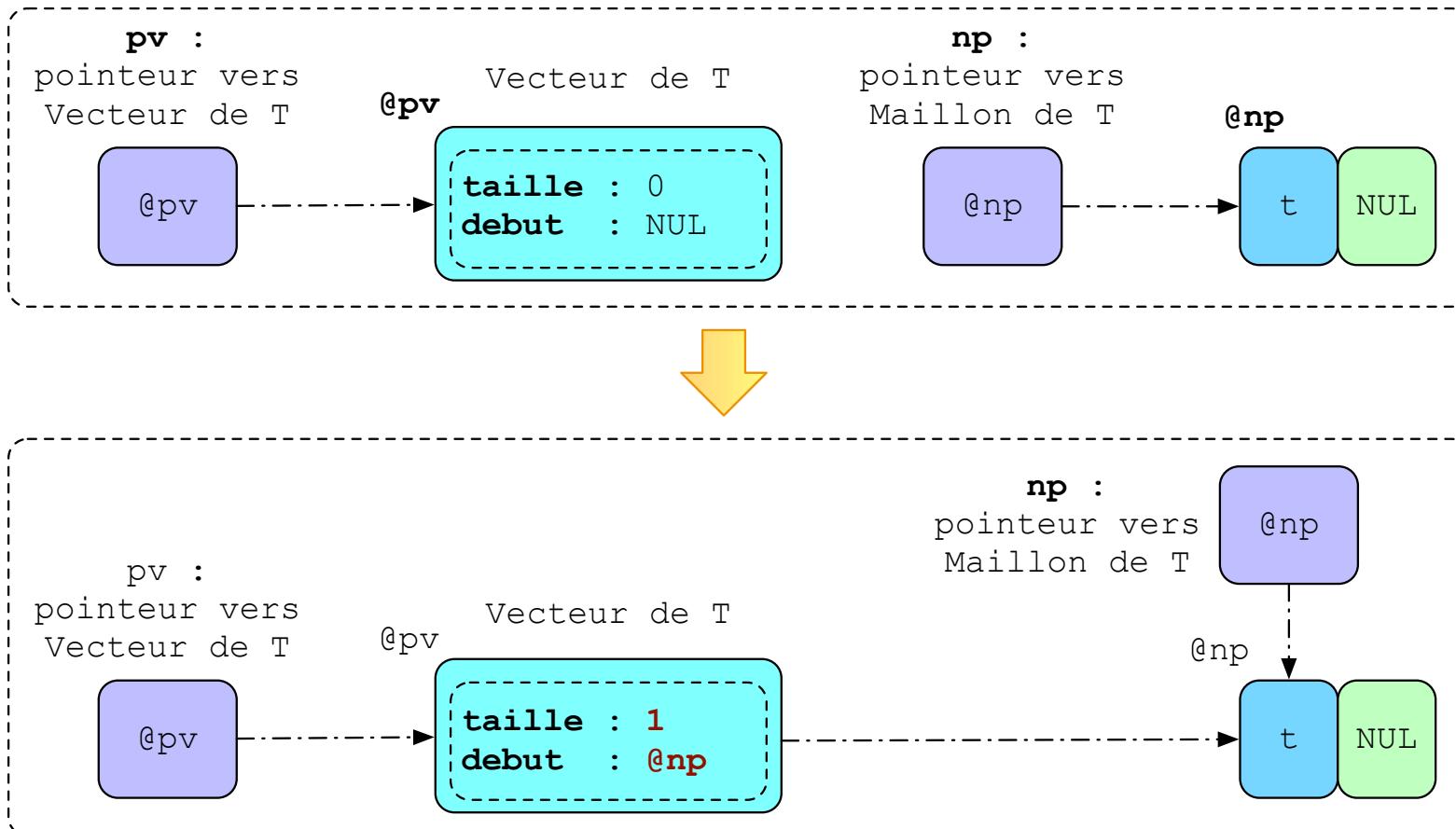
```
FONCTION inserer (pv : pointeur vers Vecteur de T,  
                   index : entier, t : T) : (aucun)
```

```
// Suite
```

```
np ← ALLOUER Maillon de T  
(np↑).valeur ← t  
(np↑).suivant ← NUL  
SI (pv↑).taille = 0 ALORS  
    // Cas 1 (à suivre) : vecteur vide  
SINON  
    SI index = 1 ALORS  
        // Cas 2 (à suivre) : insertion en tête  
    SINON  
        // Cas 3 (à suivre) : insertion au milieu  
    FIN SI  
FIN SI  
(pv↑).taille ← (pv↑).taille + 1  
FIN FONCTION
```

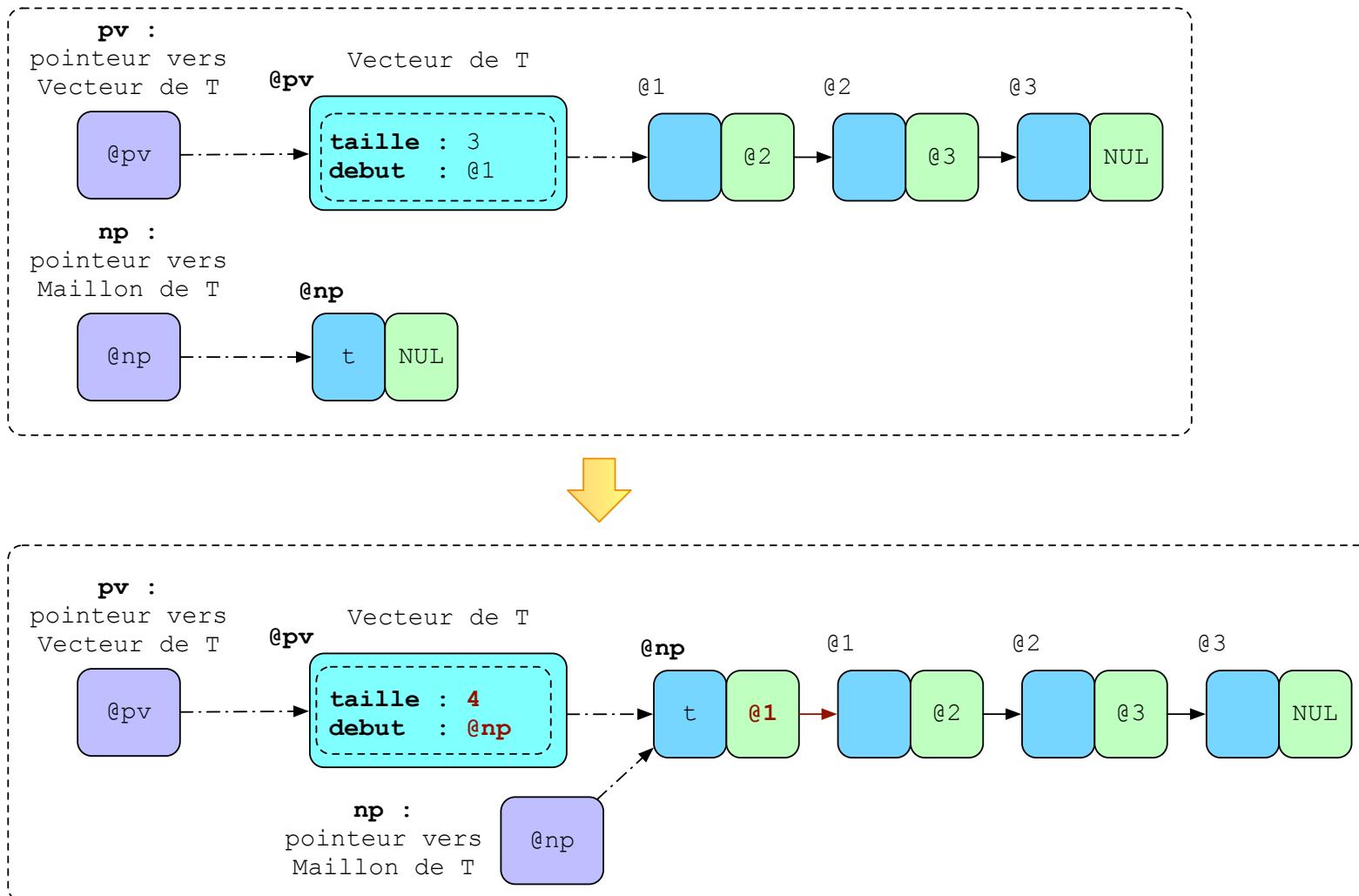
inserer : vecteur vide

$(pv \uparrow) . \text{debut} \leftarrow np$



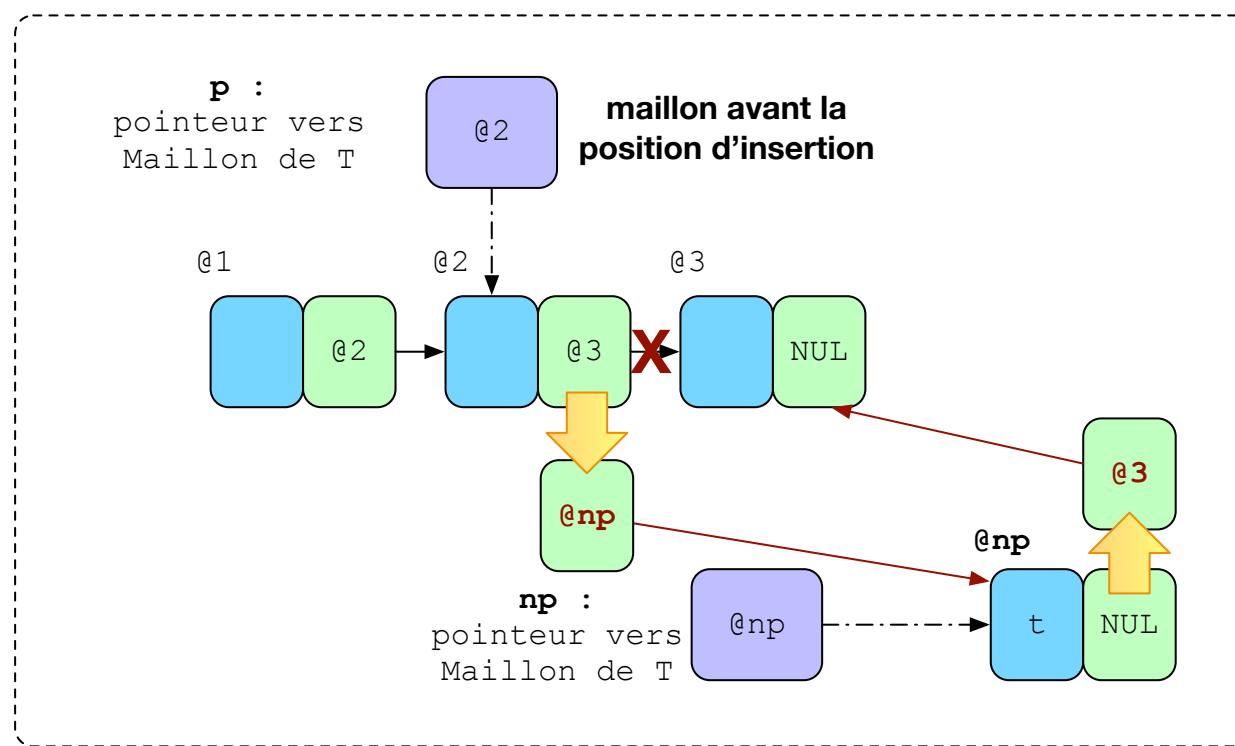
insérer : insertion en tête, vecteur non vide

```
(np↑).suivant ← (pv↑).début  
(pv↑).début ← np
```



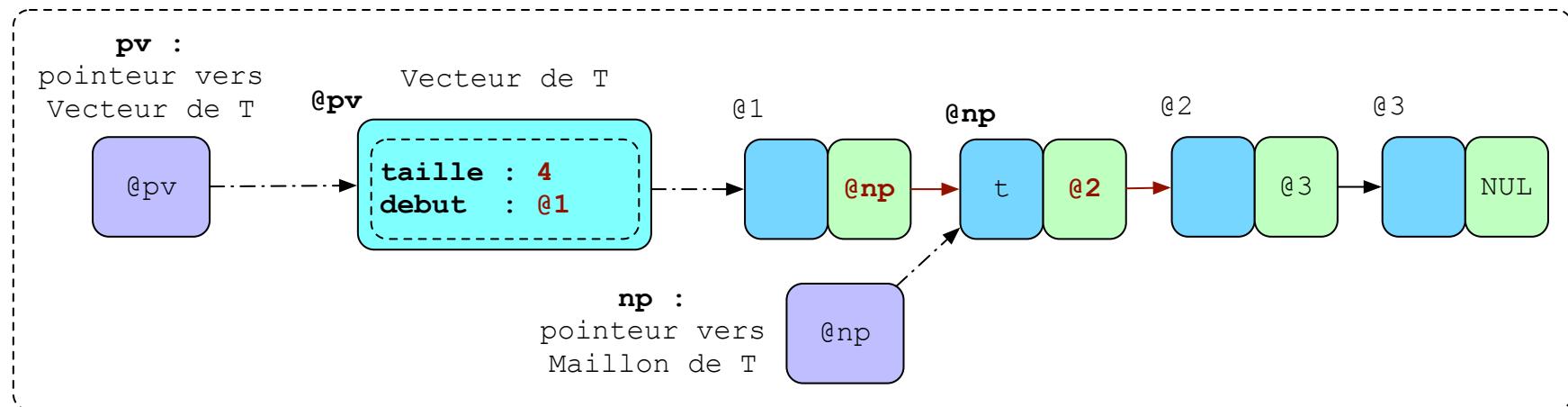
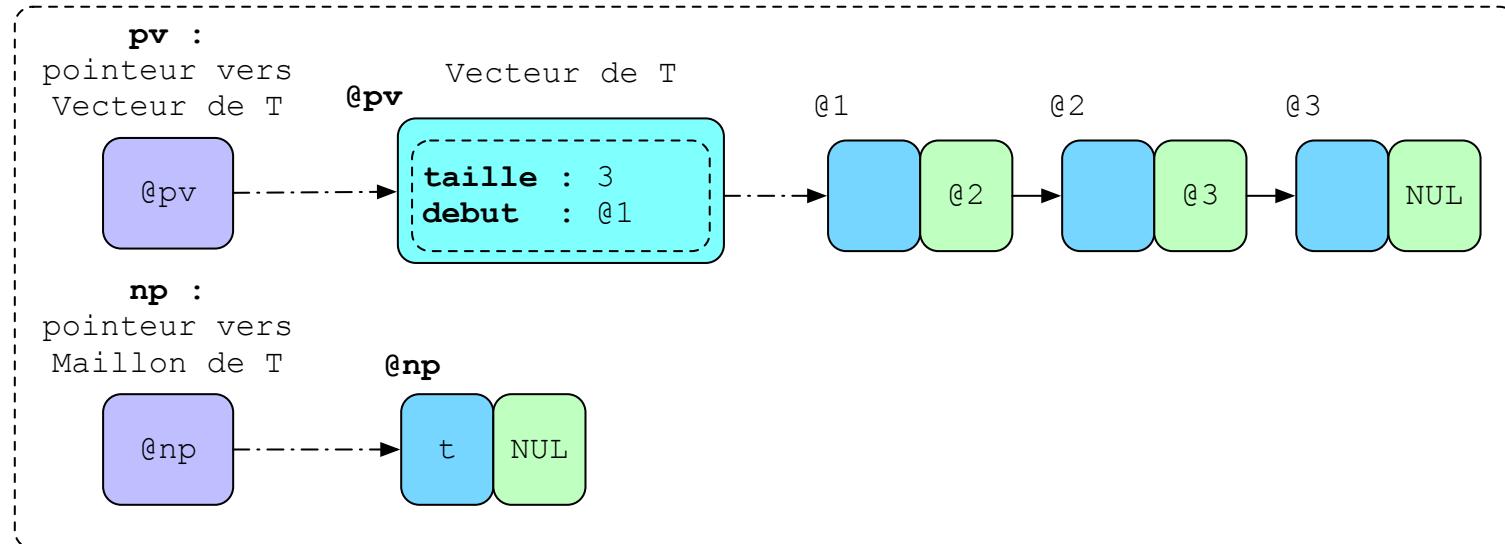
inserer : insertion au milieu, vecteur non vide

```
p ← (pv↑).debut
POUR i DE 1 à index - 2 PAR PAS DE 1
    p ← (p↑).suivant
FIN POUR
(np↑).suivant ← (p↑).suivant
(p↑).suivant ← np
```



inserer : cas du vecteur non vide

- Exemple avec insertion en position 2



retirer

```
FONCTION retirer (pv : pointeur vers Vecteur de T,  
                  index : entier) : (aucun)
```

```
VARIABLE i : entier
```

```
VARIABLE p : pointeur vers Maillon de T
```

```
VARIABLE xp : pointeur vers Maillon de T
```

```
SI (pv↑).taille = 1 ALORS
```

```
    // Cas du dernier (seul) élément
```

```
SINON
```

```
    SI index = 1 ALORS
```

```
        // Cas du premier élément
```

```
    SINON
```

```
        // Autre cas...
```

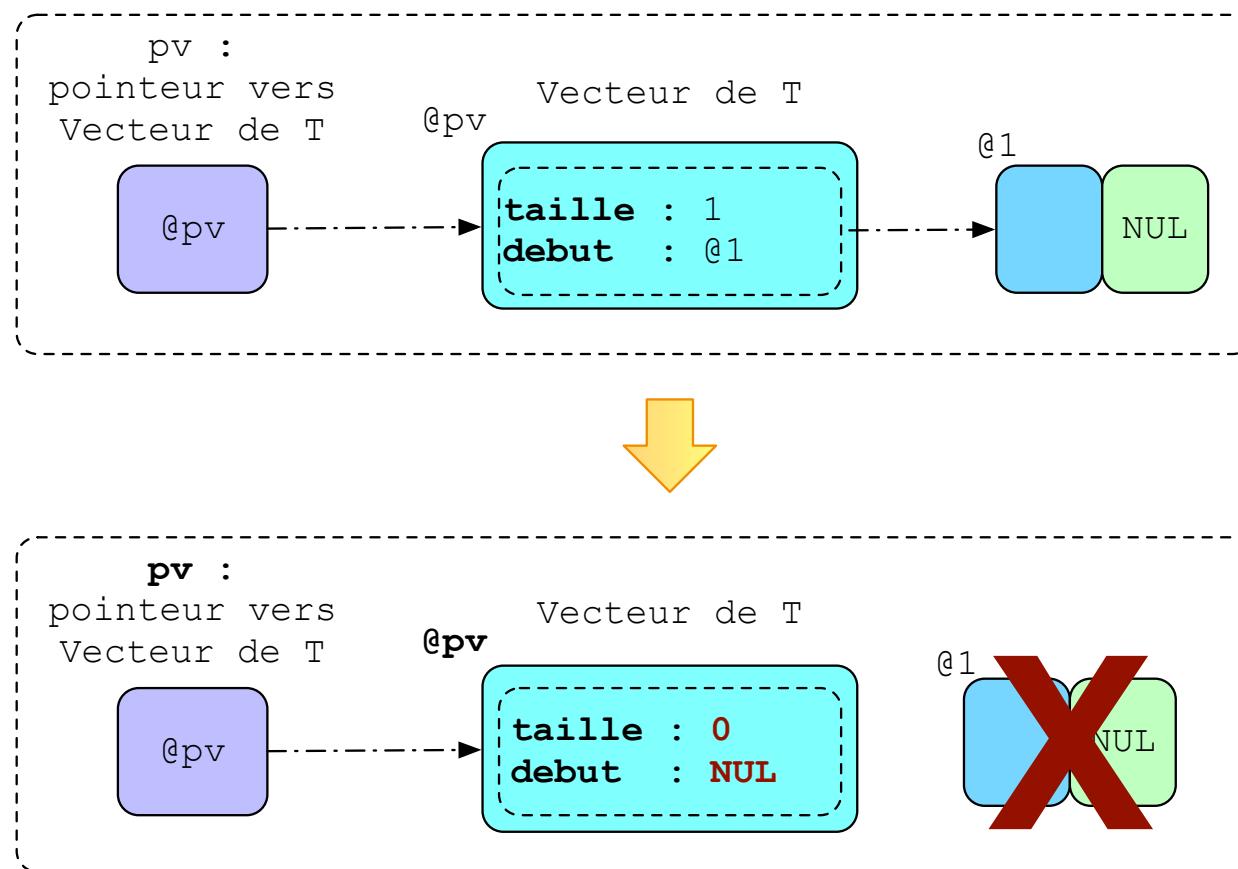
```
    FIN SI
```

```
FIN SI
```

```
(pv↑).taille ← (pv↑).taille - 1
```

retirer : cas du seul élément

LIBERER (pv↑).debut
(pv↑).debut = NUL

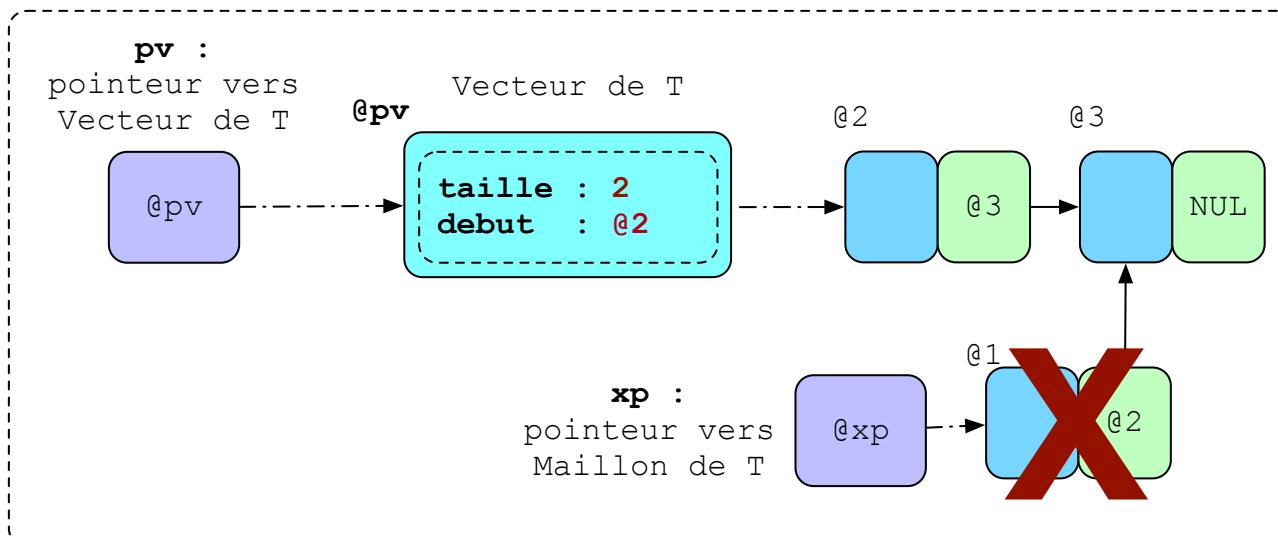
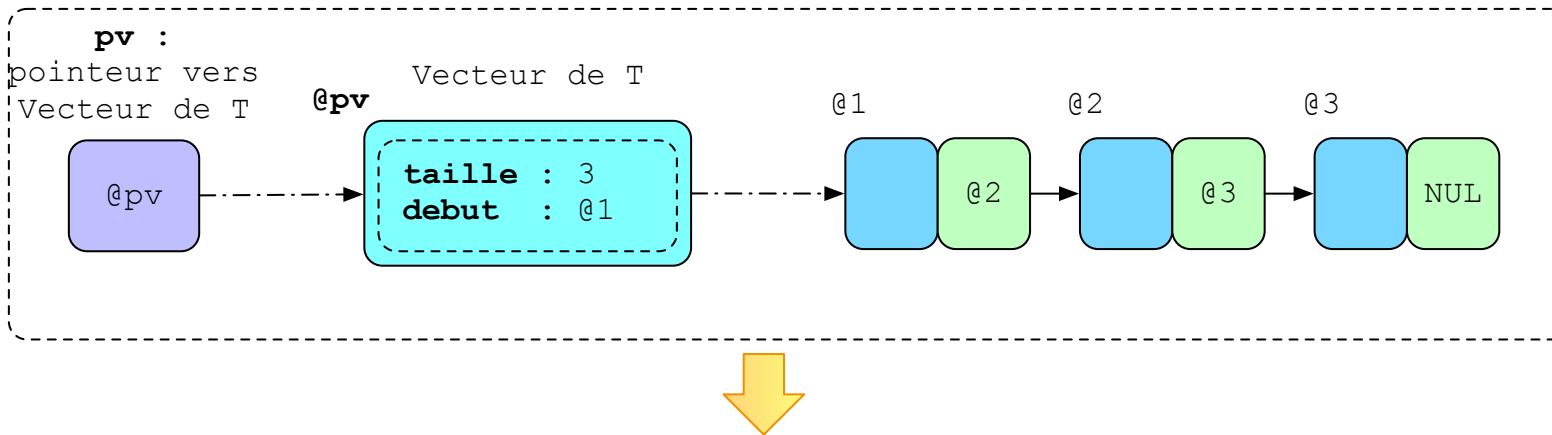


retirer : cas du premier élément

$xp \leftarrow (pv\uparrow).debut$

$(pv\uparrow).debut \leftarrow (xp\uparrow).suivant$

LIBERER (xp)

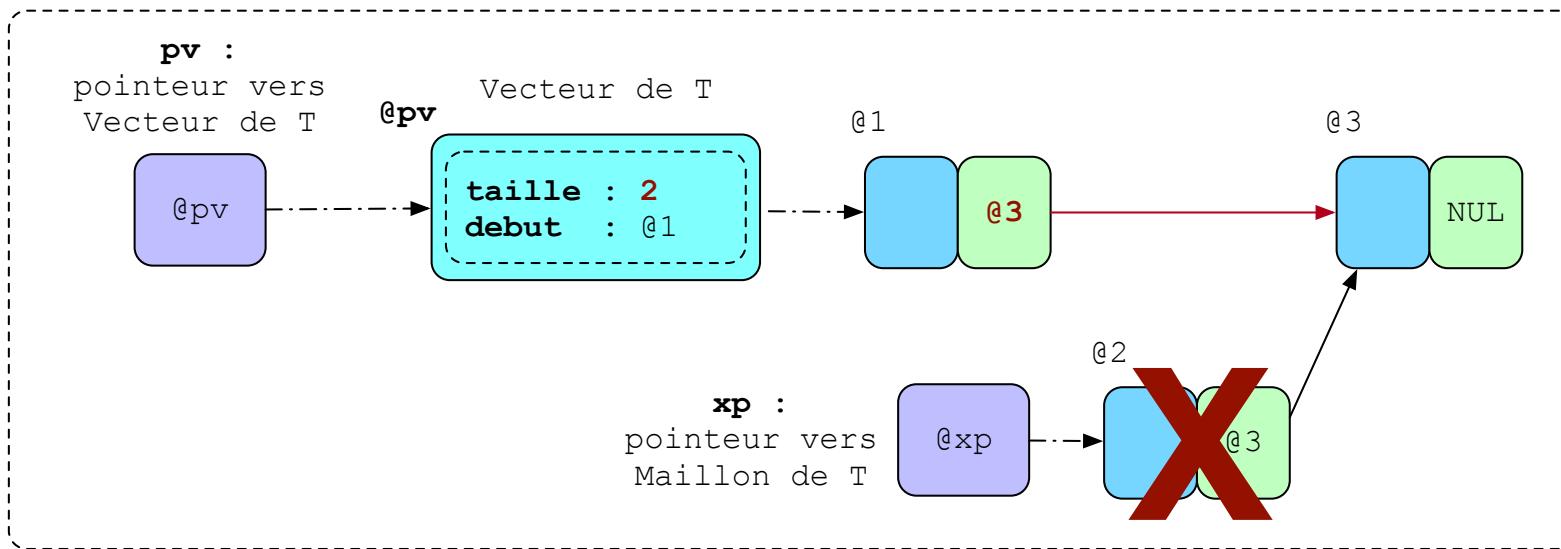
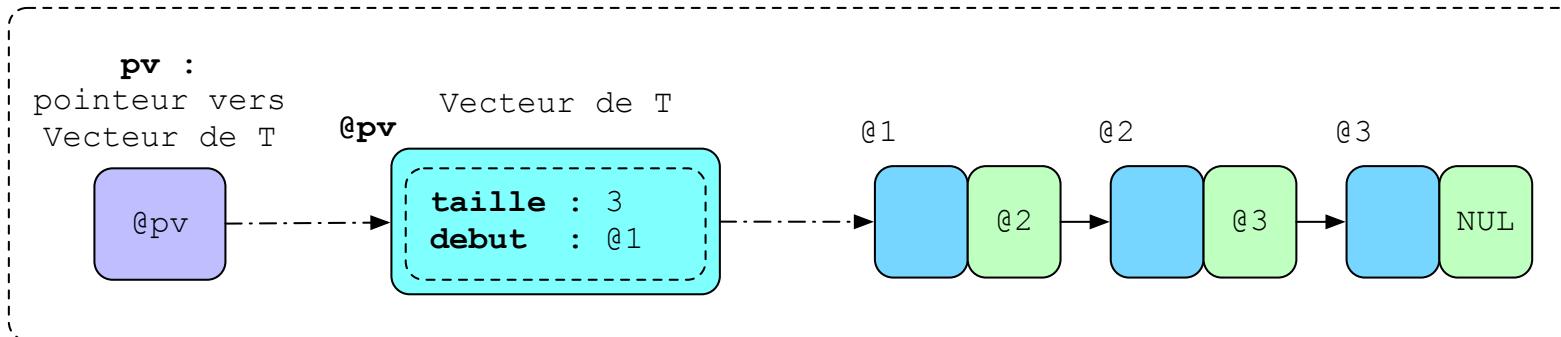


retirer : autre cas

```
p ← (pv↑).debut
POUR i DE 1 à index-2 PAR PAS DE 1
    p ← (p↑).suivant
FIN POUR
xp ← (p↑).suivant
(p↑).suivant ← (xp↑).suivant
LIBERER (xp)
```

retirer : autre cas

- Exemple avec retrait en position 2



Fin !

