

Allocation dynamique de mémoire, vecteurs

Sébastien Jean

IUT de Valence
Département Informatique

v1.0, 15 décembre 2025

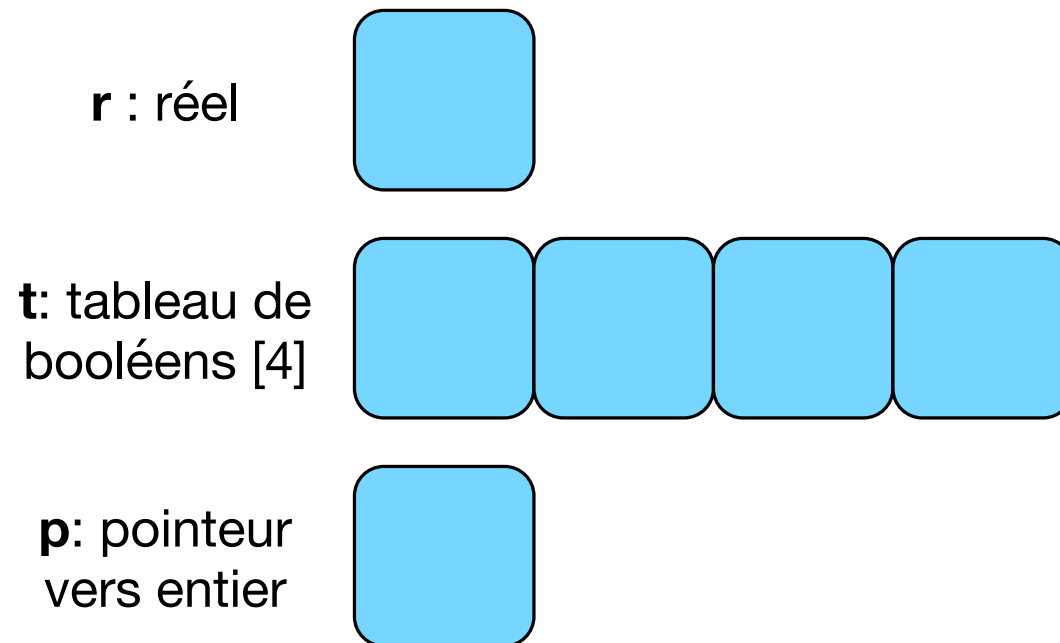
Allocation dynamique de mémoire

- **Allocation statique** (rappel)
 - Association d'une zone mémoire à une variable

VARIABLE r : réel

VARIABLE t : tableau de booléens [4]

VARIABLE p : pointeur vers entier



Allocation dynamique de mémoire

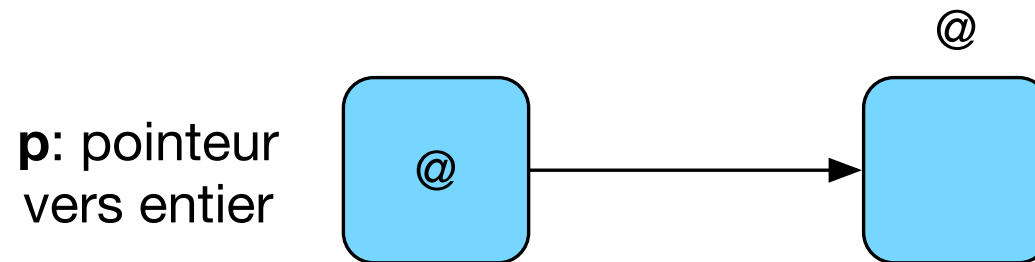
- Allocation dynamique

- Réserve 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



Allocation dynamique de mémoire

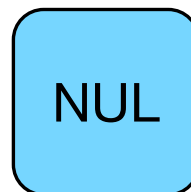
- **Libération de mémoire**

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

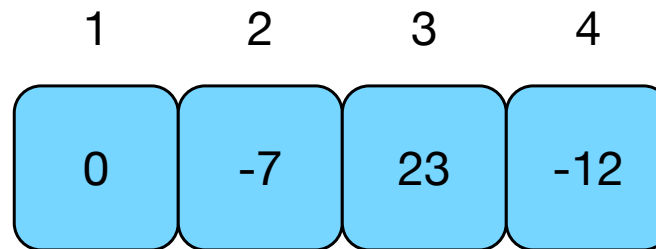
```
VARIABLE p : pointeur vers entier  
p ← ALLouer entier  
...  
LIBERER p
```

- 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



TDA : *Vecteur* (rappels)



- 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)

TDA : *Vecteur* (rappels, suite)

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

TDA : *Vecteur* (rappels, suite)

- Pré-conditions

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

TDA : *Vecteur* (rappels, suite)

- **Axiomes**

- `taille(vecteur_vide()) = 0`
- `taille(ecrire(v, n, e)) = taille(v)`
- `taille(insérer(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 $n \geq 1$`
- `retirer(insérer(v, n, e), n) = v`
- `lire(insérer(v, n, e), n) = e`
- `lire(insérer(v, n, e), p) = lire(v, p) si $p < n$`
- `lire(insérer(v, n, e), p) = lire(v, p-1) si $p \geq n$`
- `lire(retirer(v, n), p) = lire(v, p) si $p < n$`
- `lire(retirer(v, n), p) = lire(v, p+1) si $p \geq n$`

Enregistrements (rappels)

- 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
```

Exercice

- On définit un enregistrement Vecteur de T permettant de manipuler un Vecteur (dynamique) en s'appuyant sur un tableau (statique)
 - capacité / 2 si taille \leq 25% capacité, capacité x 2 si taille $>$ capacité

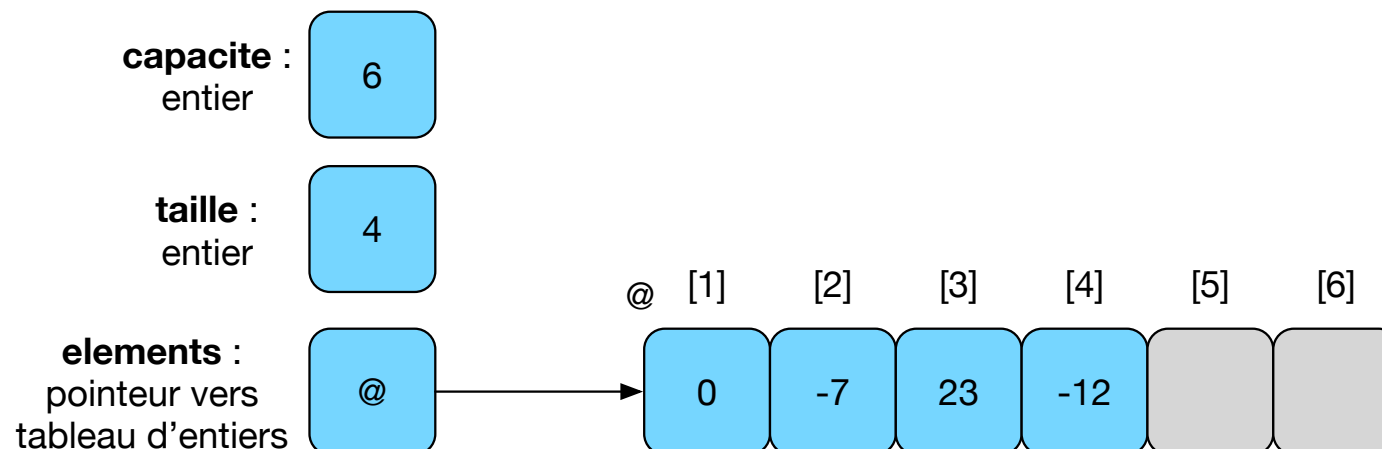
ENREGISTREMENT Vecteur de T

CHAMPS capacite : entier

CHAMPS taille : entier

CHAMPS elements : pointeur vers tableau de T

FIN ENREGISTREMENT



Exercice

- Ecrire les opérations :
 - vecteur_vide, taille,
 - lire, ecrire,
 - retirer et inserer



- Les transformateurs sont considérés comme des producteurs

Interlude : fonctions paramétrées

- Une fonction peut aussi être **paramétrique** (paramètre de type)
 - Finalement la fonction décrit une **famille de fonctions** qui ont un **comportement similaire** mais qui s'appliquent à des **types de données différents**

```
FONCTION est_present(tab : tableau de T,  
                    n : entier,  
                    t : T) : booléen  
  
FIN FONCTION
```

Interlude : fonctions paramétrées

```
FONCTION  est_present(tab : tableau de T,  
                    n : entier,  
                    t : T) : booléen
```

```
    VARIABLE i : entier
```

```
    POUR i DE 1 A n PAR PAS DE 1
```

```
        SI tab[i] = t ALORS
```

```
            RETOURNER VRAI
```

```
        FIN SI
```

```
    FIN POUR
```

```
    RETOURNER FAUX
```

```
FIN FONCTION
```

Interlude : fonctions paramétrées

- Dans notre pseudo-code, à l'**appel d'une fonction** paramétrée, le(s) type(s) de **paramètres** fixe(nt) ce que vaut T

```
VARIABLE t : tableau d'entiers [3]
```

```
t[1] ← 7
```

```
t[2] ← 2
```

```
t[3] ← -5
```

```
affiche_booleen(est_present(t, 6))
```

Exercice : vecteur_vide

```
FONCTION vecteur_vide() : Vecteur de T
```

```
    CONSTANCE C_DEF : entier (10) // (locale ou globale)
```

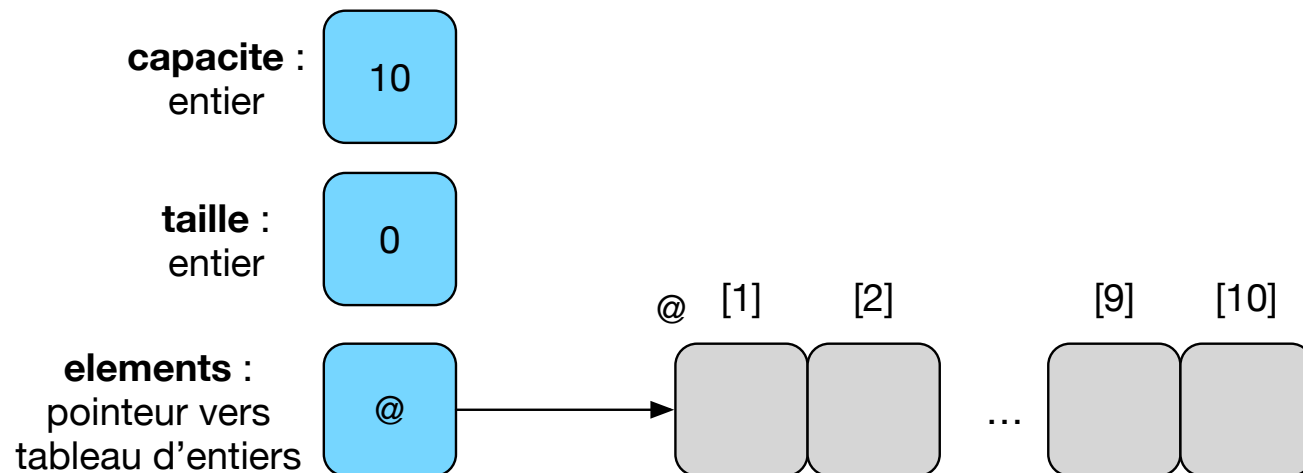
```
    VARIABLE resultat : Vecteur de T
```

```
    resultat.capacite ← C_DEF
```

```
    resultat.taille ← 0
```

```
    resultat.elements ← ALLOUER tableau de T [C_DEF]
```

```
FIN FONCTION
```



Exercice : taille

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

```
    RETOURNER v.taille
```

```
FIN FONCTION
```

- N.B. : on pourrait aussi se passer de fonction et accéder directement au champs `taille`
 - Accéder directement à la structure risque de compromettre l'intégrité (taille et elements sont liés)
- cf. plus tard la notion d'*encapsulation* en *conception orienté objet* ...

Exercice : lire

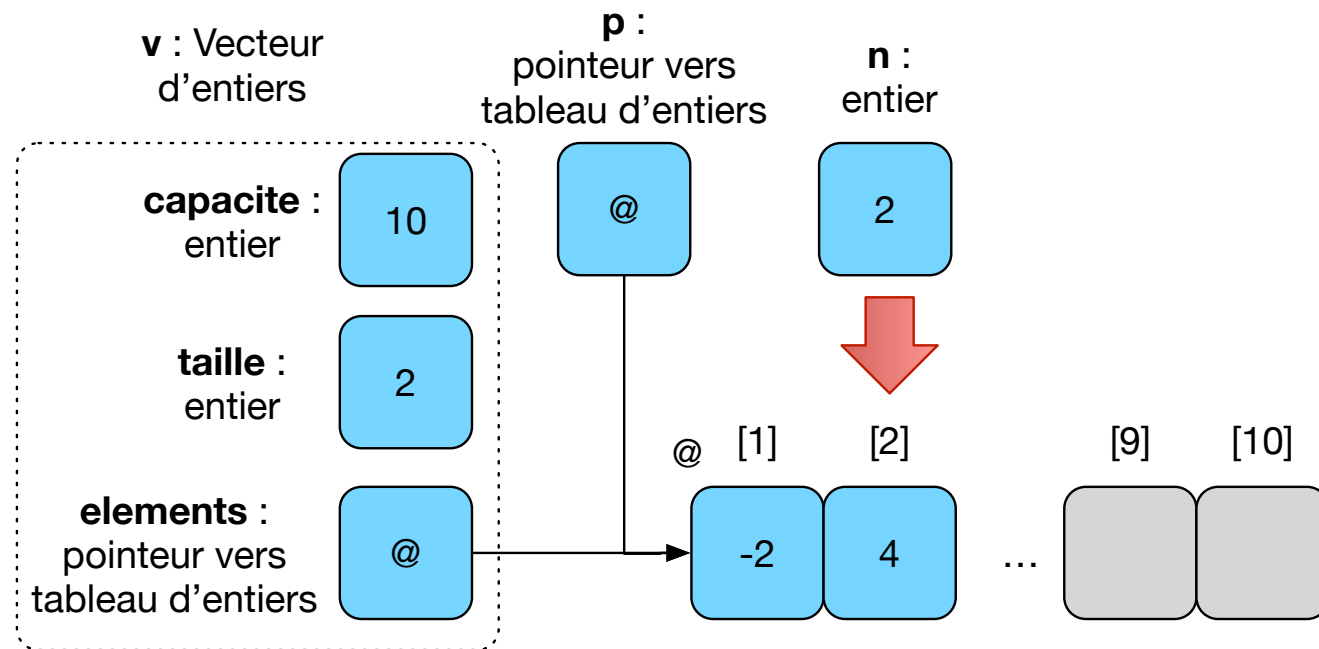
```
FONCTION lire(v : Vecteur de T, n: entier) : T
```

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

```
    p ← v.elements
```

```
    RETOURNER (p↑)[n]
```

```
FIN FONCTION
```



Exercice : écrire

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

```
    VARIABLE copie : Vecteur de T
```

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

```
    copie ← v
```

```
    p ← copie.elements
```

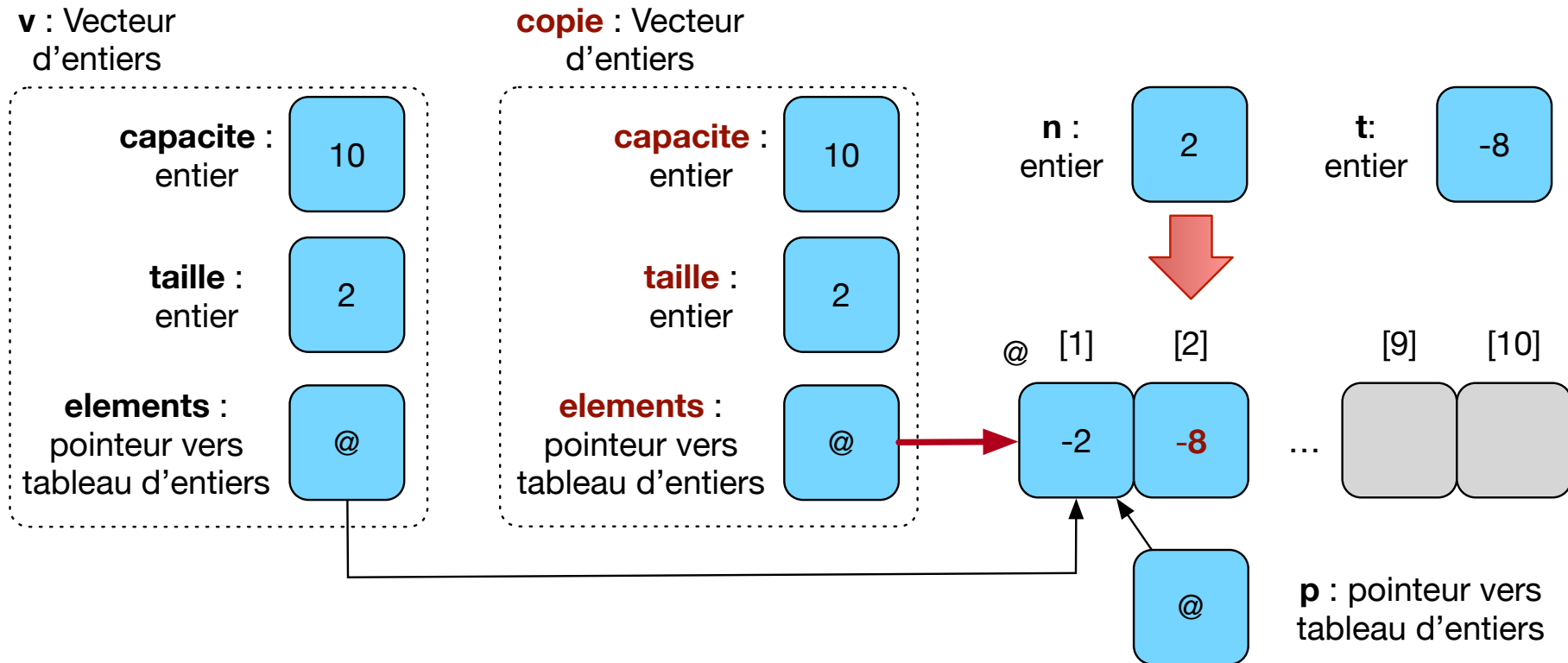
```
    (p↑)[n] ← t
```

```
    RETOURNER copie
```

```
FIN FONCTION
```

Exercice : écrire

- Copie d'enregistrement ← duplication des champs
- Copie de pointeur ← partage de référence



Exercice : écrire

- *Copie profonde Vs copie superficielle*

```
FONCTION copier(v: Vecteur de T) : Vecteur de T

    VARIABLE copie : Vecteur de T
    VARIABLE i      : entier

    copie ← v
    copie.elements = ALLouer tableau de T [v.capacite]

    POUR i DE 1 A v.taille PAR PAS DE 1
        ((copie.elements)↑)[i] ← ((v.elements)↑)[i]
    FIN POUR

    RETOURNER copie

FIN FONCTION
```

Exercice : ecrire (bis)

```
FONCTION  ecrire(v: Vecteur de T, n: entier, t: T)
           : Vecteur de T

    VARIABLE copie : Vecteur de T
    VARIABLE p    : pointeur vers tableau de T

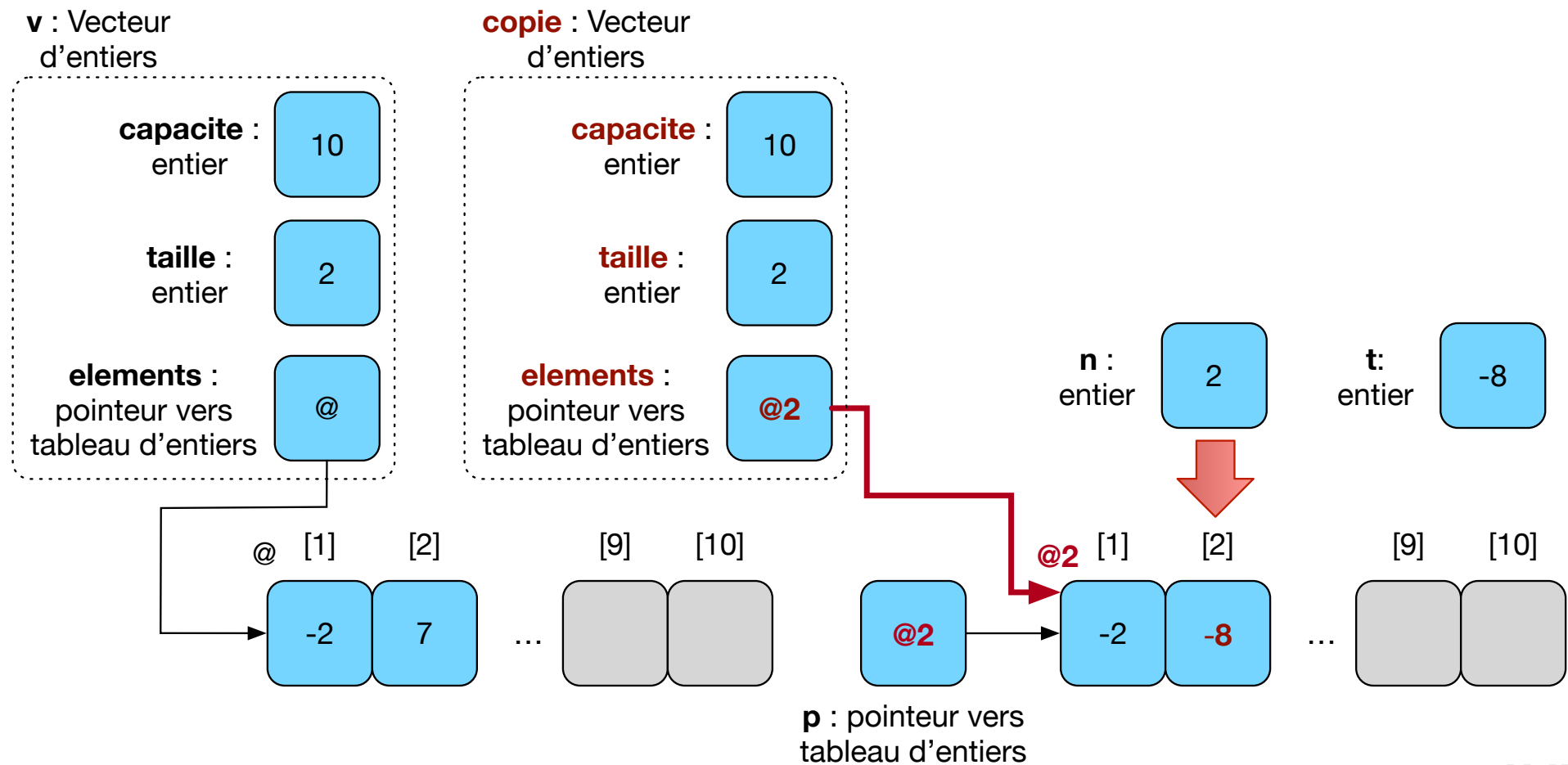
    copie ← copier(v)
    p ← copie.elements
    (p↑)[n] ← t

    RETOURNER  copie

FIN FONCTION
```

Exercice : écrire

- Copie profonde Vs copie superficielle*



Exercice : inserer

```
FONCTION  inserer(v: Vecteur de T, n: entier, t: T)
           : Vecteur de T
```

```
    VARIABLE copie    : Vecteur de T
    VARIABLE p        : pointeur vers tableau de T
    VARIABLE i        : entier
```

```
    copie ← copier(v)
    p ← copie.elements
```

```
    SI copie.taille = copie.capacite ALORS
        copie.elements =
            ALLouer tableau de T[copie.capacite*2]
    FIN SI
```

(A suivre ...)

```
FIN FONCTION
```

Exercice : insérer

```
FONCTION insérer(v: Vecteur de T, n: entier, t: T)
                : Vecteur de T
(...Suite)
```

```
POUR cpt DE 1 A n-1 PAR PAS DE 1
  ((copie.elements)↑)[cpt] ← ((v.elements)↑)[cpt]
FIN POUR
```

```
POUR cpt DE n A copie.taille PAR PAS DE 1
  ((copie.elements)↑)[cpt+1] ← ((v.elements)↑)[cpt]
FIN POUR
```

```
((copie.elements)↑)[n] ← t
copie.taille ← copie.taille + 1
libérer(vecteur.elements)
RETOURNER copie
```

```
FIN FONCTION
```


Exercice : retirer

```
FONCTION retirer(v: Vecteur de T, n: entier)
                : Vecteur de T
```

```
    VARIABLE copie    : Vecteur de T
    VARIABLE p        : pointeur vers tableau de T
    VARIABLE i        : entier
```

```
    copie ← copier(v)
    p ← copie.elements
```

```
    SI copie.taille / copie.capacite <= 0.25 ALORS
        copie.elements =
            ALLouer tableau de T[copie.capacite div 2]
    FIN SI
```

(A suivre ...)

```
FIN FONCTION
```

Exercice : retirer

```
FONCTION retirer(v: Vecteur de T, n: entier)
           : Vecteur de T
```

```
(...Suite)
```

```
POUR cpt DE 1 A n-1 PAR PAS DE 1
  ((copie.elements)↑)[cpt] ← ((v.elements)↑)[cpt]
FIN POUR
```

```
POUR cpt DE n+1 A copie.taille PAR PAS DE 1
  ((copie.elements)↑)[cpt] ← ((v.elements)↑)[cpt]
FIN POUR
```

```
copie.taille ← copie.taille - 1
liberer(vecteur.elements)
RETOURNER copie
```

```
FIN FONCTION
```

Fin !

