# Module : Structures de données et programmation C++

# Élément 1: Structures de données

**Filière : Génie Informatique, Semestre 2**

**Année Universitaire 2021-2022**

Pr. Rachid AIT DAOUD

# Chapitre 3: Les piles et les files

# Plan

- **Introduction**
- **Les piles**
  - Définitions
  - Opérations sur les piles
- **Les files**
  - Définitions
  - Opérations sur les files

# Introduction

- Les piles et les files ne sont pas de nouveaux types de données mais plutôt une **manière de gérer un ensemble de données**.

- Elles sont très souvent utiles et servent, entre autres, à **mémoriser des évènements en attente de traitement**.

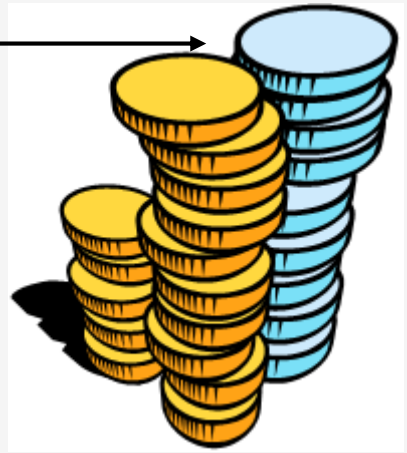
- Elles utilisent une structure de **liste chainée** pour **rassembler** les éléments.
- Les piles et les files sont **deux variantes un peu particulières** des listes chaînées. Elles permettent de contrôler **la manière** dont sont ajoutés les nouveaux éléments (Ajout au début ou à la fin).

- → Les piles et les files sont donc très utiles pour des programmes qui doivent traiter des données qui **arrivent au fur et à mesure**.

# 1. Les piles

Quand on vous dit **pile** penser directement à une **pile d'assiettes ou pile de pièces** qu'il faut manipuler avec attention pour éviter les dégâts.

Sommet →

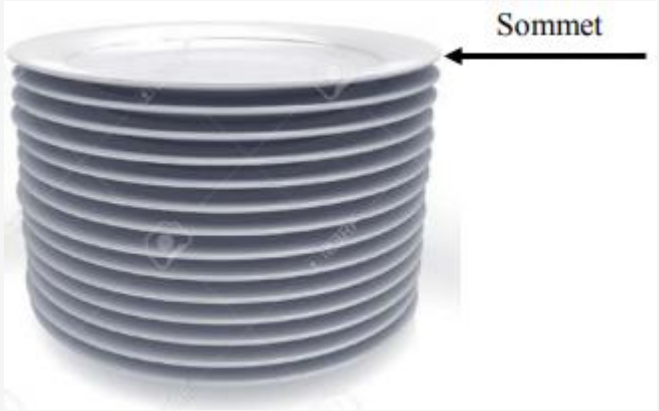



→ Une pile est un ensemble de valeurs ne permettant des insertions ou des suppressions qu'a une seule extrémité, appelée **sommet de la pile**.

# 1. Les piles

**Empiler** un objet sur une pile P consiste à **insérer** cet objet au sommet de P (dans la pile d'assiettes une nouvelle assiette ne peut être ajoutée qu'au dessus de celle qui se trouve au sommet).

**Dépiler** un objet de P consiste à **supprimer** de P l'objet placé au sommet (dans la pile d'assiettes seule peut être retirée celle qui se trouve au sommet). L'objet dépilé **est retourné comme résultat** du traitement.



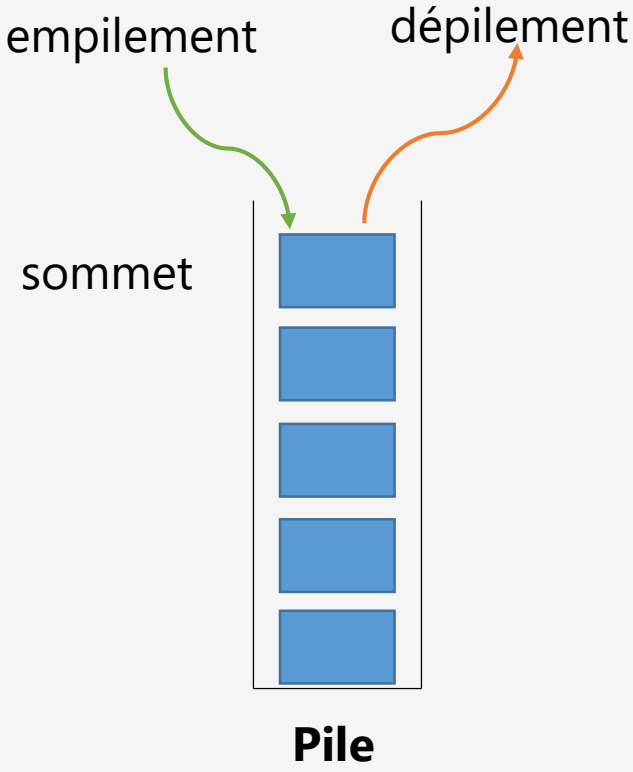
# 1. Les piles

❑ **Définition:**

Une pile (stack en anglais) est une structure dynamique dans laquelle l'insertion ou la suppression d'un élément s'effectue toujours à partir de la même extrémité de cette structure. Cette extrémité est appelée le **sommet de la pile**.

En d'autres termes:

- Une **pile** est un contenant pour des éléments insérés et retirés selon le principe **dernier entré, premier sorti** (LIFO: Last-In, First Out).
- Les éléments peuvent être insérés à tout moment, mais seulement le dernier (le plus récemment inséré) peut être retiré.
- Insérer un élément correspond à **empiler** l'élément (pushing). **Dépiler** la pile (popping) correspond au retrait d'un élément.



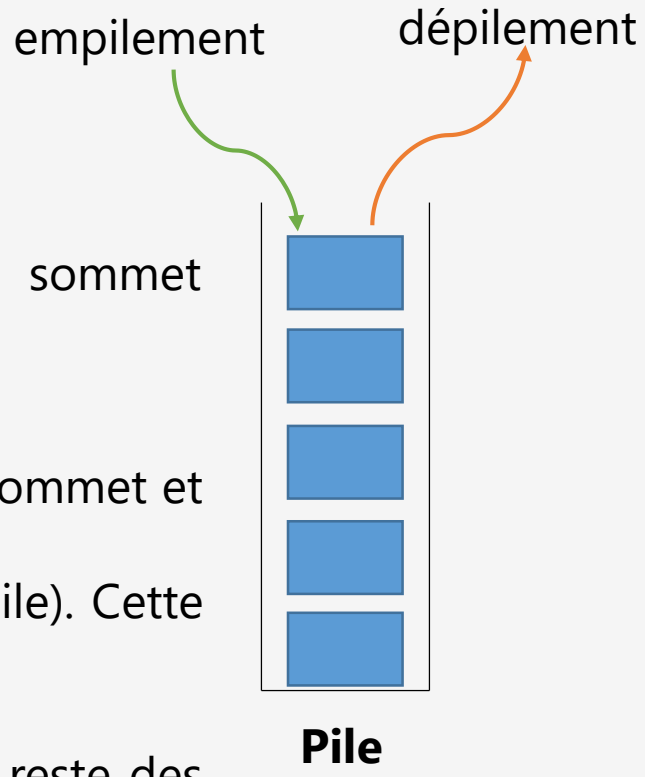
**Pile**

# 1. Les piles

## ❑ Les principales fonctions:

En termes de programmation, une pile est un enregistrement avec :

1. Une structure de données pour enregistrées les valeurs (elle peut être statique ou dynamique).
2. Une variable sommet qui indique le sommet de la pile.



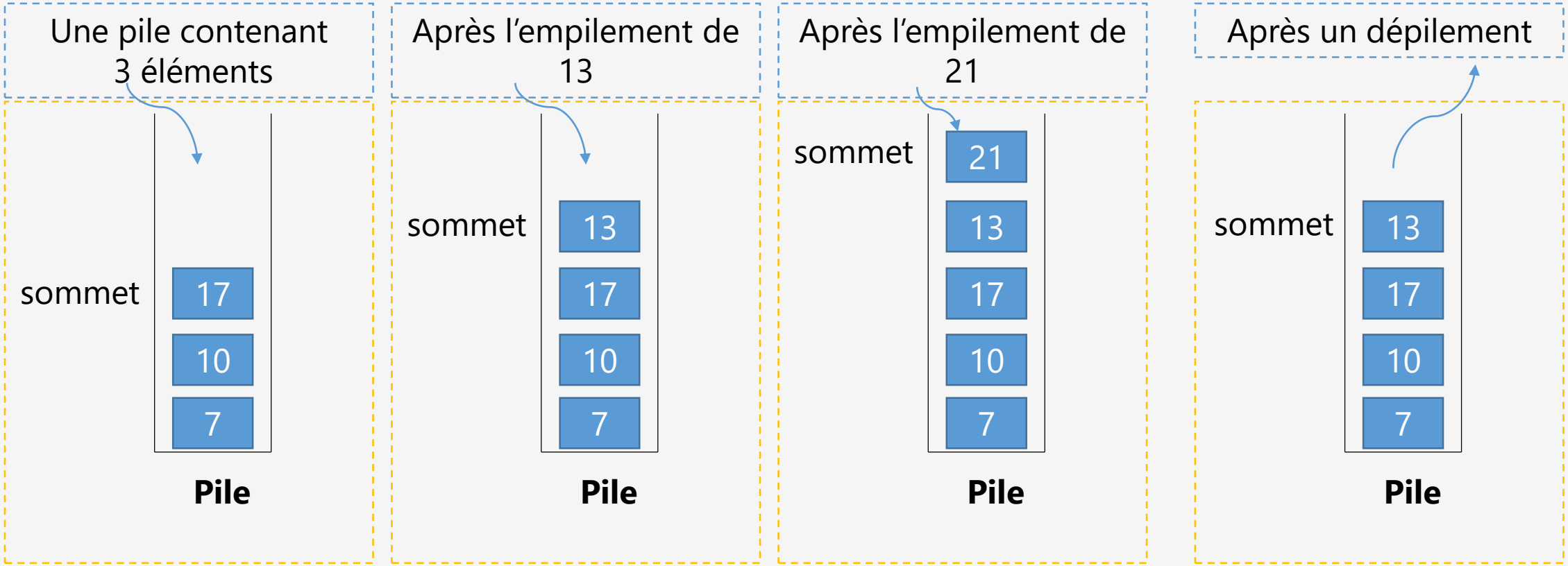

**La manipulation** d'une pile revient à l'appel de fonctions et procédures dites de bases définies une seule fois et utilisées autant de fois qu'il est nécessaire.

Les principales fonctions/Procédures associées aux piles sont:

- **Empiler():** Permet d'ajouter un nouvel élément à la pile (au dessus du sommet et dans le cas d'une pile non pleine) ;
- **Depiler():** Permet de retirer une valeur (se trouvant au sommet de la pile). Cette opération n'est possible que si la file n'est pas vide.
- **GetVal():** Retourner le sommet de la pile.
- **Pile_vide():** pour vérifier si une pile est vide ou non et savoir alors s'il reste des valeurs à traiter ou non.
- **Pile_pleine():** Pour vérifier s'il est possible de rajouter ou non un nouveau élément (utilisée dans le seul cas des piles statiques) ;

# 1. Les piles

## ❑ Exemple de pile



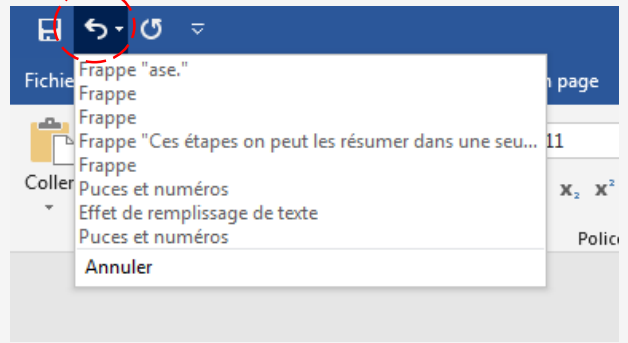
# 1. Les piles

## ❑ Utilisation

De nombreuse applications s'appuient sur l'utilisation d'une pile, on peut citer:

- Dans un navigateur web, une pile sert à mémoriser les pages Web visitées. L'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile l'adresse de la page précédente en cliquant le bouton **précédent.**


- La fonction « **Annuler la frappe** » (en anglais « Undo ») d'un traitement de texte mémorise les modifications apportées au texte dans une pile.



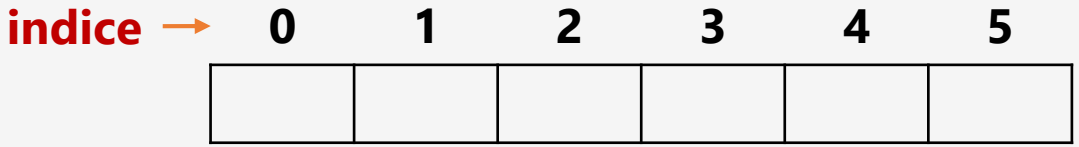

- Appel des fonctions d'un programme lors de l'utilisations des fonctions imbriquées.

→ Pour « retenir » l'ordre dans lequel les fonctions ont été appelées, l'ordinateur **crée une pile** de ces fonctions au fur et à mesure.

# 1. Les piles

## ❑ Représentation d'une pile

- **Représentation contiguë (*par tableau*) :**




  - Les éléments de la pile sont rangés dans un tableau (les éléments sont adjacents)
  - Un entier représente la position du sommet de la pile

- **Représentation chaînée (par pointeurs) :**
  - Les éléments de la pile sont chaînés entre eux par un pointeur.
  - Un pointeur sur le premier élément désigne la pile et représente le sommet de cette pile.
  - Une pile vide est représentée par le pointeur ***NULL***.

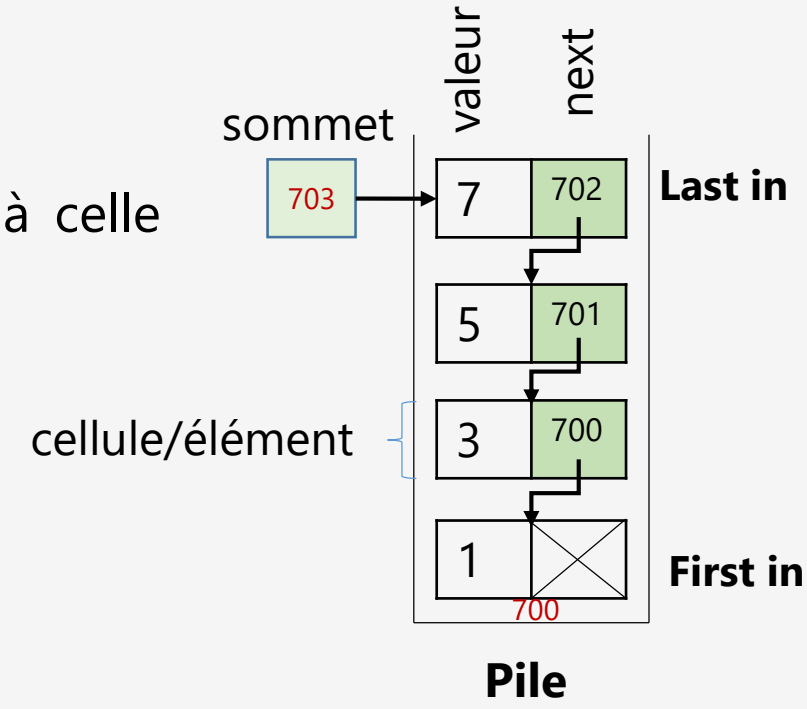
# 1. Les piles

## ❑ Représentation par une liste chainée

### Implémentation en C

- Chaque élément de la pile aura une structure identique à celle **d'une liste chainée**:

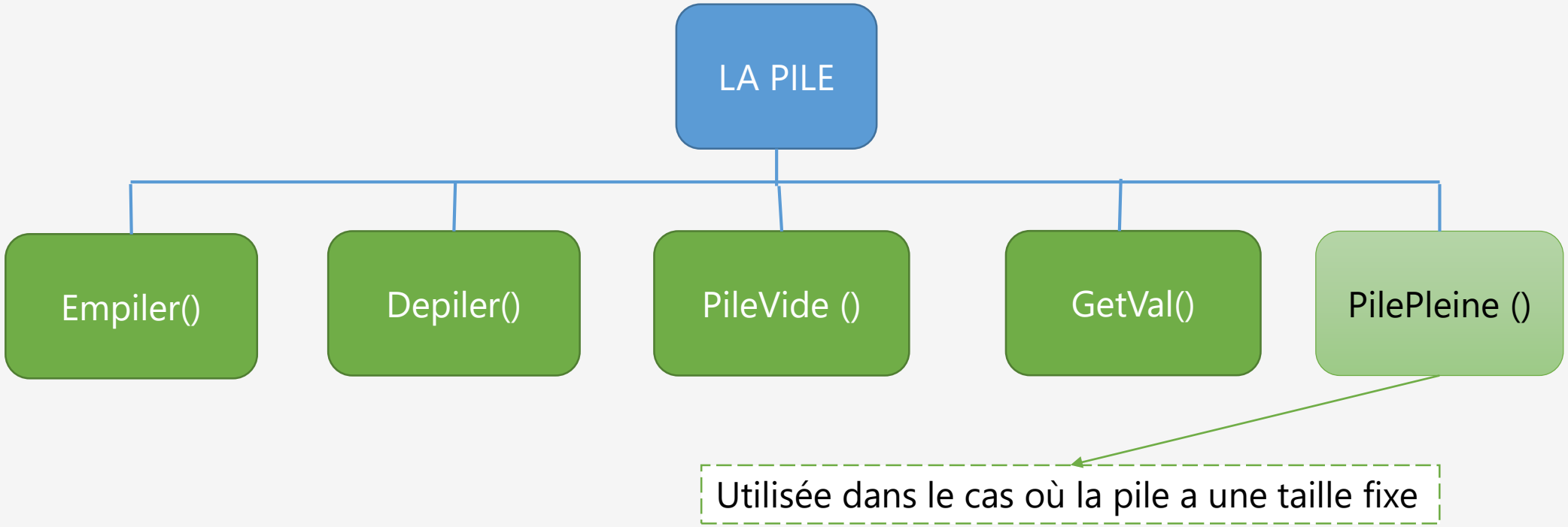
```
struct cellule {
int valeur;
struct cellule *next;
};

int main()
{
/*Pointeur sur le sommet de la pile*/
struct cellule *sommet = NULL;

    return 0;
}
```



# 1. Les piles

- **Représentation par une liste chainée**

## Implémentation en C



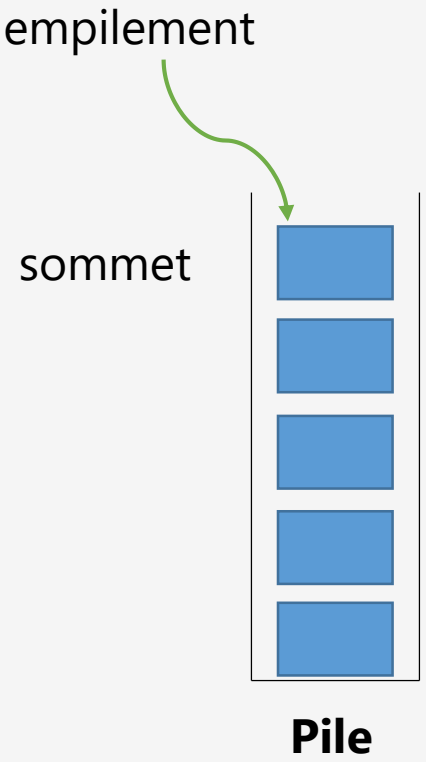
# 1. Les piles

- **Représentation par une liste chainée**

## Implémentation en C

→ Empiler: Ajout un nouveau élément au sommet de la pile.

```
void Empiler(struct cellule **sommet, int val)
{
    struct cellule *newcellule = malloc(sizeof(struct cellule));
    newcellule->valeur = val;
    newcellule->next = *sommet;
    *sommet = newcellule;
}
```



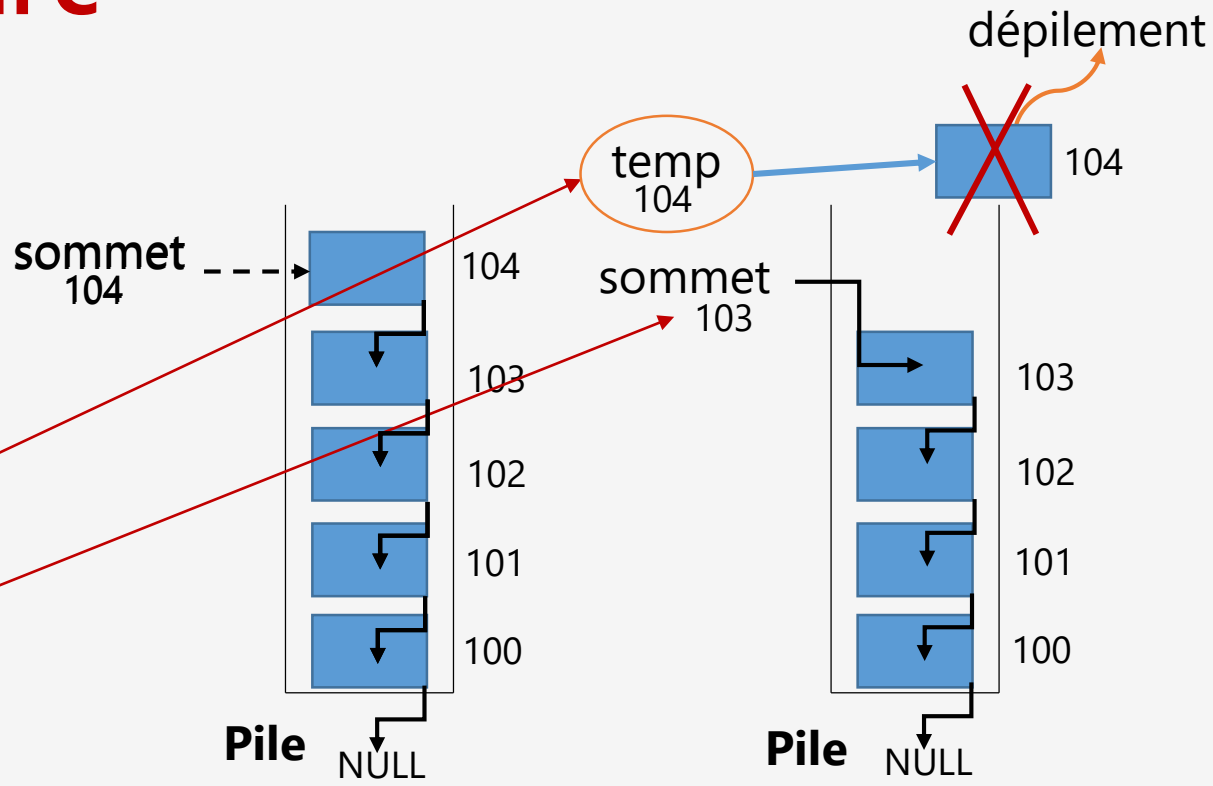
# 1. Les piles

## ❑ Représentation par une liste chainée

## Implémentation en C

→ Dépiler: Retirer le sommet

```
////////////estVide////////////
int PileVide(struct cellule *sommet){
if (sommet == NULL) return 1;
return 0;
}
////////////Depiler////////////
void Depiler(struct cellule **sommet){
    if(!PileVide(*sommet))
        {
            struct cellule *temp = *sommet;
            *sommet = (*sommet)->next;
            free(temp);
        }
    else printf("La pile est vide !!\n");
}
```



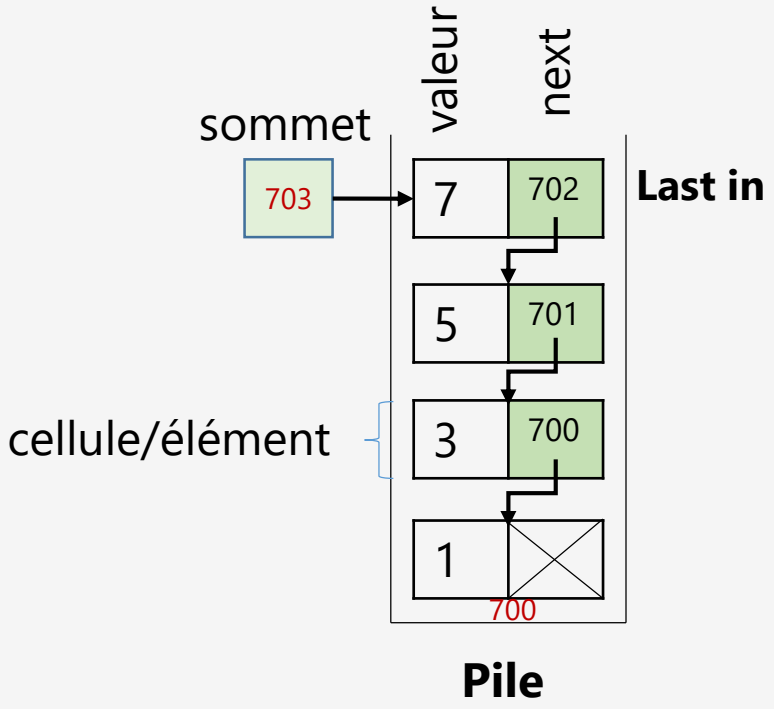
# 1. Les piles

- **Représentation par une liste chainée**

## Implémentation en C

→ GetVal: Retourner le contenu du sommet de la pile:

```
int getVal(struct cellule *sommet){
    if(!PileVide(sommet))
    return sommet->valeur; // la valeur 7
    return -1;
}
```



# 1. Les piles

## Implémentation en C

→ Fonction principale main()

```c
int main()
{
/*Pointeur sur le sommet de la pile*/
struct cellule *sommet = NULL;//@699
Empiler(&sommet,1); //creer et ajouter le 1er element de la pile
Empiler(&sommet,3); //creer et ajouter le 2eme element de la pile
Empiler(&sommet,5); //creer et ajouter le 3eme element de la pile
Empiler(&sommet,7); //creer et ajouter le dernier element de la pile
printf("%d\n",getVal(sommet));
Depiler(&sommet);
printf("%d\n",getVal(sommet));
Depiler(&sommet);
printf("%d\n",getVal(sommet));
Depiler(&sommet);
printf("%d\n",getVal(sommet));
printf("----------------\n");
Depiler(&sommet);
printf("%d\n",getVal(sommet));
Depiler(&sommet);
printf("%d\n",getVal(sommet));
Depiler(&sommet);
printf("%d\n",getVal(sommet));
return 0;
}
```

```
7
5
3
1
---------------
-1
La pile est vide !!
-1
La pile est vide !!
-1

Process returned 0 (0x0)
Press any key to continue.
```

# 2. Les files

Quand on vous dit **file** penser directement à une file d'attente où chacun à son tour qu'il doit respecter.


Une file est un ensemble de valeurs qui a un début (Debut) et une fin (Queue).

**Enfiler** un objet sur une file F consiste à **insérer** cet objet à la fin de la file F (dans la file d'attente un nouvel arrivant se met à la queue c.-à-d., après la personne arrivée juste avant lui) ;

**Défiler** un objet de F consiste à supprimer de F l'objet placé en début de file (dans la file d'attente seule peut être servie la personne qui se trouve en début de file). L'objet défilé est **retourné comme résultat** du traitement.

# 2. Les files

## ❑ Définition:

Une File (queue en anglais ) est une structure de données dans laquelle **l'insertion se fait à la fin** et **la suppression d'un élément s'effectue à partir de début** de cette structure.
Le fonctionnement ressemble à une file d'attente: les premières personnes arrivées, se sont les premières personnes à servir.

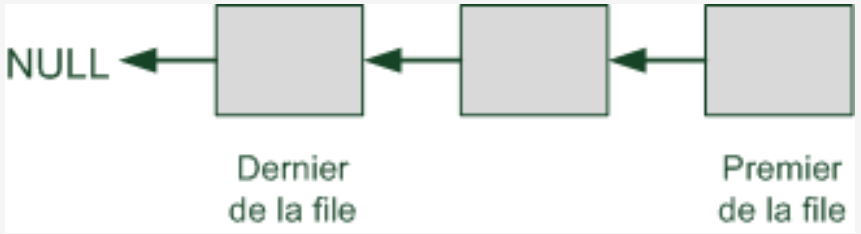
# 2. Les files

## ❑ Principe de fonctionnement:

- Un élément ne peut être ajouté qu'à la queue de la file.
- Un élément ne peut être retiré qu'à la tête de la file.
- Il s'agit donc, d'une structure de type **FIFO** (FIFO: First In, First Out). **"Le premier qui arrive est le premier à sortir".**

  → Les données sont retirées dans l'ordre où elles ont été ajoutées.
- Insérer un élément correspond à **enfiler** l'élément. **Défiler** la file correspond au retrait de l'élément situé au début de la file.

En C, une file est une liste chaînée où chaque élément pointe vers le suivant, tout comme les piles. Le dernier élément de la file pointe vers ***NULL***



# 2. Les files

❑ **Exemple de file:**

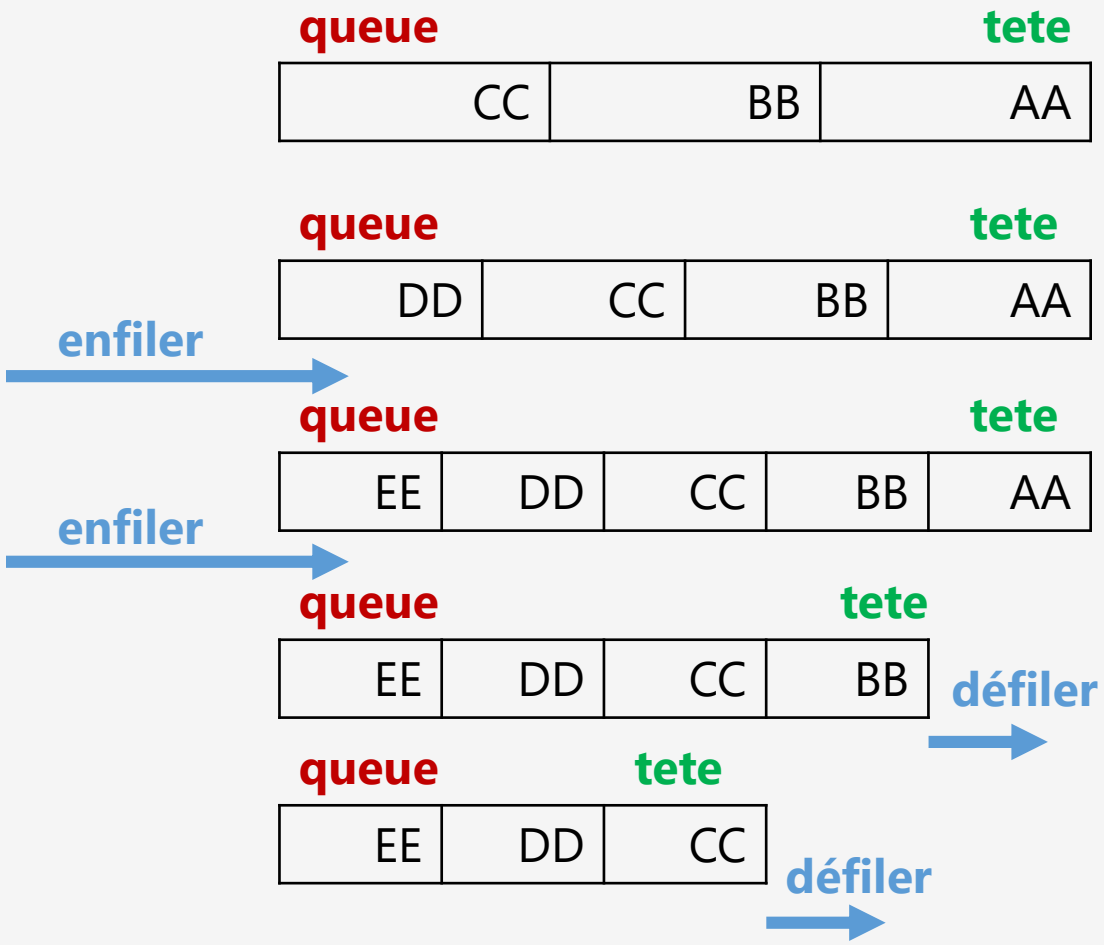
Une file contenant 3 éléments

Après l'enfilement de DD

Après l'enfilement de EE

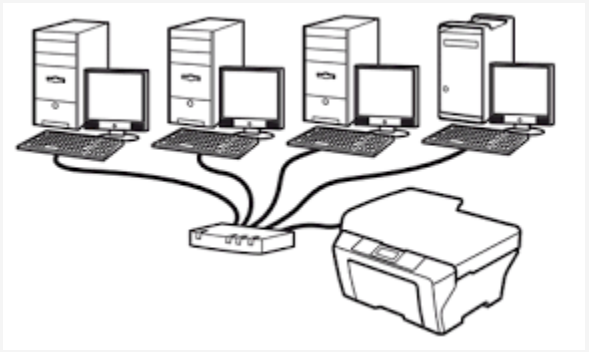
Après un défilement

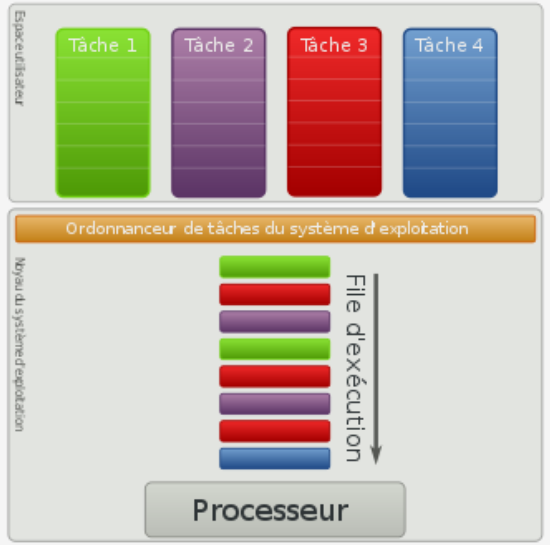
Après un 2ème défilement



# 2. Les files

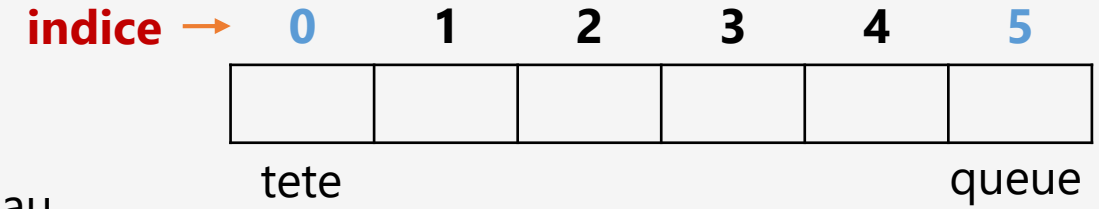
- **Utilisation:**
  - En général, on utilise des files pour mémoriser temporairement des transactions qui doivent attendre pour être traitées.
  - Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente ( ou une queue).
  - Ordonnanceur de tâches du système d'exploitation (file d'exécution des tâches, sans en privilégier aucune).
  - Construire des systèmes de réservation.
  - Le routage de paquets réseau.



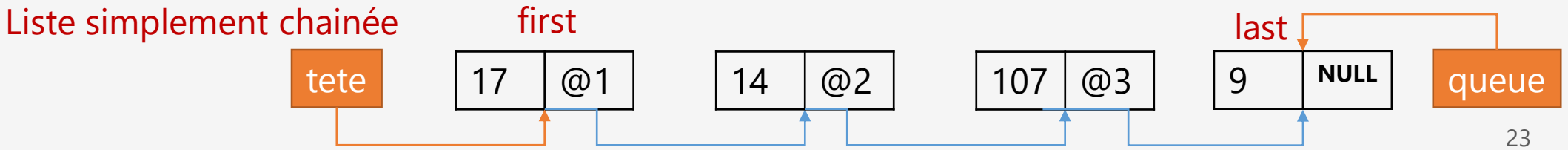


# 2. Les files

## ❑ Représentation d'une file




- **Représentation contiguë (*par tableau*) :**
  - Les éléments de la file sont rangés dans un tableau.
  - Deux entiers représentent respectivement les positions de la tête et de la queue de la file
- **Représentation chaînée (par pointeurs) :**
  - Les éléments de la file sont chaînés entre eux par pointeurs.
  - Un pointeur sur le premier élément désigne la file et représente la tête de cette file.
  - Un pointeur sur le dernier élément représente la queue de file.
  - Une file vide est représentée par le pointeur ***NULL***.



# 2. Les files

## ❑ Représentation par une liste chainée

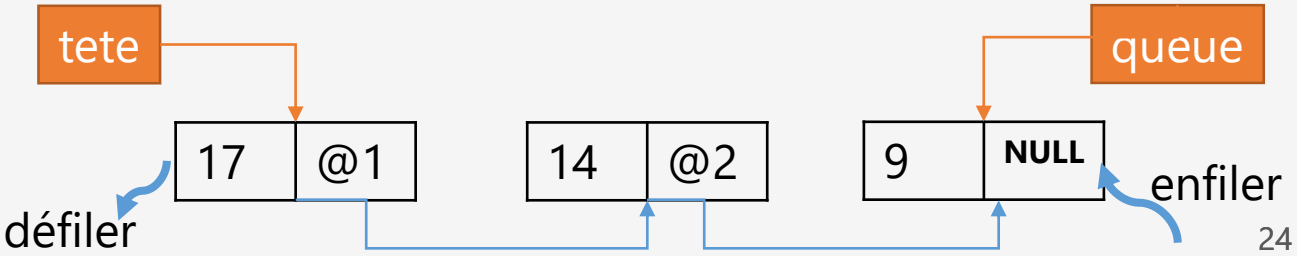
## Implémentation en C

Afin de présenter une file par une liste chainée, nous allons besoin de définir deux structures:

- Une structure **cellule_file** qui représente un élément de la file.
- Une structure **file** qui représente la file.

```
struct cellule_file {
int valeur;
struct cellule_file *next;
};
struct file {
struct cellule_file *tete;
struct cellule_file *queue;
};
```

- ✓ Une file vide --> tete = queue = NULL
- ✓ Une file a un seul élément --> tete = queue ≠ NULL
- ✓ Une file a plus d'un élément --> tete ≠ queue



- **Représentation par une liste chainée**

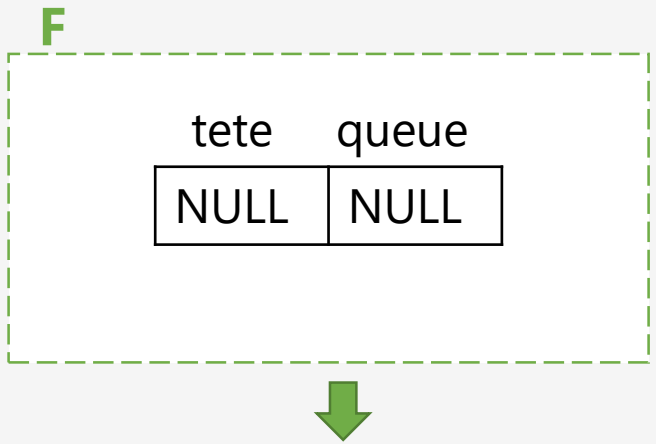
## Implémentation en C

**Déclaration d'une file:**

----------**Statique**----------

**struct file F;**

**F.tete = NULL;**
**F.queue = NULL;**




Une file vide --> tete = queue = NULL

# 2. Les files

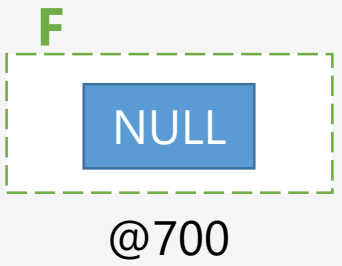
- **Représentation par une liste chainée**

## Implémentation en C

**Déclaration d'une file:**

----------**Dynamique**----------

**struct file *F = NULL;**




La file n'est pas encore créée (n'existe plus dans la mémoire), on a juste un pointeur qui va recevoir l'adresse d'une file.

# 2. Les files

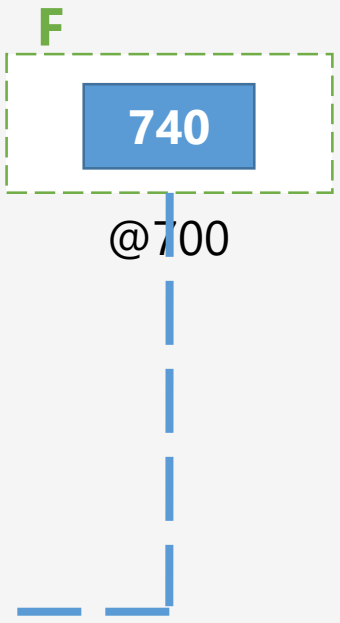
- **Représentation par une liste chainée**

## Implémentation en C

**Déclaration d'une file:**

----------**Dynamique**----------

**struct file *F = NULL;**




**F = malloc(sizeof(struct file);**

Allocation de mémoire par malloc, et affectation de l'adresse de la file vers le pointeur F.

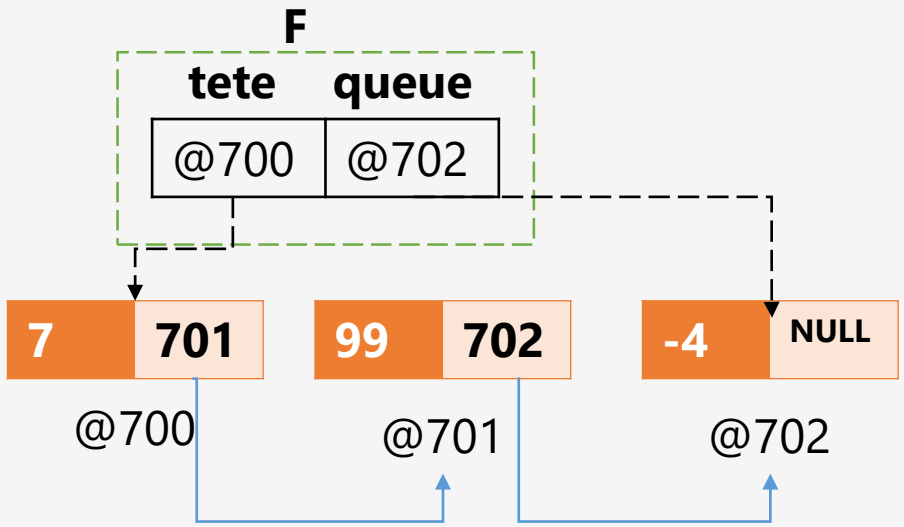
**F->tete = NULL;**
**F->queue = NULL;**

- **Représentation par une liste chainée**

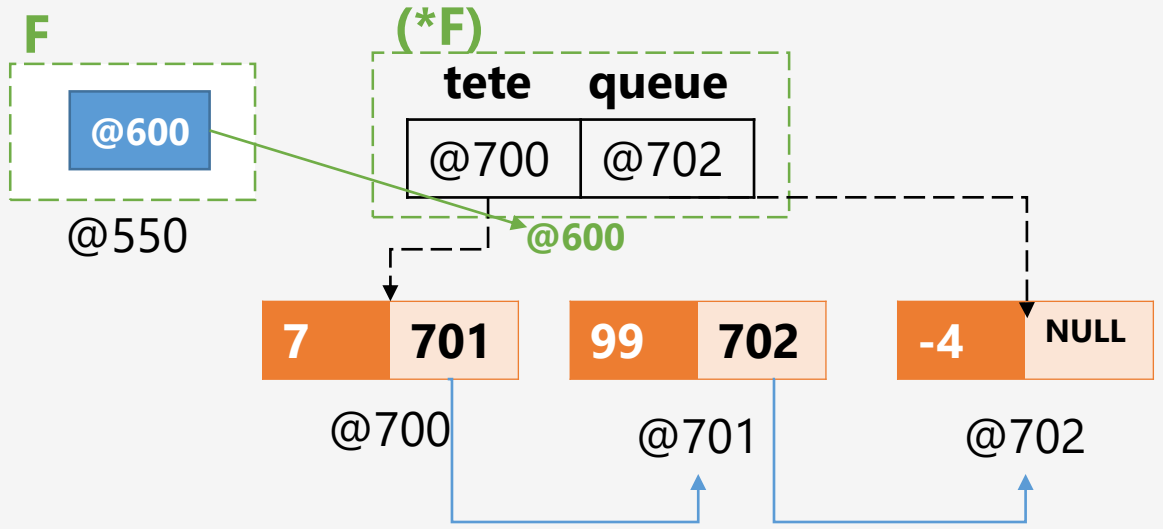
# Implémentation en C

**Déclaration d'une file:**

**Déclaration statique:**



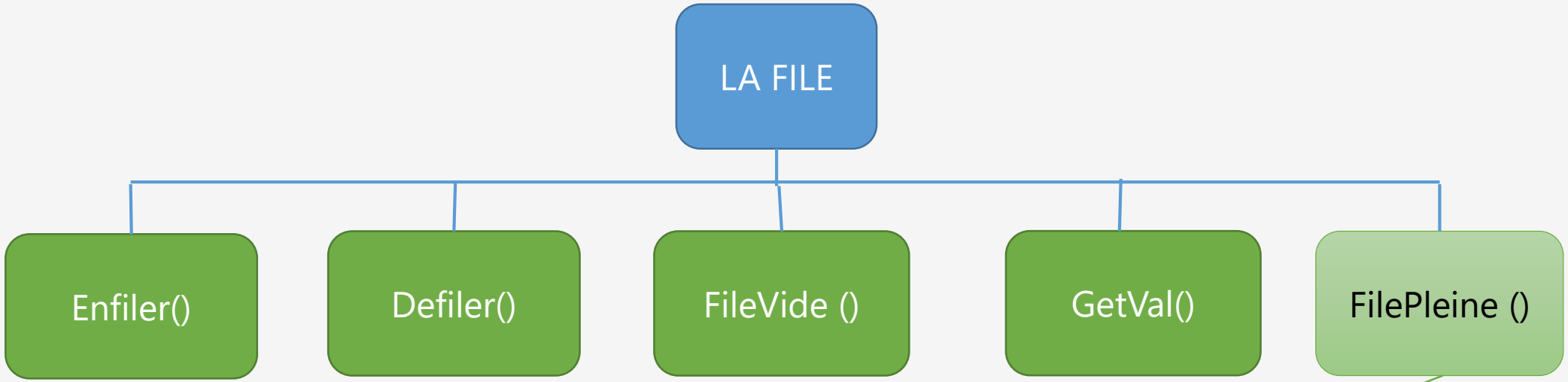

**Déclaration dynamique:**



# 2. Les files

- **Représentation par une liste chainée**

## Implémentation en C




**Liste des fonctions/procédures autorisées pour la manipulation d'une file.**

Utilisée dans le cas où la file a une taille fixe

# 2. Les files

## ❑ Représentation par une liste chainée

## Implémentation en C

→ Enfiler: Ajout un nouveau élément au queue de la file.

Deux cas de figure:

1. File vide → **F->tete == NULL**
2. File non vide → **F->tete != NULL**

```c
int main()
{
struct file *f = NULL;
f=malloc(sizeof(struct file));
f->tete = f->queue = NULL;
enfiler(&f,8);
enfiler(&f,6);
enfiler(&f,4);
return 0;
}
```

```c
void enfiler(struct file **f, int val)
{
    struct cellule_file *newcellule = malloc(sizeof(struct cellule_file));
    newcellule->valeur = val;
    newcellule->next = NULL;
    if((*f)->tete == NULL)/* si la file est vide*/
    {
        (*f)->tete = (*f)->queue = newcellule;

    } else
    {
        (*f)->queue->next = newcellule; //Ancienne dernière cellule
        (*f)->queue = newcellule;
    }
}
```

# 2. Les files

## ❑ Représentation par une liste chainée

## Implémentation en C

→ Défiler: Retirer un élément

Trois cas de figure se présentent:

1. File vide ➔ **F->tete == NULL**
2. File contient seul élement ➔ **F->tete == F->queue**
3. **File contient plusieur élements ➔ F->tete != F->queue**

```
void defiler(struct file **F){
if((*F)->tete == NULL) printf("Echec, la file est deja vide!!\n");
struct file *temp = (*F)->tete;
else if ((*F)->tete == (*F)->queue) {

(*F)->tete = (*F)->queue == NULL;

}
else {
(*F)->tete = (*F)->tete->next;
}
free(temp);
}
```