Les Piles & Files (Stacks)

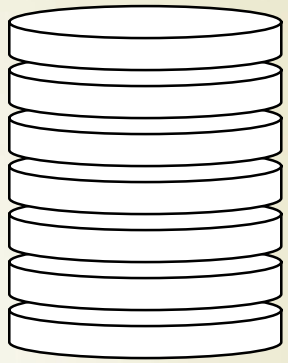
Pr F.Omary

FSR-Université MohammedV

2019-2020

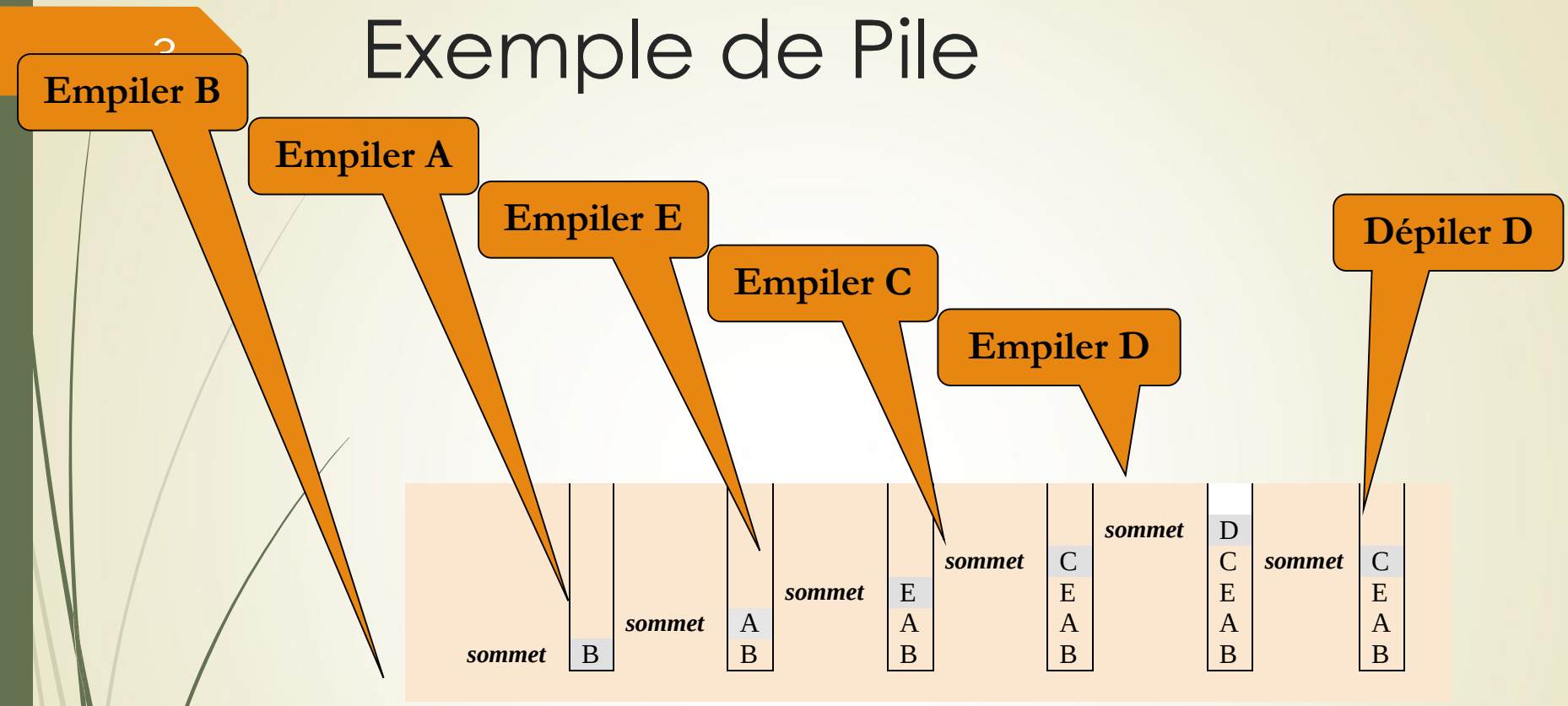
Notion de Pile (Stack)

- Les piles sont très utilisées en informatique



- Notion intuitive :
  - pile d'assiettes, pile de dossiers à traiter, …
- Une pile est une structure linéaire permettant de stocker et de restaurer des données selon un ordre LIFO (Last In, First Out ou « dernier entré, premier sorti »)
- Dans une pile :
  - Les insertions (empilements) et les suppressions (dépilements) sont restreintes à une extrémité appelée sommet de la pile.

Exemple de Pile



Type Abstrait Pile

Type Pile
Utilise Elément, Booléen
Opérations
  pile_vide : → Pile
  est_vide : Pile → Booléen
  empiler : Pile x Elément → Pile
  dépiler : Pile → Pile
  sommet : Pile → Elément
Préconditions
  dépiler(p) *est-défini-ssi* est_vide(p) = faux
  sommet(p) *est-défini-ssi* est_vide(p) = faux
Axiomes
  Soit, e : Element, p : Pile
  est_vide(pile_vide) = vrai
  est_vide(empiler(p,e)) = faux
  dépiler(empiler(p,e)) = p
  sommet(empiler(p,e)) = e

Opérations sur une Pile

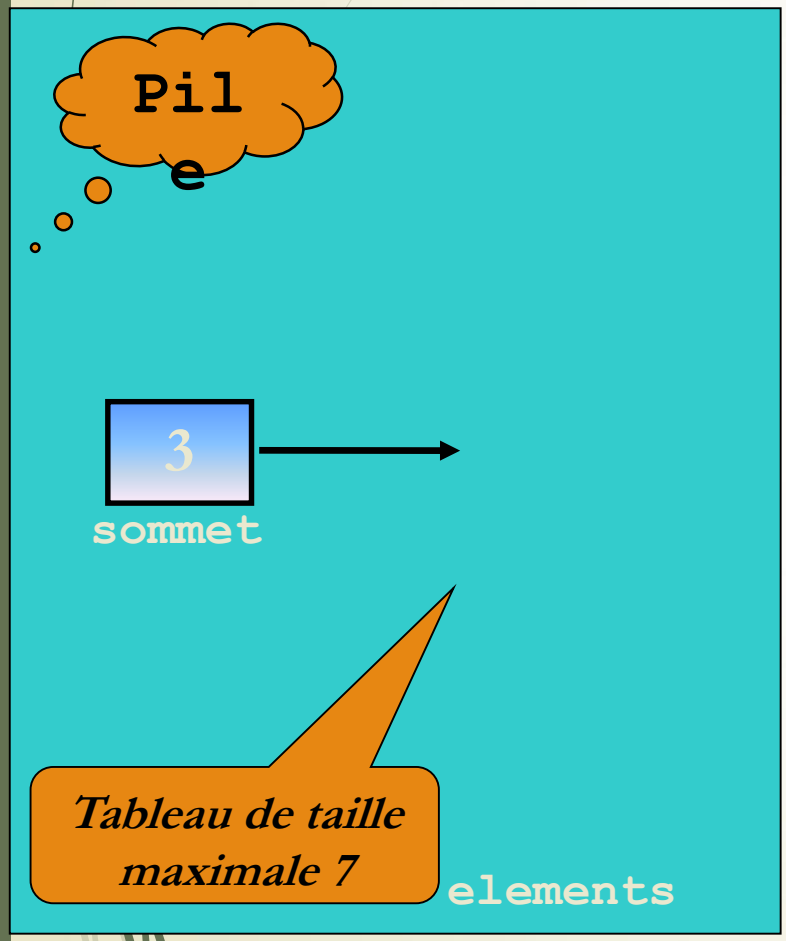
- **`pile_vide : → Pile`**
  - opération d'initialisation ; *la pile créée est vide*
- **`est_vide : Pile → Booléen`**
  - teste si pile vide ou non
- **`sommet : Pile → Elément`**
  - permet de consulter l'élément situé au sommet ; *n'a pas de sens si pile vide*
- **`empiler : Pile x Elément → Pile`**
  - ajoute un élément dans la pile
- **`dépiler : Pile → Pile`**
  - enlève l'élément situé au sommet de la pile ; *n'a pas de sens si pile vide*

Représentation d'une Pile

- **Représentation contiguë (*par tableau*) :**
  - *Les éléments de la pile sont rangés dans un tableau*
  - *Un entier représente la position du sommet de la pile*

- **Représentation chaînée (par pointeurs) :**
  - *Les éléments de la pile sont chaînés entre eux (voir listes chaînées)*
  - *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`

Pile Contiguë




```
    /* Pile contiguë en C */

// taille maximale pile
#define MAX_PILE 7

// type des éléments
typedef int Element;

// type Pile
typedef struct {
 Element elements[MAX_PILE];
 int sommet;
} Pile;
```

Spécification d'une Pile Contiguë

```
/* fichier "Tpile.h" */
#ifndef _PILE_TABLEAU
#define _PILE_TABLEAU

#include "Booleen.h"
// Définition du type Pile (implémentée par un tableau)
#define MAX_PILE 7  /* taille maximale d'une pile */
typedef int Element;  /* les éléments sont des int */

typedef struct {
 Element elements[MAX_PILE]; /* les éléments de la pile */
 int sommet;  /* position du sommet */
} Pile;
// Déclaration des fonctions gérant la pile
Pile pile_vide ();
Pile empiler ( Pile p, Element e );
Pile depiler ( Pile p );
Element sommet ( Pile p );
Booleen est_vide ( Pile p );

#endif
```

type Pile : une structure à deux champs

Réalisation d'une Pile Contiguë

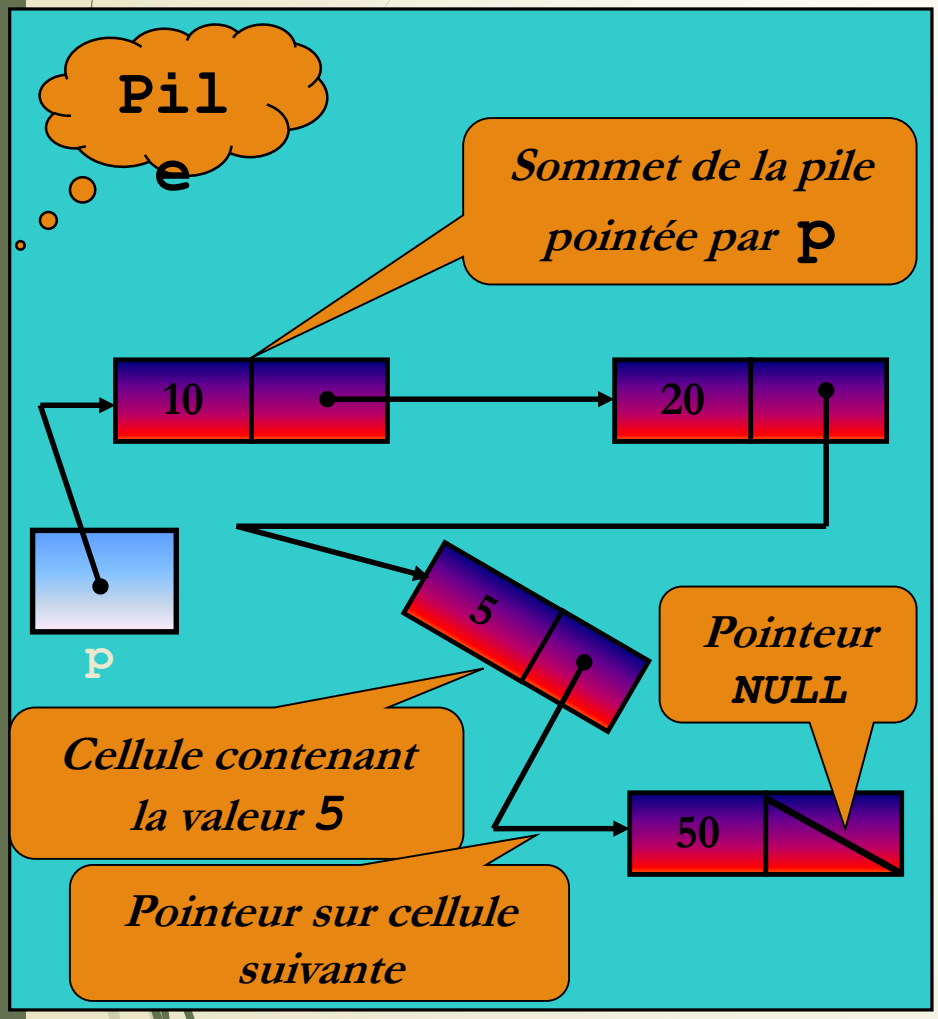

```
/* fichier "Tpile.c" */

#include "Tpile.h"
// Définition des fonctions gérant la pile
// initialiser une nouvelle pile
 Pile pile_vide() {
 Pile p;
 p.sommet = -1;
 return p;
}
// tester si la pile est vide
Booleen est_vide(Pile p) {
 if (p.sommet == -1) return vrai;
 return faux;
}
// Valeur du sommet de pile
Element sommet(Pile p) {
/* pré-condition : pile non vide ! */
 if (est_vide(p)) {
  printf("Erreur: pile vide !\n");
  exit(-1);
 }
 return (p.elements)[p.sommet];
}
```

```
// ajout d'un élément
Pile empiler(Pile p, Element e) {
 if (p.sommet >= MAX_PILE-1) {
  printf("Erreur : pile pleine !\n");
  exit(-1);
 }
 (p.sommet)++;
 (p.elements)[p.sommet] = e;
 return p;
}

// enlever un élément
Pile depiler(Pile p) {
/* pré-condition : pile non vide !*/
 if (est_vide(p)) {
  printf("Erreur: pile vide !\n");
  exit(-1);
 }
 p.sommet--;
 return p;
}
```

Exemple d'Utilisation d'une Pile Contiguë


```
                    /* fichier "UTpile.c" */
#include <stdio.h>
#include "Tpile.h"

int main () {
 Pile p = pile_vide();

 p = empiler(p,50);
 p = empiler(p,5);
 p = empiler(p,20);
 p = empiler(p,10);
 printf("%d au sommet après empilement de 50, 5, 20 et"
        " 10\n", sommet(p));
 p = depiler(p);
 p = depiler(p);
 printf("%d au sommet après dépilement de 10 et 20\n",
        sommet(p));
 return 0;
}
```

Pile chaînée




```
   /* Pile chaînée en C */

// type des éléments
typedef int element;

// type Cellule
typedef struct cellule {
   element valeur;
   struct cellule *suivant;
 } Cellule;

// type Pile
typedef Cellule *Pile;
```

Complexité

- **Les opérations sur les piles sont toutes en O(1)**
- **Ceci est valable pour les deux représentations proposées**

Applications d'une Pile Exemples (1)

- **Vérification du bon équilibrage d'une expression parenthèsée :**
  - Pour vérifier qu'une expression parenthèsée est équilibrée, à chaque rencontre d'une parenthèse ouvrante on l'empile et à chaque rencontre d'une parenthèse fermante on dépile ;
- **Evaluation des expressions arithmétiques postfixées (*expressions en notation polonaise inverse*) :**
  - Pour évaluer une telle expression, on applique chaque opérateur aux deux opérandes qui le précédent. Il suffit d'utiliser une pile dans laquelle les opérandes sont empilés, alors que les opérateurs dépilent deux éléments, effectuent l'opération et empilent le résultat. Par exemple, l'expression postfixée ***2 3 5 * + 1 –*** s'évalue comme suit : ***((2 (3 5 *) +) 1 –) = 16*** ;
- **Conversion d'une expression en notation infixe (*parenthèsée*) en notation postfixée ;**

Applications d'une Pile
Exemples (2)

- **Gestion par le compilateur des appels de fonctions :**
  - les paramètres, l'adresse de retour et les variables locales sont stockés dans la pile de l'application
- **Mémorisation des appels de procédures imbriquées au cours de l'exécution d'un programme, et en particulier les appels des procédures récursives ;**
- **Parcours « en profondeur » de structures d'arbres** (***voir arbres***) ;

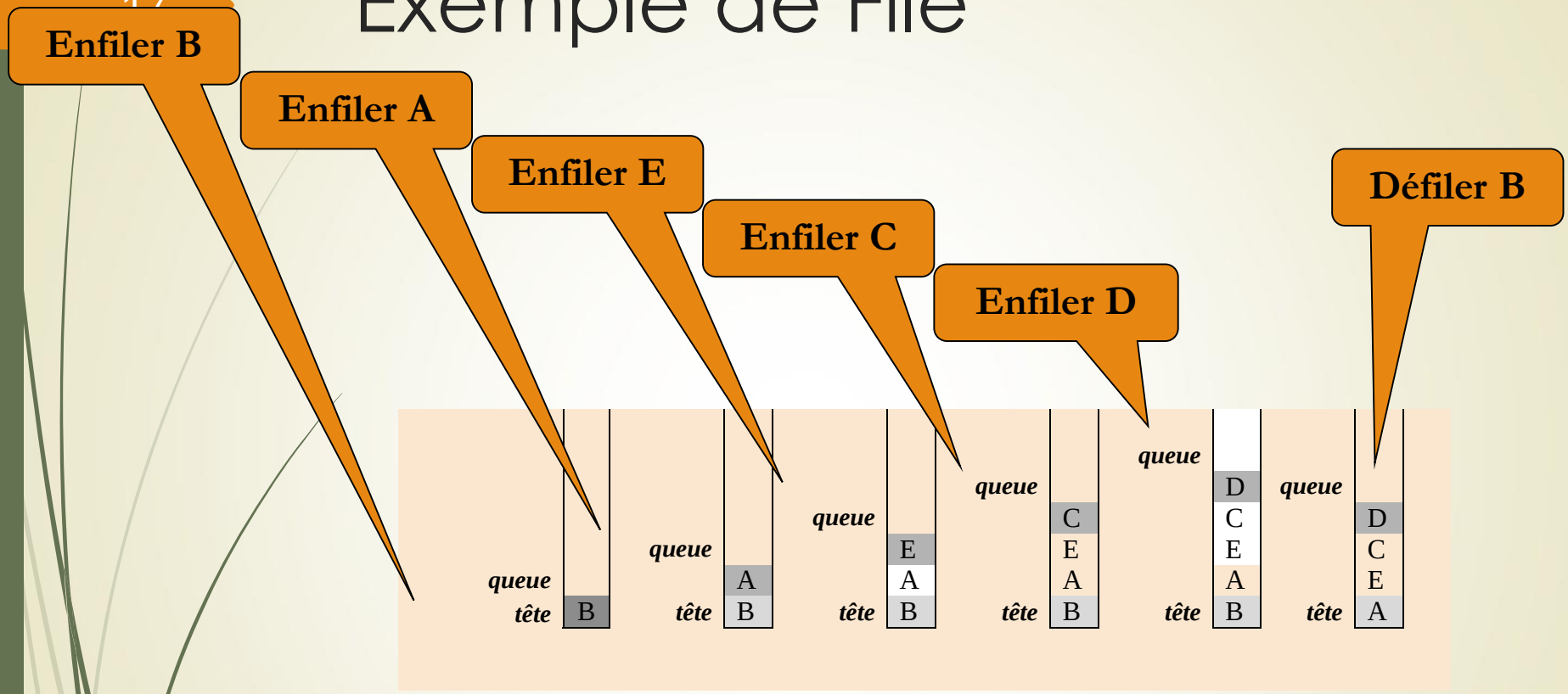
Les Files (Queues)

Notion de File (Queue)

- **Les files sont très utilisées en informatique**
- **Notion intuitive :**
  - File d'attente à un guichet, file de documents à imprimer, …

- **Une file est une structure linéaire permettant de stocker et de restaurer des données selon un ordre FIFO (*First In, First Out* ou « premier entré, premier sorti »)**
- **Dans une file :**
  - Les insertions (***enfilements***) se font à une extrémité appelée ***queue*** de la file et les suppressions (***défilements***) se font à l'autre extrémité appelée ***tête*** de la file

Exemple de File



Type Abstrait File


```
Type File
Utilise Elément, Booléen
Opérations
    file_vide    : → File
    est_vide     : File → Booléen
    enfiler  : File x Elément → File
    défiler  : File → File
    tête     : File → Elément
Préconditions
    défiler(f) est-défini-ssi est_vide(f) = faux
    tête(f) est-défini-ssi est_vide(f) = faux
Axiomes
    Soit, e : Element, f : File
    est_vide(file_vide) = vrai
    est_vide(enfiler(f,e)) = faux
    si est_vide(f) = vrai alors tête(enfiler(f,e)) = e
    si est_vide(f) = faux alors tête(enfiler(f,e)) = tête(f)
    si est_vide(f) = vrai alors défiler(enfiler(f,e)) = file_vide
    si est_vide(f) = faux
      alors défiler(enfiler(f,e)) = enfiler(défiler(f),e)
```

Opérations sur une File

- **`file_vide : → File`**
  - opération d'initialisation ; *la file créée est vide*
- **`est_vide : File → Booléen`**
  - teste si file vide ou non
- **`tête : File → Elément`**
  - permet de consulter l'élément situé en tête de file ; *n'a pas de sens si file vide*
- **`enfiler : File x Elément → File`**
  - ajoute un élément dans la file
- **`défiler : File → File`**
  - enlève l'élément situé en tête de file ; *n'a pas de sens si file vide*

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 (voir listes chaînées)*
  - *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`

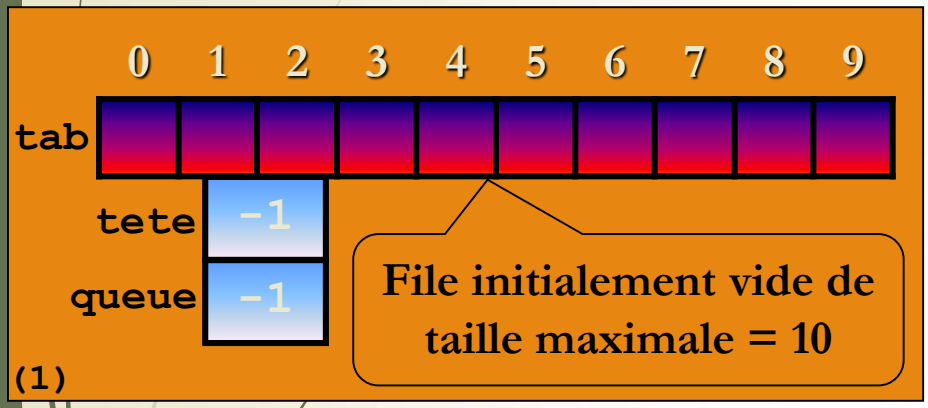
Représentation Contiguë d'une File (par tableau simple)

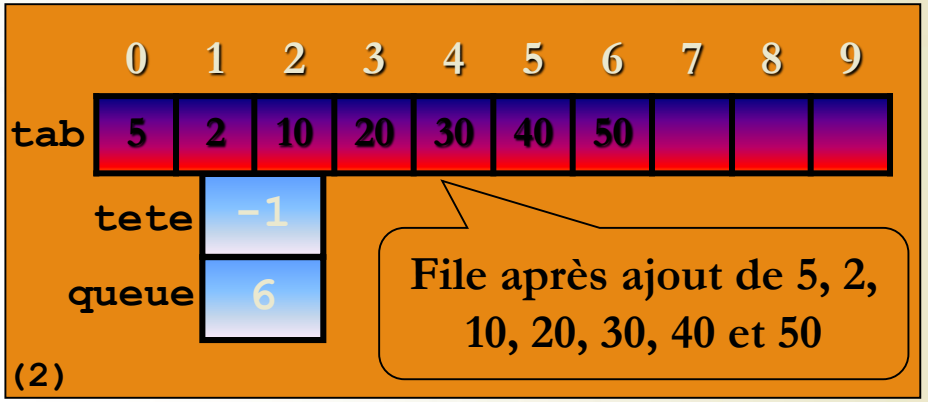
- ***tête de file :*** position précédant premier élément
- ***queue de file :*** position du dernier élément

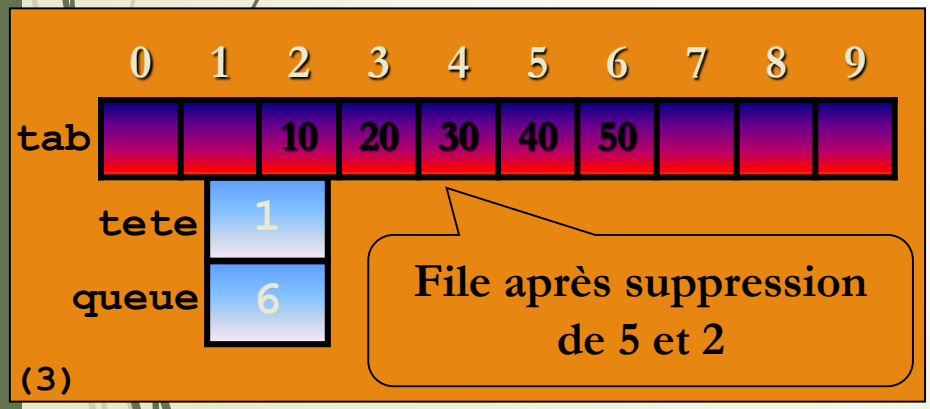
- ***Initialisation :*** `tête ← queue ← -1`

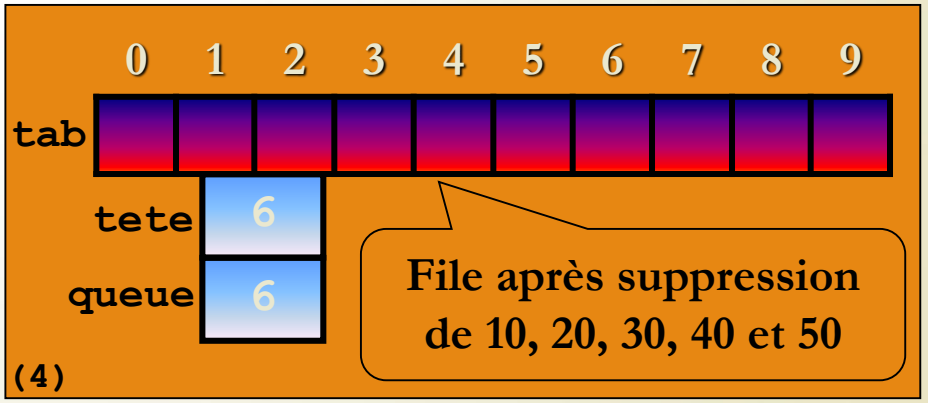
- ***Inconvénient :*** on ne peut plus ajouter des éléments dans la file, alors qu'elle n'est pas pleine !

Représentation Contiguë d'une File (par tableau simple) (1)

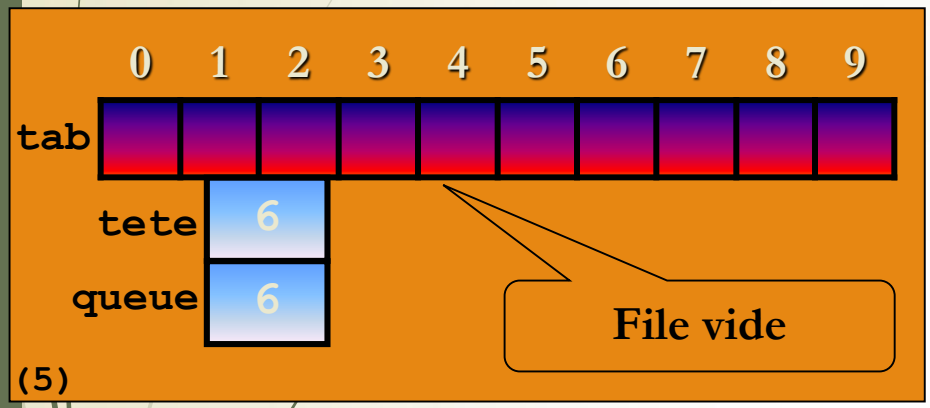


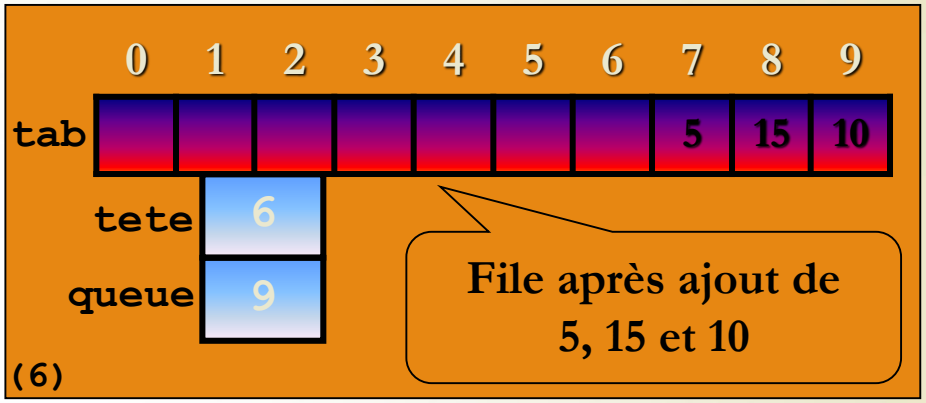


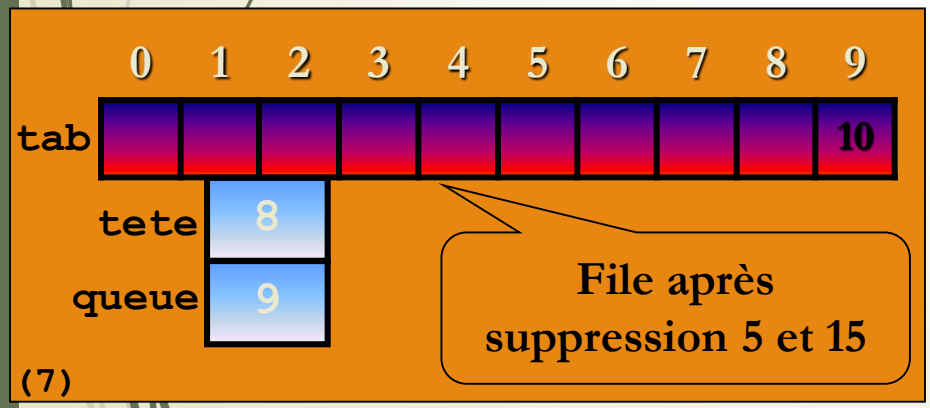


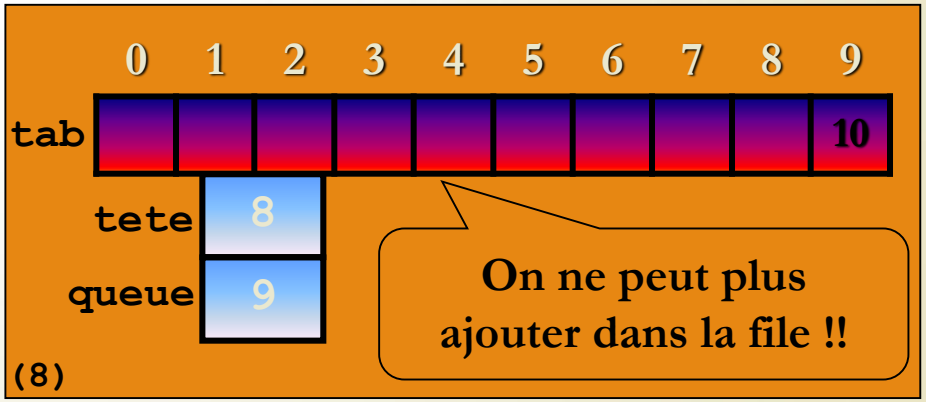


Représentation Contiguë d'une File (par tableau simple) (2)

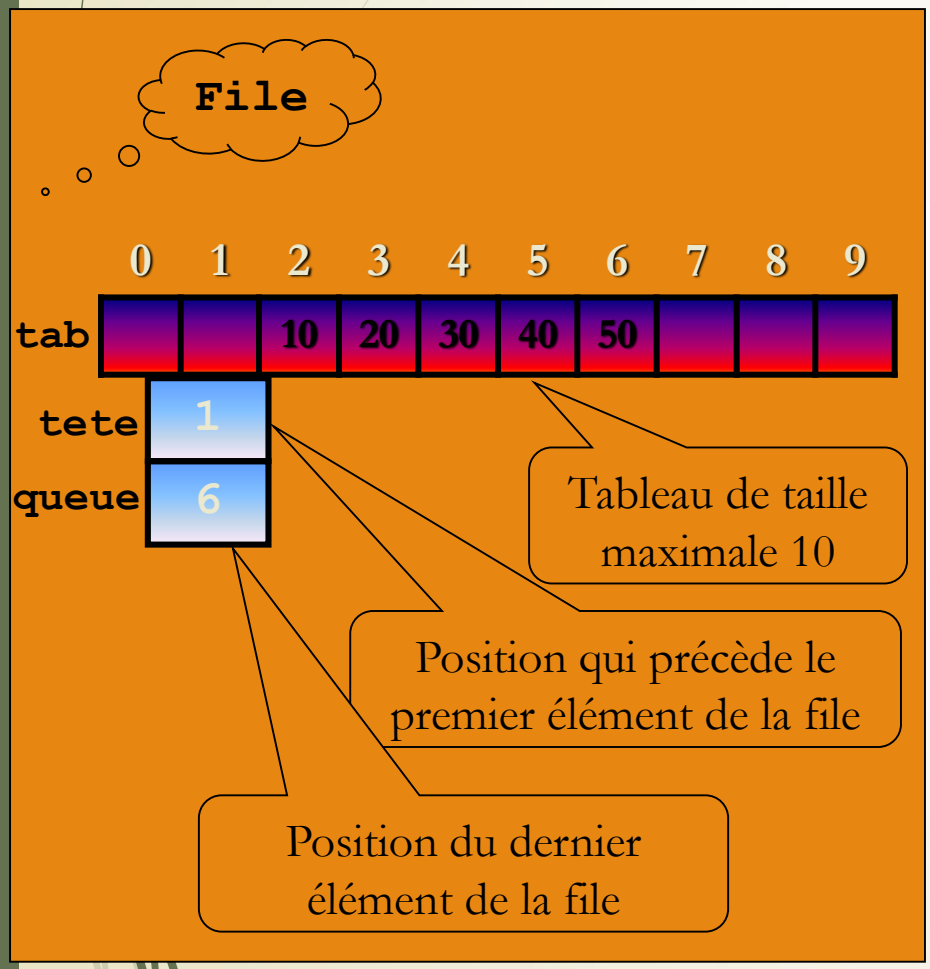








File Contiguë




```
  /* File contiguë en C */

// taille maximale file
#define MAX_FILE 10

// type des éléments
typedef int Element;

// type File
typedef struct {
 Element tab[MAX_FILE];
 int tete;
 int queue;
} File;
```

Spécification d'une File Contiguë


```
                    /* fichier "Tfile.h" */
#ifndef _FILE_TABLEAU
#define _FILE_TABLEAU

#include "Booleen.h"

// Définition du type File (implémentée par un tableau simple)
#define MAX_FILE 10  /* taille maximale d'une file */
typedef int Element;  /* les éléments sont des int */

typedef struct {
 Element tab[MAX_FILE]; /* les éléments de la file */
 int tete;  /* position précédant premier élé
 int queue; /* position dernier element */
} File;
// Déclaration des fonctions gérant la pile
File file_vide ();
File enfiler ( File f, Element e );
File defiler ( File f );
Element tete ( File f );
Booleen est_vide ( File f );

#endif
```

type File : une structure à trois champs

Réalisation d'une File Contiguë

```
/* fichier "Tfile.c"*/

#include "Tfile.h"
// Définition des fonctions gérant la file
// initialiser une nouvelle file
File file_vide() {
 File f;
 f.queue = f.tete = -1;
 return f;
}
// tester si la file est vide
Booleen est_vide(File f) {
 if (f.tete == f.queue) return vrai;
 return faux;
}
// valeur en tête de file
Element tete(File f) {
/* pré-condition : file non vide ! */
 if (est_vide(f)) {
  printf("Erreur: file vide !\n");
  exit(-1);
 }
 return (f.tab)[f.tete+1];
}
```

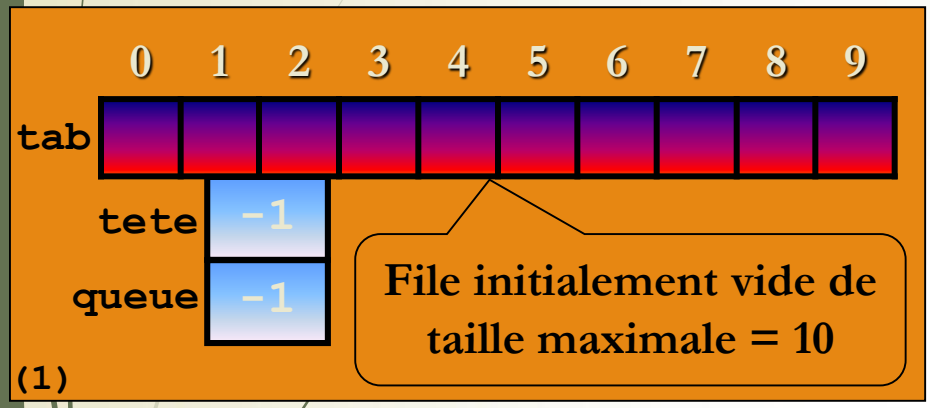
```
// ajout d'un élément
File enfiler(File f, Element e) {
 if (f.queue == MAX_FILE-1) {
  printf("Erreur: on ne peut ajouter !\n");
  exit(-1);
 }
 (f.queue)++;
 (f.tab)[f.queue] = e;
 return f;
}

// enlever un élément
File defiler(File f) {
/* pré-condition : file non vide !*/
 if (est_vide(f)) {
  printf("Erreur: file vide !\n");
  exit(-1);
 }
 f.tete++;
 return f;
}
```

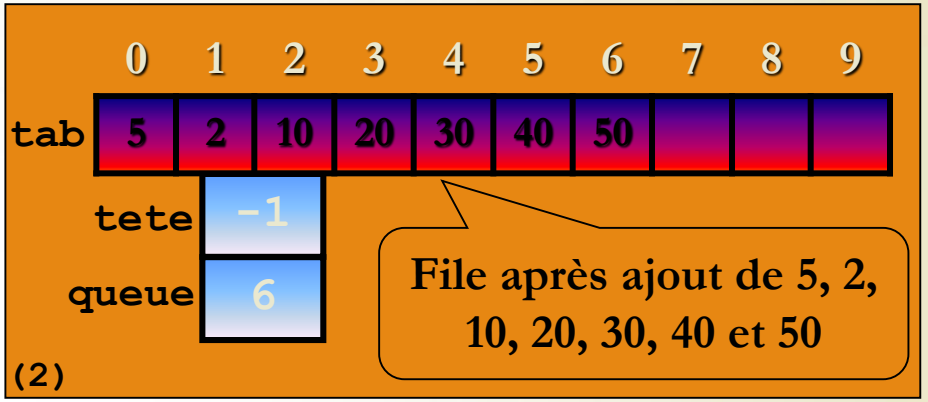
Représentation Contiguë d'une File (par tableau simple avec décalage)

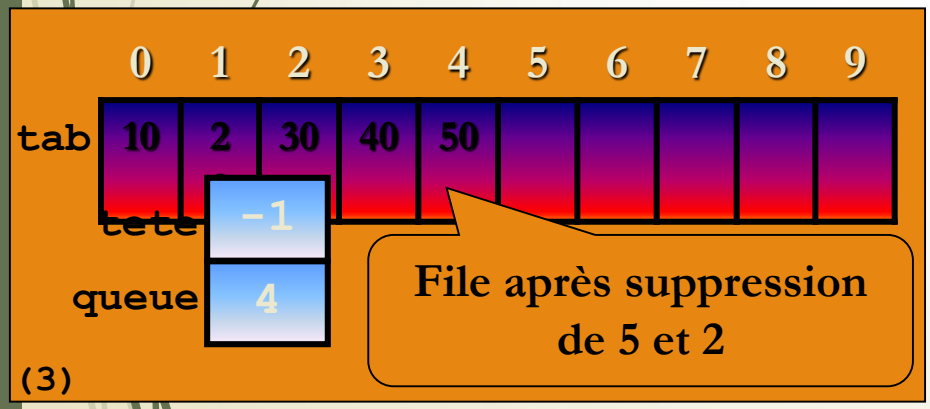
- ***Décaler les éléments de la file après chaque suppression***

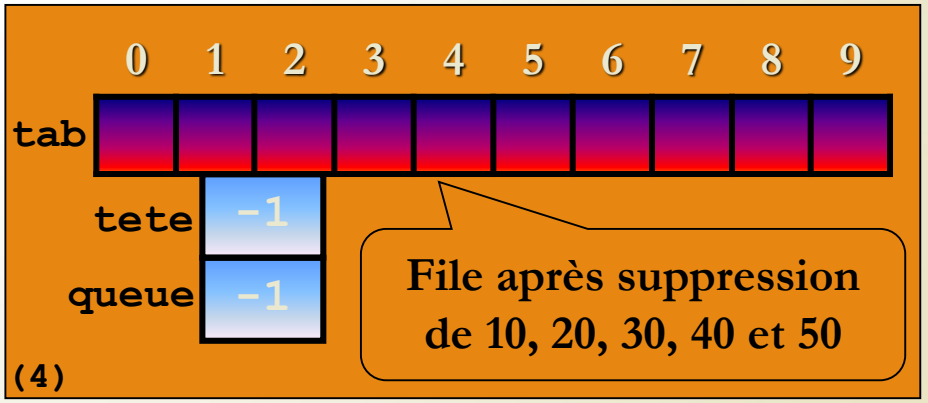
- ***Inconvénient :*** décalage très coûteux si la file contient plusieurs d'éléments

Représentation Contiguë d'une File (par tableau simple avec décalage)

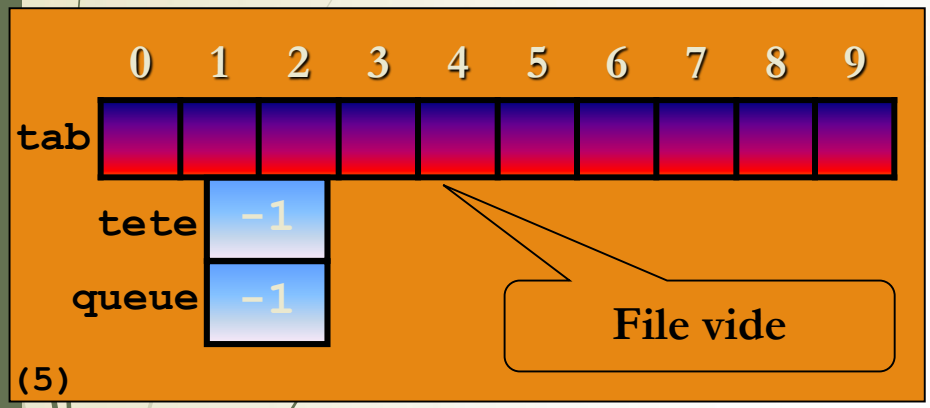


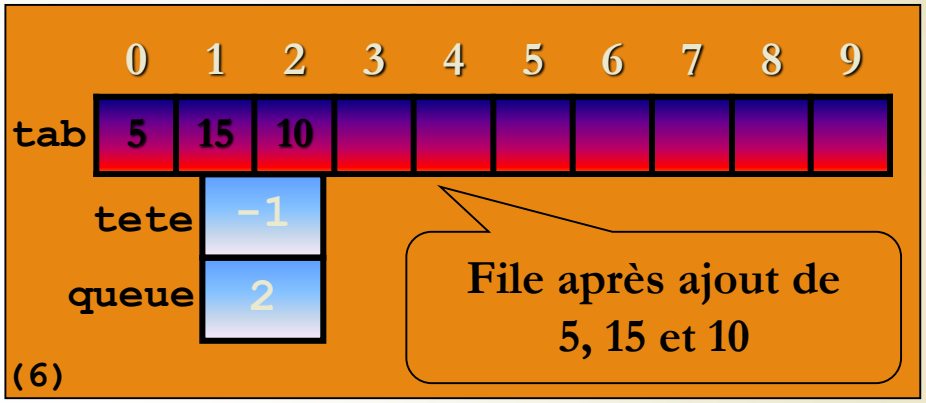


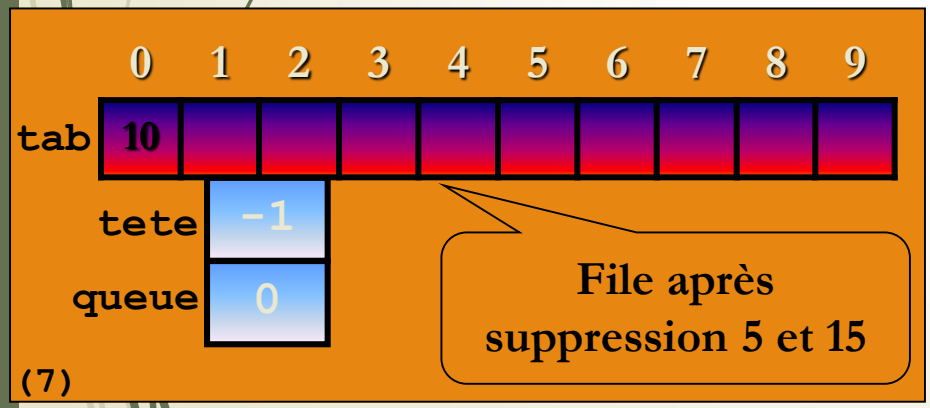


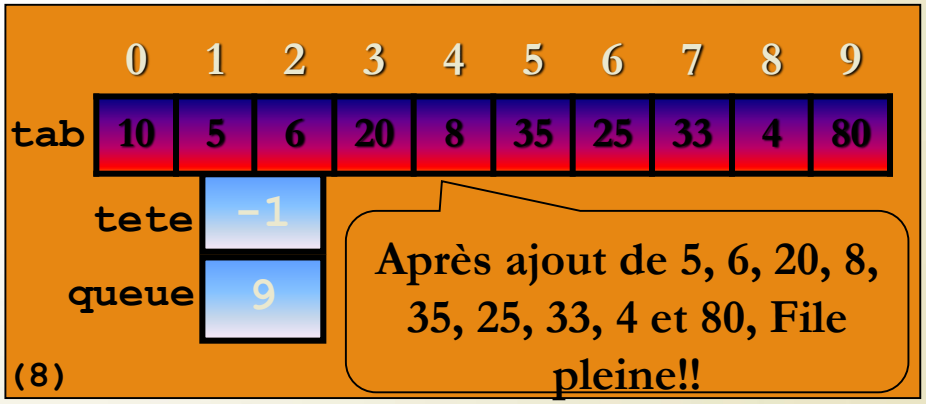


Représentation Contiguë d'une File (par tableau simple avec décalage)









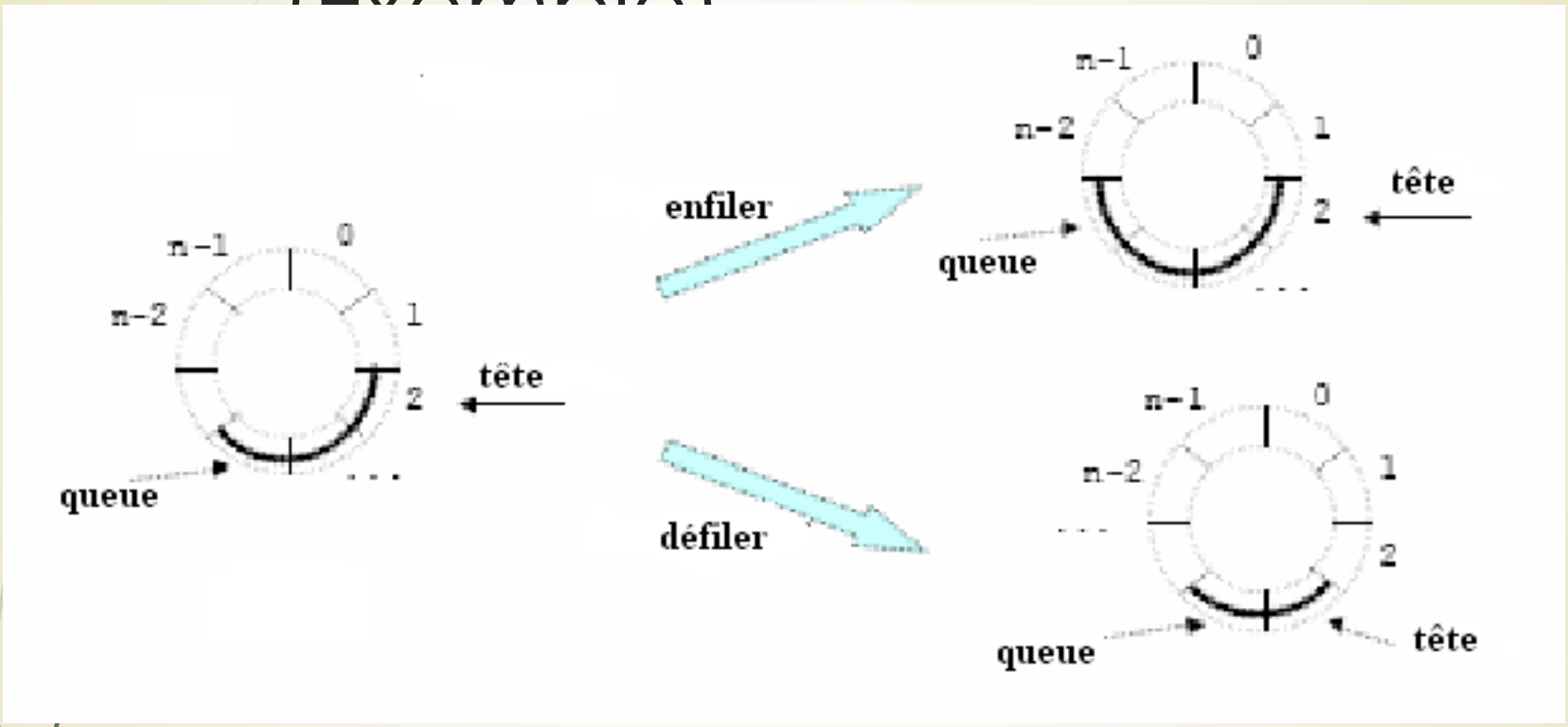
Représentation Contiguë d'une File (Par tableau circulaire)

- ***Gérer le tableau de manière circulaire :*** suivant de l'élément à la position `i` est l'élément à la position **`(i+1) modulo MAX_FILE`**

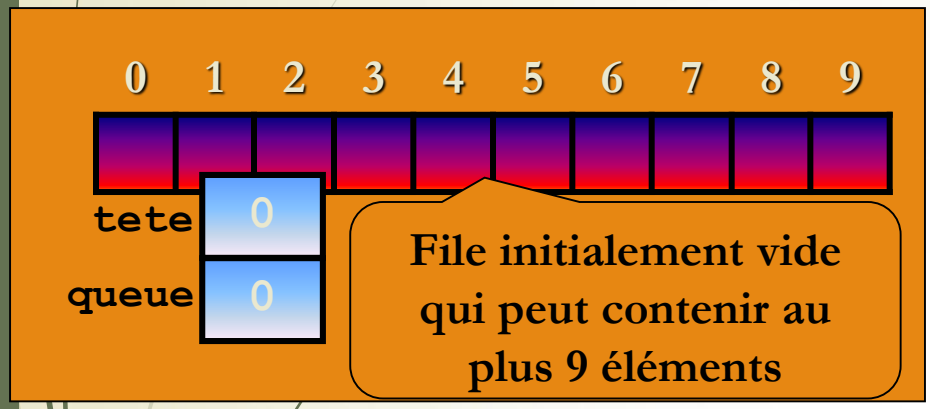
- ***Convention :*** file autorisée à contenir **`MAX_FILE-1`** éléments

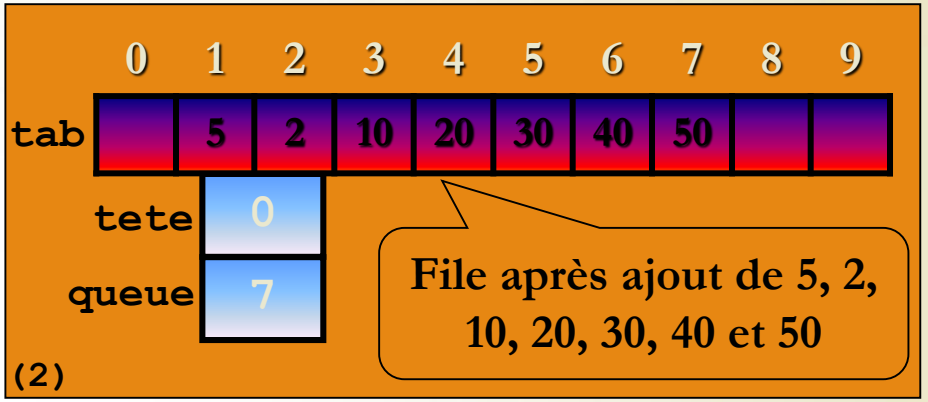
- ***Initialisation :*** **`tête ← queue ← 0`**

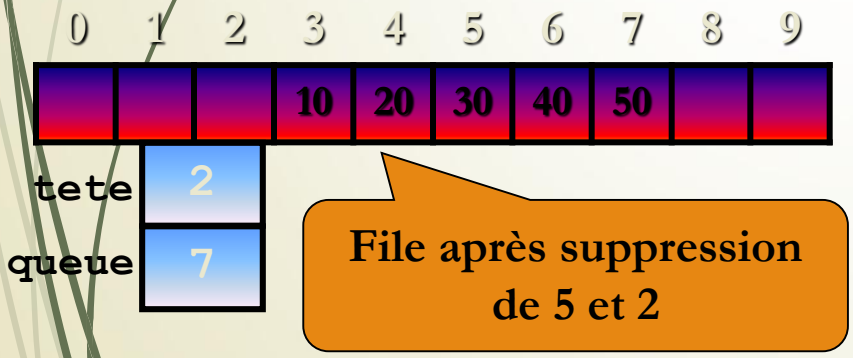
File Contiguë Circulaire (Exemple)

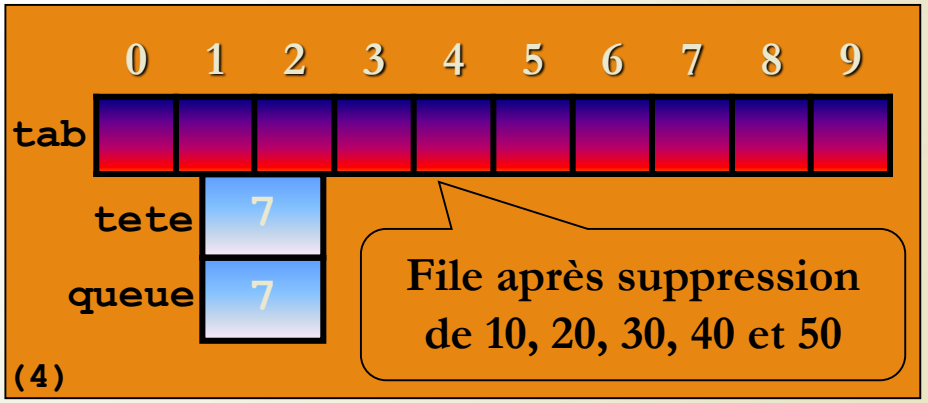


Représentation d'une File Contiguë circulaire(1)

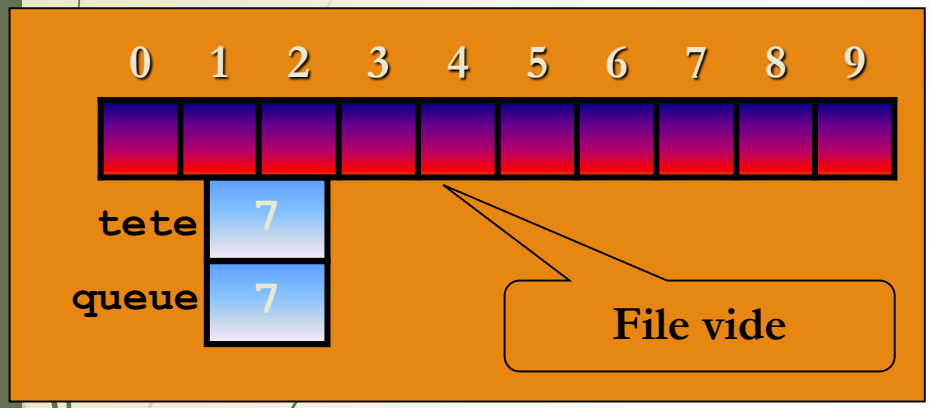


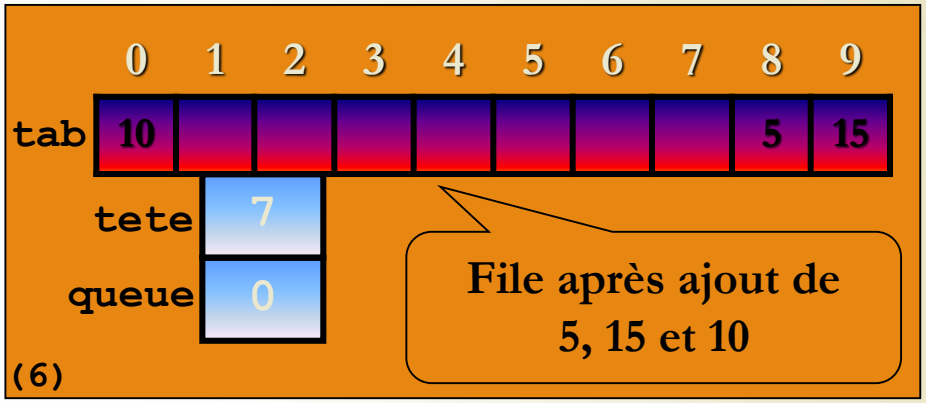


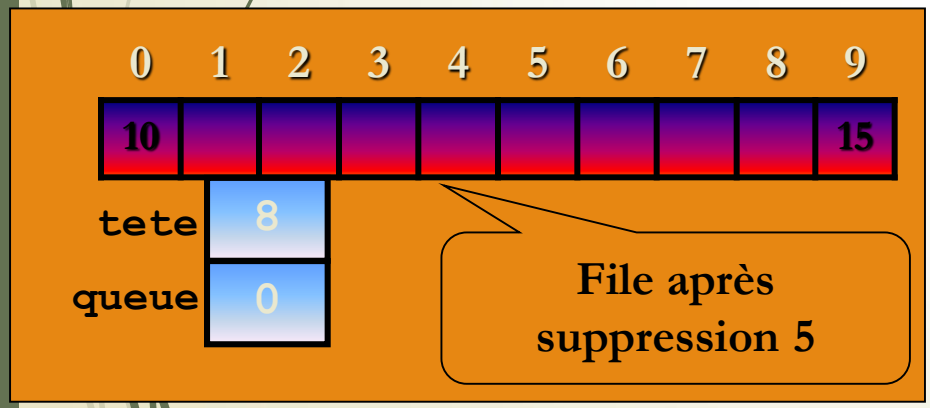


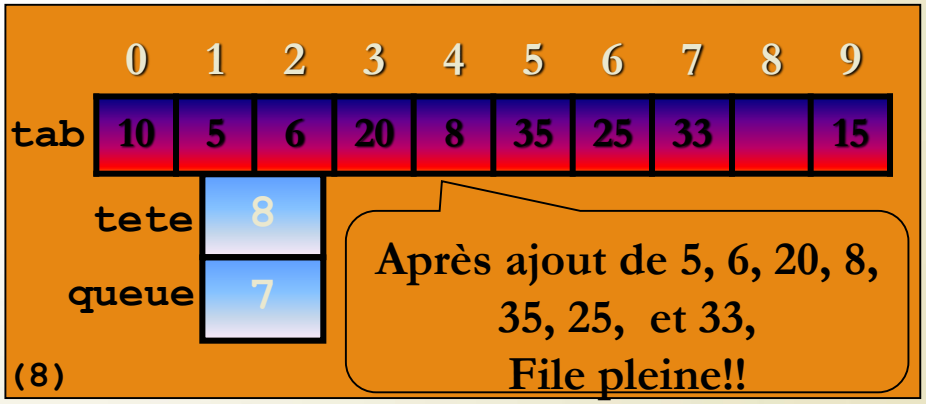


Réalisation d'une File Contiguë Circulaire (2)







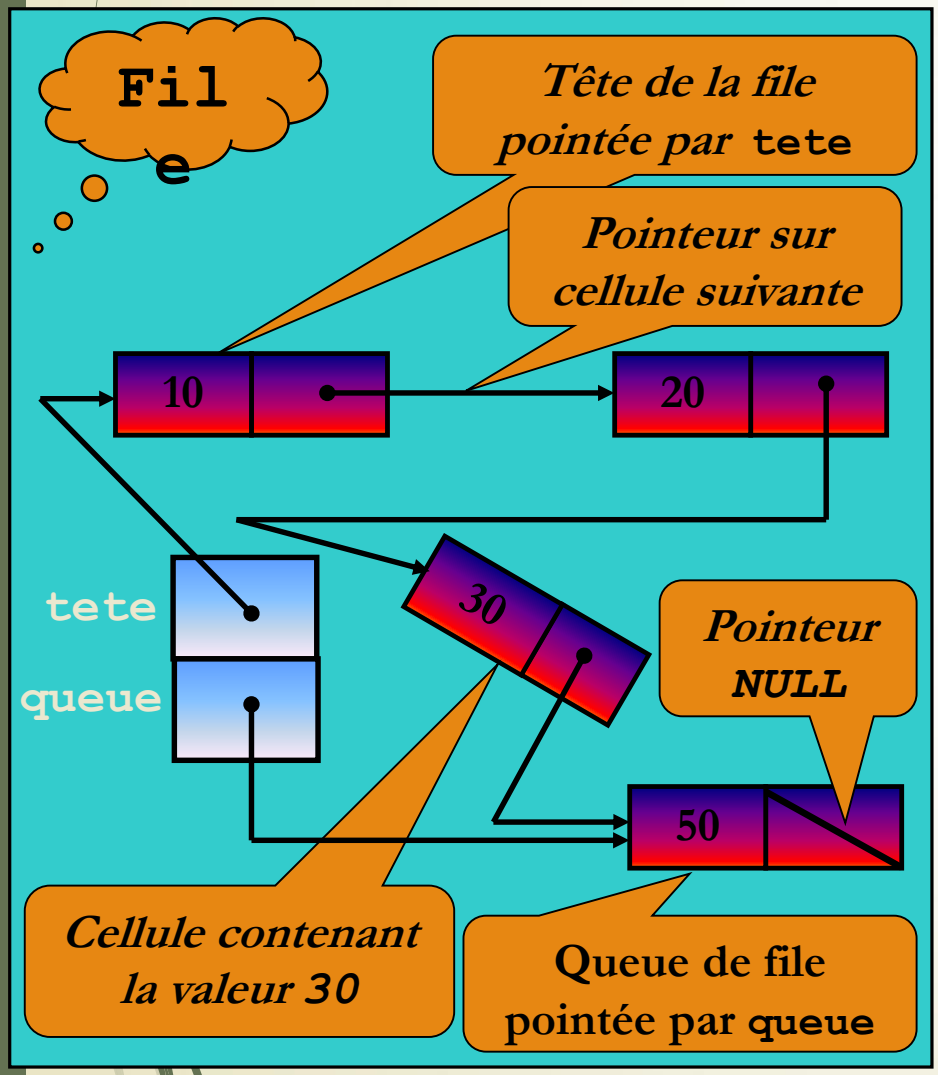


Réalisation d'une File Contiguë Circulaire


```
        /* fichier "TCfile.c" */
#include "Tfile.h"
// Définition des fonctions gérant la file

// initialiser une nouvelle file
File file_vide() {
 File f;
 f.queue = f.tete = 0;
 return f;
 }
// tester si la file est vide
Booleen est_vide(File f) {
 if (f.tete == f.queue) return vrai;
 return faux;
 }
// valeur en tête de file
Element tete(File f) {
/* pré-condition : file non vide ! */
 if (est_vide(f)) {
  printf("Erreur: file vide !\n");
  exit(-1);
  }
  return (f.tab)[(f.tete+1) % MAX_FILE];
  }
```

```
// ajout d'un élément
File enfiler(File f, Element e) {
 if (f.tete == (f.queue+1) % MAX_FILE) {
  printf("Erreur : file pleine !\n");
  exit(-1);
 }
 f.queue=(f.queue+1) % MAX_FILE;
 (f.tab)[f.queue] = e;
 return f;
}

// enlever un élément
File defiler(File f) {
/* pré-condition : file non vide !*/
 if (est_vide(f)) {
  printf("Erreur: file vide !\n");
  exit(-1);
 }
 f.tete=(f.tete+1) % MAX_FILE;
 return f;
}
```

File chaînée




```
   /* File chaînée en C */

// type des éléments
typedef int element;

// type Cellule
typedef struct cellule {
   element valeur;
   struct cellule *suivant;
 } Cellule;

// type File
typedef struct {
 Cellule *tete;
 Cellule *queue;
} File;
```

Complexité

- **Les opérations sur les files sont toutes en O(1)**
- **Ceci est valable pour les deux représentations : file contiguë circulaire et file chaînée**

Applications d'une File Exemples

- **Gestion des travaux d'impression d'une imprimante :**
  - Cas d'une imprimante en réseau, où les tâches d'impressions arrivent aléatoirement de n'importe quel ordinateur connecté. Les tâches sont placées dans une file d'attente, ce qui permet de les traiter selon leur ordre d'arrivée
- **Ordonnanceur (*dans les systèmes d'exploitation*) :**
  - Maintenir une file de processus en attente d'un temps machine ;
- **Parcours « en largeur » d'un arbre (*voir arbres*)**