Université Sultan Moulay Slimane
Ecole Supérieure de Technologie
- Beni Mellal -

STRUCTURES DE DONNEES

Filières : DUT **GI** / DUT **IDIA**
Semestre : S3
Année Universitaire : 2025/2026

Pr. M. OUTANOUTE
m.outanoute@usms.ma

- Chapitre 6 -

LES PILES ET FILES DYNAMIQUES

I. LES PILES DYNAMIQUES

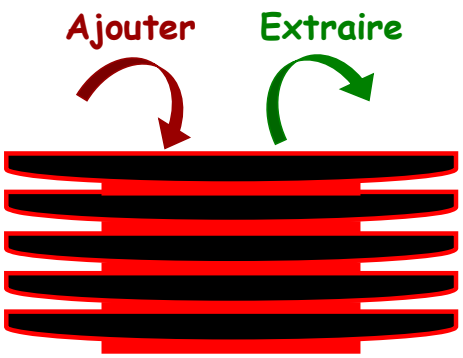
1) DÉFINITION

2) REPRÉSENTATION

3) OPÉRATIONS DE BASE

I-1) LES PILES : DÉFINITION

- Les Piles (stack) constituent des structures de données .
- Une Pile est une liste linéaire dont une seule extrémité (le sommet) est accessible -visible-.
- L'extraction ou l'ajout se font au sommet de la pile.
- Une Pile permet de réaliser ce qu'on nomme une LIFO (Last In First Out) : *Dernier Entré Premier Sorti*.
- **Exemple** : une pile d'assiettes
  Lorsqu'on ajoute une assiette en haut de la pile, on retire toujours en premier celle qui se trouve en haut sinon tout le reste s'écroule.



I-2) REPRÉSENTATION DES PILES

- La Pile en théorie est un objet dynamique (en opposition aux tableaux qui sont des objets statiques).

  Son état (et surtout sa taille) est variable.

- **Différentes représentations :**
  - Représentation statique
    - Un tableau, une variable globale indiquant le sommet;
    - Un enregistrement avec deux champs.
  - Représentation dynamique
    - En utilisant des listes chaînées.

I-2) REPRÉSENTATION DES PILES

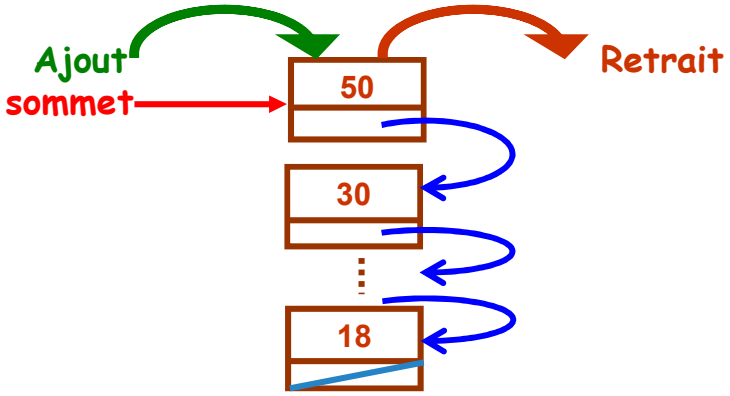
- La structure de la pile :
  - Nous allons créer une pile d'entiers (int).
  - Notre pile sera basée sur une liste chaînée (simple).
  - Chaque élément de la pile pointera vers l'élément précédent.
  - La Pile pointera toujours vers le sommet de la pile.

```
typedef struct element {
    int donnee; // La donnée que notre pile stockera
    struct element *precedent; // Pointeur vers l'élément précédent
} Element;
```

I-2) REPRÉSENTATION DES PILES

➢ La structure de la pile :

- Pour avoir une idée de ce à quoi ressemblera notre pile, voici un schéma qui pourra vous aider à mieux visualiser :



- Chaque case représente un élément de la pile ;
- Les cases sont emboîtées les unes sur les autres ;
- Le pointeur sommet devra toujours pointer vers le sommet de la pile.

I-3) OPÉRATIONS DE BASE SUR LES PILES

➢ Pour permettre les opérations sur la pile, nous allons sauvegarder certains éléments :

- Le premier élément : se trouve en haut de la pile et permet de réaliser l'opération de récupération des données (sommet) ;
- Le nombre d'éléments : taille .

➢ On suppose que la pile est déclarée de la façon suivante :

```
typedef struct element {
    char  *donnee;
    struct element  *precedent;
} Element;
// les variables globales
Element *sommet;
int taille;
```

I-3) OPÉRATIONS DE BASE SUR LES PILES

➢ Initialisation :

- **Prototype de la fonction :** void initialisation () ;
  - Cette opération doit être faite avant toute autre opération sur la Pile.
  - Elle initialise le pointeur sommet avec le pointeur NULL, et la taille avec la valeur 0.
- **Fonction :**

```
void initialisation () {
        sommet = NULL;
        taille = 0;
}
```


I-3) OPÉRATIONS DE BASE SUR LES PILES

➢ Ajout d'un nouvel élément :

L'ajout d'un nouvel élément se fera à la fin de la Pile .

- **Prototype de la fonction :** int Empiler(char *donnee);

  La fonction renvoie -1 en cas d'échec sinon elle renvoie 0.
- **Algorithme de la fonction :**
  - Déclarer le nouvel élément à insérer ;
  - Allouer de la mémoire pour le nouvel élément ;
  - Remplir le contenu du champ de données ;
  - Mettre à jour le pointeur sommet (le haut de la pile) ;
  - Mettre à jour la taille de la pile .

I-3) OPÉRATIONS DE BASE SUR LES PILES

➢ Ajout d'un nouvel élément :

- **La fonction :** // Empiler (ajouter) un élément dans la pile.

```
int Empiler (char *data) {
    Element *nouveau;
    nouveau = (Element *) malloc(sizeof(Element));
    if (nouveau == NULL) return -1;
    nouveau->donnee = (char *) malloc(50*sizeof(char));
    strcpy (nouveau->donnee, data);
    nouveau->precedent = sommet;
    sommet = nouveau;
    taille++;
    return 0;
}
```


I-3) OPÉRATIONS DE BASE SUR LES PILES

➢ Retrait d'un élément :

L'élément qui sera retiré de la pile sera le dernier élément que l'on a ajouté (l'élément se trouve au sommet de la pile) .

- **Prototype de la fonction :** int Depiler() ;

  La fonction renvoie -1 en cas d'échec sinon elle renvoie 0.

- **Algorithme de la fonction :**
  - Le pointeur supp_element contiendra l'adresse du pointeur sommet ;
  - Le pointeur sommet pointera vers l'élément précédent du pointeur sommet ;
  - La taille de la pile sera décrémentée d'un élément .

I-3) OPÉRATIONS DE BASE SUR LES PILES

➢ Retrait d'un élément :

- **La fonction :** // dépiler (supprimer) un élément de la pile.

```
int Depiler( ) {
    Element *supp_element;
    if (taille == 0)    return -1;  // Pile vide
    supp_element = sommet;
    sommet = sommet->precedent;
    free (supp_element->donnee);
    free (supp_element);
    taille--;
    return 0;
}
```


I-3) OPÉRATIONS DE BASE SUR LES PILES

➢ Affichage de la pile :

- Il faut se positionner au sommet de la pile ;
- En utilisant le pointeur precedent de chaque élément, la pile est parcourue du haut vers le bas ;
- La condition d'arrêt est donnée par la taille de la pile.

```
Afficher () {
    Element *courant ;  int i ;
    courant = sommet ;
    for( i=1 ; i <= taille ; i++ ) {  // ou while (courant != NULL)
        printf("%s\n", courant->donnee);
        courant = courant->precedent;
    }
}
```

I-3) OPÉRATIONS DE BASE SUR LES PILES

➢ Vidage de la pile :

Il s'agit d'une fonction permettant d'effacer la Pile .

- **Prototype de la fonction :** Vider( ) ;
- **Algorithme de la fonction :**

  Tant que le sommet n'est pas nul, effacer l'élément le plus haut de la pile .

- **La fonction :**

```
Vider() {
    while (sommet != NULL)  Depiler(); // ou while (taille !=0)
}
```


II. LES FILES DYNAMIQUES

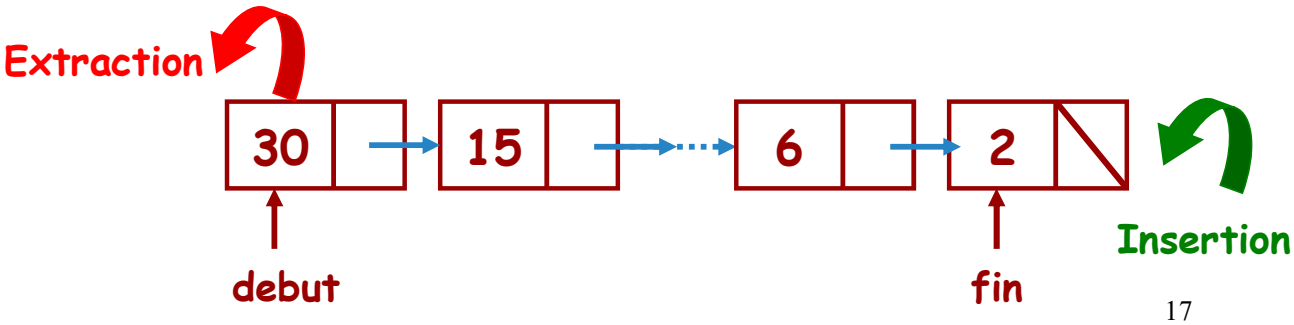
1) DÉFINITION

2) REPRÉSENTATION

3) OPÉRATIONS DE BASE

II-1) LES FILES : DEFINITION

- Identiquement aux piles, cette structure est basée sur les listes chaînées .
- La file est une structure de données, qui permet de stocker les données dans l'ordre FIFO (First In First Out) : *Premier Entré Premier Sorti* .
- L'ajout d'un élément se fera à la fin (queue, arrière, fin) de la liste (File) et le retrait d'un élément se fera au début (avant, tête, debut) de la liste (File) .



II-2) REPRESENTATION DES FILES

- La récupération (extraction) des données sera faite dans l'ordre d'insertion .
- Cependant, on ne pointera plus sur le sommet mais sur la base de la file (sur le premier élément de la file) .
- Nous utilisons un pointeur vers l'élément suivant et non plus vers l'élément précédent (ce qui est expliqué par le fait que nous pointons à la base de la file) .

```
typedef struct element {
    int info;    // La donnée que notre file stockera
    struct element *suivant; // Pointeur vers l'élément suivant
} Element;
```

II-2) REPRESENTATION DES FILES

- Pour avoir le contrôle de la file, il est préférable de sauvegarder certains éléments :
  - Le premier élément : se trouve au début de la file et il permet de réaliser l'opération d'extraction des données : Element *debut ;
  - Le dernier élément : se trouve à la fin de la file et il permet de réaliser l'opération d'insertion des données : Element *fin ;
  - Le nombre d'éléments : int taille ;
- On suppose que la File est déclarée de la façon suivante :

```
typedef struct element {
    char *donnee;
    struct element *suivant;
} Element;
// les variables globales
Element *debut, *fin;
int taille;
```


II-3) LES OPÉRATIONS DE BASE SUR LA FILE

- **Initialisation de la file :**
  - **Prototype de la fonction :** void initialisation () ;
  - Cette opération doit être faite avant toute autre opération sur la File.
  - Elle initialise le pointeur *debut* et *fin* avec le pointeur NULL, et la *taille* avec la valeur 0 .
  - **La fonction :**

```
void initialisation () {
        debut = NULL;
        fin = NULL;
        taille = 0;
}
```

II-3) LES OPÉRATIONS DE BASE SUR LA FILE

➢ Insertion d'un nouvel élément :

- **Prototype de la fonction :**

  La fonction renvoie -1 en cas d'échec sinon elle renvoie 0

  int Enfiler(char *donnee) ;

- **Algorithme de la fonction :**
  - Déclarer le nouvel élément à insérer ;
  - Allouer de la mémoire pour le nouvel élément ;
  - Remplir le contenu du champ de données ;
  - Mettre à jour le pointeur debut vers le 1er élément (le début de la file) ;
  - Mettre à jour le pointeur fin (ça nous servira pour l'insertion vers la fin de la file) ;
  - Mettre à jour la taille de la file.

II-3) LES OPÉRATIONS DE BASE SUR LA FILE

➢ Insertion d'un nouvel élément :

```
int Enfiler(char *data) {
    Element *nouveau;
    nouveau=(Element *) malloc (sizeof(Element));
    if (nouveau==NULL) return -1;
    nouveau->donnee = (char *) malloc (50 * sizeof(char));
    strcpy (nouveau->donnee, data);
    nouveau->suivant = NULL;
    if (taille == 0) {  fin = nouveau; debut = nouveau;}
    else { fin->suivant = nouveau; fin = nouveau; }
    taille++;
    return 0;
}
```

II-3) LES OPÉRATIONS DE BASE SUR LA FILE

➢ Suppression d'un élément :

- Pour supprimer l'élément de la file, il faut tout simplement supprimer l'élément vers lequel pointe le pointeur debut.
- Cette opération ne permet pas de récupérer la donnée au début de la file, mais seulement de la supprimer.
- **Algorithme de la fonction :**
  - Le pointeur supp_elem contiendra l'adresse du 1er élément (debut) ;
  - Le pointeur debut pointera vers le 2ème élément (après la suppression du 1er élément, le 2ème sera au début de la file);
  - La taille de la file sera décrémentée d'un élément

II-3) LES OPÉRATIONS DE BASE SUR LA FILE

➢ Suppression d'un élément :

La fonction renvoie -1 en cas d'échec sinon elle renvoie 0.

```
int Defiler () {
    Element *supp_element;
    if (taille == 0)    return -1;
    supp_element = debut;
    debut = debut->suivant;
    if  (taille == 1) fin=NULL;
    free (supp_element->donnee);
    free (supp_element);
    taille--;
    return 0;
}
```

II-3) LES OPÉRATIONS DE BASE SUR LA FILE

➢ Affichage de la file :

- **Algorithme :**
  - il faut se positionner au debut de la file ;
  - En utilisant le pointeur suivant de chaque élément, la file est parcourue du debut vers la fin ;
  - La condition d'arrêt est donnée par la taille de la file.
- **Fonction :**

```
void afficher () {
    Element *courant;  int i;
    courant = debut;
    for(i=0 ; i < taille ; i++) {      // while (courant!=NULL)
       printf("%s\n", courant->donnee);
       courant = courant->suivant;
    }
}
```


II-3) LES OPÉRATIONS DE BASE SUR LA FILE

➢ Vidage de la file :

Il s'agit d'une fonction permettant d'effacer la file.

- **Algorithme :**

  Tant que le pointeur fin n'est pas NULL

  Effacer le premier élément de la file

- **Fonction :**

```
Vider_file() {

    while (fin != NULL)  Defiler();  // while (taille!=0) Defiler ();

}
```