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 2-

Les piles et les files

- PILE : DÉFINITION ET REPRÉSENTATION
- LES OPÉRATIONS DE BASE SUR LA PILE
- EXEMPLE D'APPLICATION
- FILE : DÉFINITION ET REPRÉSENTATION
- LES OPÉRATIONS SUR LA FILE
- EXEMPLE D'APPLICATION

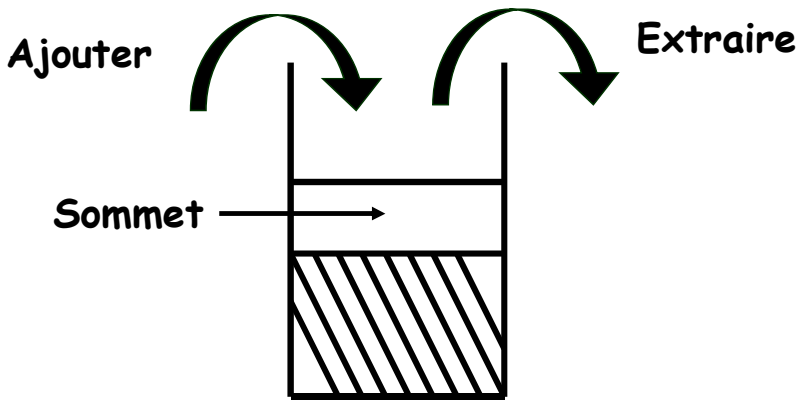
- Chapitre 2-

Les piles et les files

LES PILES

1. PILE: DÉFINITION ET REPRÉSENTATION

- 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.
- Les PILEs suivent une démarche LIFO (Last In First Out) ce qui signifie en clair que les derniers éléments à être ajoutés à la pile seront les premiers à être récupérés.



1. PILE: DÉFINITION ET REPRÉSENTATION

- 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.

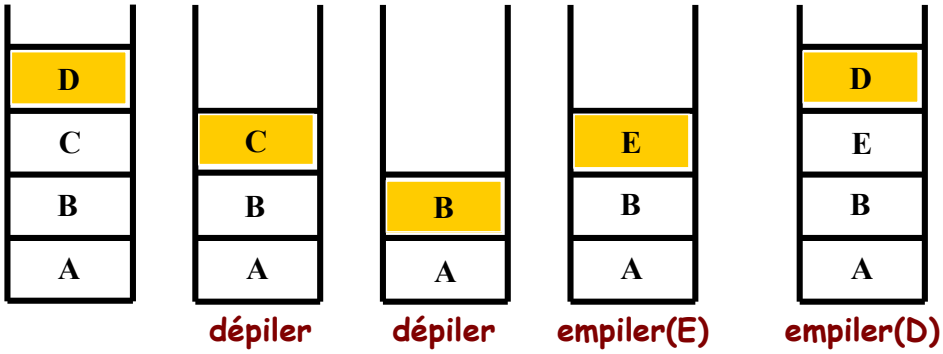
- **Différentes représentations**
  - **Représentation statique**
    - Un tableau et une variable globale indiquant le sommet
    - Un enregistrement avec deux champs
  - **Représentation dynamique**
    - Les listes chaînées ?

1. PILE: DÉFINITION ET REPRÉSENTATION

- **Exemple** : une pile d'assiettes

  Lorsqu'on ajoute une assiette en haut de la pile, on retire toujours celle qui se trouve en haut de la pile : c'est à dire celle qui a été ajoutée en dernier, sinon tout le reste s'écroule.

- Les piles peuvent être utilisées dans des algorithmes d'évaluation des expressions mathématiques.
- Chaque problème qui utilise cette démarche peut être simulé (dans sa résolution) par les piles.



2. LES OPÉRATIONS DE BASE SUR LA PILE

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

```
# define Max 20
typedef char element;
typedef element Pile[Max];
/* les variables globales*/
Pile p;
int Sommet;
```


2. LES OPÉRATIONS DE BASE SUR LA PILE

- fonction **Empiler** : insérer l'élément au sommet de la pile.

```
void Empiler(element x) {
   if  (Sommet >= Max-1)
      printf(" Stack overflow : dépassement de la capacité");
   else {
      Sommet = Sommet+1;
      p[Sommet] = x;
   }
}
```

2. LES OPÉRATIONS DE BASE SUR LA PILE

- **fonction Depiler** : extraire l'élément qui est au sommet de la pile et le retourner comme valeur de la fonction.

```
element Depiler(void) {
   element x;
   if (Sommet < 0)  printf(" Stack Underflow ");
   else {
        x = p[Sommet];
        Sommet = Sommet - 1;
        return x;
    }
}
```


2. LES OPÉRATIONS DE BASE SUR LA PILE

- **fonction initialisation**: initialise la pile.

```
void initialisation(void) {
   Sommet = -1;
}
```

- **fonction Pile_vide :** teste si la pile est vide ou non.

```
int Pile_vide(void) {
   return (Sommet < 0);
}
```

- **fonction Pile_pleine :** teste si la pile est pleine ou non.

```
int Pile_pleine(void) {
   return (Sommet >= Max-1);
}
```

3. EXEMPLE D'APPLICATION

➢ Traitement des expressions mathématiques

- ❑ Si on a un seul type de délimiteur, on utilise un compteur :

  +1 si on ouvre le délimiteur

  -1 si on ferme le délimiteur

* A + B)	0 0 0 0 -1
*) A + B (- C	0 -1 -1 -1 -1 0 0 0
(A + B) - C	1 1 1 1 0 0 0

- ❑ Une expression est correcte si :
  - le compteur est toujours >= 0
  - le compteur = 0 à la fin

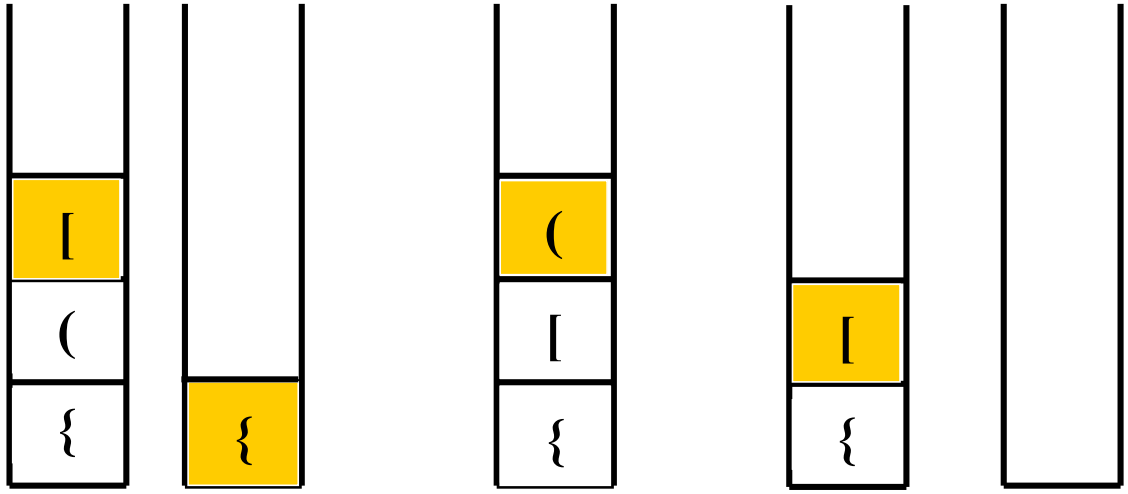
3. EXEMPLE D'APPLICATION

➢ Traitement des expressions mathématiques

- Si on a plus d'un délimiteur par exemple : ([{
- On peut utiliser la méthode précédente, mais il faut garder la trace du nombre d'ouvrant et fermant pour chaque type de délimiteur.
- La résolution de ce problème peut être simulée par la structure de la pile :
  - Délimiteur ouvrant : Empiler
  - Délimiteur fermant : Dépiler
  - Le dernier délimiteur ouvert est le premier à être fermé (LIFO).

3. EXEMPLE D'APPLICATION

Exemple: { x + (y - [a + b]) * c - [(d + e)] }



{x+(y-[{x+(y-[a+b]) …+b])*c-[(d … [(d+e) … [(d+e)]}

expression correcte

3. EXEMPLE D'APPLICATION

- **Problème d'underflow :**
  Pile_vide et on essaie de dépiler : correspond à une fermeture de plus.
- **Problème d'overflow :**
  Dépassement de la capacité en nombre de délimiteurs ouvrants et fermants.
- Il faut toujours s'assurer que le délimiteur qu'on ferme correspond bien (même type) à celui qu'on vient d'ouvrir.

3. EXEMPLE D'APPLICATION

Algorithme :

```
lire une expression
TANT QUE un symbole est valide (expression valide) FAIRE
   SI (symbole) est un délimiteur ouvrant ALORS Empiler(symbole)
   SINON
      SI (symbole) est un délimiteur fermant ALORS
         S ⟵ Dépiler
         SI S ne correspond pas au délimiteur fermant ALORS
              l'expression est incorrecte (invalide)
         FINSI
      FINSI
   FINSI
   passer au symbole suivant
FIN TANT QUE
SI la pile est vide ALORS l'expression est correcte
SINON l'expression est incorrecte
FINSI
```


3. EXEMPLE D'APPLICATION

```
#include<stdio.h>
#include<string.h>
#define Max 20

// Les structures utilisées
typedef char element;
typedef element Pile[Max];

// les variables globales
Pile p;
int Sommet;
```

3. EXEMPLE D'APPLICATION

- Fonction associant à chaque symbole fermant le symbole ouvrant correspondant en le retournant

  Exemple : Ouvrant('] ') doit retourner le symbole ' ['

```
element Ouvrant(element symb) {
   switch (symb) {
         case ' ] ' : return ' [ ' ;
         case ' ) ' : return ' ( ' ;
         case ' } ' : return ' { ' ;
   }
}
initialisation() { sommet = -1 ;}
int Pile_vide() { return (sommet < 0) ;}
int Pile_pleine() { return (sommet >= Max-1) ;}
```


3. EXEMPLE D'APPLICATION

```
main() {
   element expression[4*Max];
   element symbole, S;
   int lon, i, Valide;

   printf("Entrer une expression mathématique");
   gets(expression);
   initialisation();
   long=strlen(expression);
   i=0;
   Valide=1;
```

3. EXEMPLE D'APPLICATION

```
    while ((i<=long-1) && (Valide==1)) {
        symbole=expression[i];
        if ( (symbole=='[' ) || (symbole=='(' ) || (symbole=='{' ) )
                Empiler(symbole);
        if ( (symbole==']' ) || (symbole==')' ) || (symbole=='}' ) ) {
                S=Depiler();
                if (S != Ouvrant(symbole)) Valide=0;
        }
        i++;
    }
    if (Pile_vide() && (Valide==1)) printf("Expression correcte");
    else printf("Expression incorrecte");
}
```

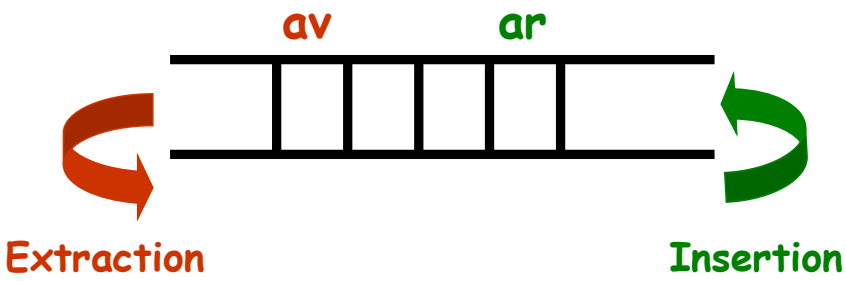

- Chapitre 2-

Les piles et les files

LES FILES

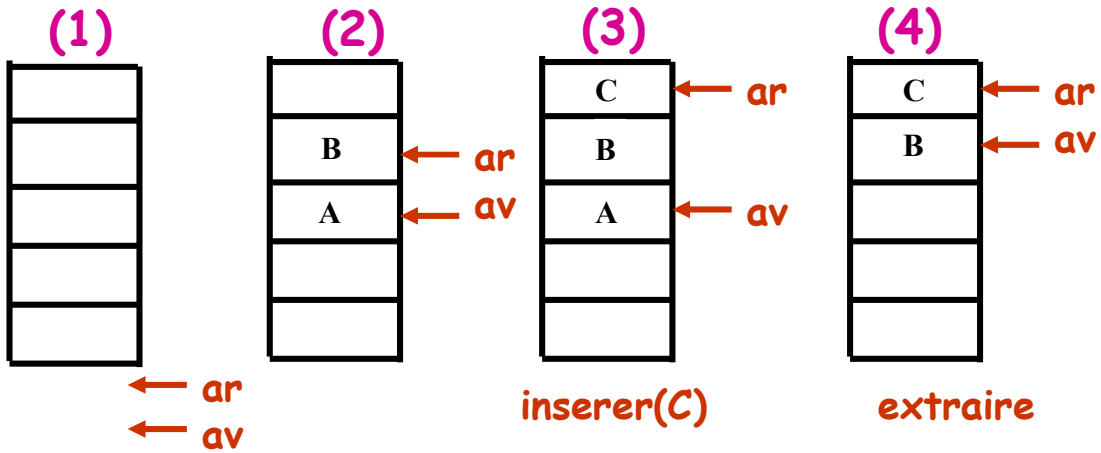
4. FILE : DÉFINITION ET REPRÉSENTATION

- Une FILE est une liste linéaire où toutes
  - les insertions se font par une extrémité (queue, arrière).
  - les extractions se font par l'autre extrémité (avant, tête).
- Exemple : file d'attente
  - dans le bus
  - des requêtes destinées à un serveur



4. FILE : DÉFINITION ET REPRÉSENTATION

- But : Une loi FIFO (First In First Out), le premier arrivé est le premier servi.
- Représentation :
  - un vecteur qui représente la FILE F[Max]
  - av et ar : deux variables globales indiquant la tête et la queue de la file.



5. LES OPÉRATIONS SUR LES FILES

- **Initialisation** : la file est vide au départ

  ar ← -1 ; av ← -1 ;

- **Insertion** (**enfilement**) de l'élément x :

  ar ← ar+1 ; F[ar] ← x ;

- **Extraction** (**défilement**) de l'élément dans une variable x:

  x ← F[av] ; av ← av+1 ;

- **Problème :** du fait que les deux variables av et ar sont toujours incrémentées, on arrive à un Underflow même si la file n'est pas saturée (voir (4), on ne peut pas insérer un autre élément).

5. LES OPÉRATIONS SUR LES FILES

- **Première solution :**

  Modifier extraire de sorte lorsqu'un élément est extrait toute la file est passée vers l'avant.

  - Si on ignore la possibilité d'un Underflow l'opération s'écrit :

```
x = F[0] ;
for (i=0; i <= ar-1; i++)  F[i] = F[i+1] ;
ar = ar-1 ;
```

  - Remarque : on n'a pas besoin dans ce cas de préciser la tête, elle est toujours au début de la file.
  - Inconvénient : lorsqu'on a un tableau de grande taille, extraire un élément nécessite le déplacement de tous les éléments du tableau.

5. LES OPÉRATIONS SUR LES FILES

➢ **Première solution :**

- **initialisation :** ar = -1
- **file est vide :** ar < 0
- **file est pleine :** ar >= n-1

```
element extraire() {
    Element x; int i;
    if file_vide() printf("Underflow");
    else  {
          x = F[0];
          for(i=0; i<=ar-1; i++) F[i] = F[i+1];
          ar = ar-1;
          return x;
    }
}
```


5. LES OPÉRATIONS SUR LES FILES

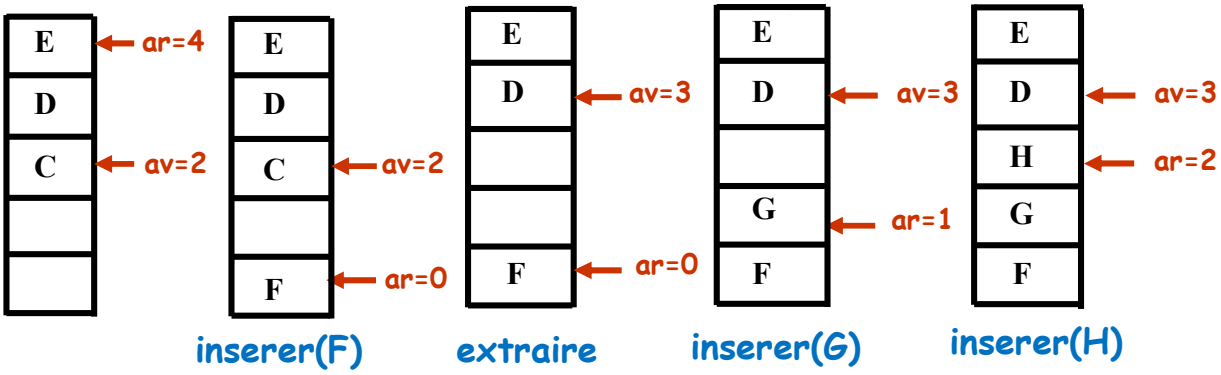
➢ **Première solution :**

```
void inserer(element x) {
    if (file_pleine()) printf("Overflow");
    else {
        ar = ar+1;
        F[ar] = x;
    }
}
```

5. LES OPÉRATIONS SUR LES FILES

➢ Deuxième solution :

Consiste à voir la FILE comme un ensemble circulaire d'éléments F[0], F[1], F[2], …. F[n-1], F[0]



- Il n 'y a pas perte de place, on exploite tout le tableau.

  On aura un Overflow lorsque (ar+1) % n == av ;

5. LES OPÉRATIONS SUR LES FILES

```
initialisation() { av = -1; ar = -1; }
file_vide() { return (ar <= av) ; }
file_pleine() { return ( (ar+1) % n == av) ; }
element extraire {
    element x ;
    if ( file_vide() ) printf("Underflow");
    else {
        x = F[av];
        if ( av == n-1)  av = 0;
        else av = av+1;
        return x ;
    }
}
```

5. LES OPÉRATIONS SUR LES FILES

```
void inserer(element x){
    if (ar == n-1) ar = 0;
    else ar = ar+1;
    if (av == ar) {
        printf("Overflow");
        if (ar == 0) ar = n-1;
        else ar = ar-1 ;
    }
    else F[ar] = x ;
}
```


5. LES OPÉRATIONS SUR LES FILES

```
void inserer(element x){
    if (ar == n-1) ar = 0;
    else ar = ar+1;
    if (av == ar) {
        printf("Overflow");
        if (ar == 0) ar = n-1;
        else ar = ar-1 ;
    }
    else F[ar] = x ;
}
```

6. EXEMPLE D'APPLICATION

On considère une file d'entiers déclarée globalement de la manière suivante: un tableau des entiers (F) et deux variables représentant l'avant (av) et l'arrière (ar) de la file. (Utiliser le principe d'une file circulaire).

On veut récupérer le maximum de la file en respectant son principe de fonctionnement (FIFO), les fonctions à rédiger sont:

- initialisation()
- int file_vide()
- int file_pleine()
- void enfiler(int)
- int defiler()
- main()

6. EXEMPLE D'APPLICATION

```
#include<stdio.h>
#define max 100
typedef int File[max];

File F;
int av, ar;

void initialisation()
{
 av=-1; ar=-1;
}
```

6. EXEMPLE D'APPLICATION

```
int file_vide()
{
  return(ar==-1 && av==-1);
}

int file_pleine()
{
  return ((ar+1)%max==av);
}
```


6. EXEMPLE D'APPLICATION

```
int defiler() {
    int x;
    if (file_vide()) printf("Underflow");
    else  {
        x=F[av];
        if(av==ar) { av=-1; ar=-1; }
        else if(av==max-1) av=0;
        else av=av+1;
        return x;
    }
}
```

6. EXEMPLE D'APPLICATION

```
void enfiler(int x) {
   if(file_pleine()) printf("Overflow \n");
   else{
      if (ar==max-1) ar=0;
      else  if (av==-1 && ar==-1) {
         ar=0; av=0;
      }
      else ar=ar+1;
      F[ar]=x;
   }
}
```


6. EXEMPLE D'APPLICATION

```
main() {
   int nb,mx,i,j,x,A[max];
   initialisation();
   printf("Entrer nb:\n");
   scanf("%d",&nb);
   for(i=1;i<=nb;i++) {
      printf("Taper un entier n° : \n",i);
      scanf("%d",&x);
      enfiler(x);
   }
   printf("------------------------------\n");
```

6. EXEMPLE D'APPLICATION

```
if(!file_vide()) {
    x=defiler();
    mx=x;
    A[0]=x;
    i=1;
    while(!file_vide()) {
        x=defiler();
        if(x>mx) mx=x;
        A[i]=x;
        i++;
    }
```


6. EXEMPLE D'APPLICATION

```
    printf("==================================\n");
    for(j=0;j<i;j++) enfiler(A[j]);
    printf("Maximum est: %d \n",mx);
  }
}
```

7. EXERCICE

On considère une file d'entiers déclarée globalement de la manière suivante: un tableau des entiers (F) et deux variables représentant l'avant (av) et l'arrière (ar) de la file. (Utiliser le principe d'une file par décalage).

On veut supprimer les entiers pairs de la file en respectant son principe de fonctionnement (FIFO). Les fonctions à rédiger sont :

- initialisation()
- int file_vide()
- int file_pleine()
- void enfiler(int)
- int defiler()
- main()