Chapitre 3
Les piles et les files

- **Partie I: Les piles**
- **Partie II: Les files**

Partie I:
Les piles

• Les piles

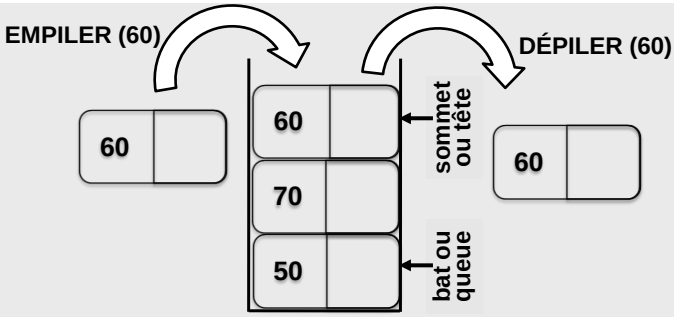
- **Définition et principe**
- Intérêt et applications
- Manipulation des piles

Définition d'une pile

- Une pile est un ensemble d'éléments de même type, seul l'élément au sommet est visible.
- Une pile est une structure de données dynamique caractérisée par un comportement particulier en ce qui concerne l'insertion et l'extraction des éléments (des éléments y sont introduits, puis extraits) ayant la propriété que, lors d'une extraction, l'élément extrait est celui qui a y été introduit le plus récemment.
- Les piles sont très utilisées : pile d'assiettes, pile de dossiers à traiter, …

Principe d'une pile

- Voyez d'abord la figure suivante:



Pile d'assiettes

Une pile

- **EMPILER** (PUSH): insérer un élément à la **tête** de la pile.
- **DÉPILER** (POP): supprimer un élément de la **tête** de la pile.

- Une pile est une structure dont la loi d'évolution respecte la règle **LIFO** (**L**ast **I**n, **F**irst **O**ut) (Dernier Entrée, Premier Sortie).

Intérêt et applications

- **Intérêt :**
  - Type Abstrait de Données (**TAD**), linéaires et asymétrique.
  - Permet de manipuler des ensembles dynamiques.

- **Applications :**
  - Mémoriser les **pages web** visitées dans un **navigateur web**: l'adresse de chaque nouvelle page visitée est mémorisée (empilée) dans une **pile**,
  - La fonction «**Annuler la frappe**» utilise une **pile** dans un système de traitement de texte: les modifications apportées au texte sont mémorisées (empilées) dans une **pile**,
  - Évaluation d'expressions arithmétiques, logiques, …
  - Les **algorithmes** récursifs admis par certains **langages** (LISP, Algol, Pascal, etc.) utilisent une **pile** d'appel.

Structure d'un élément et structure de contrôle de la pile

- Pour définir un élément de la pile, on procède comme pour une liste simplement chaînée.

Structure d'un élément de la pile

```
typedef float Ttype;
typedef struct ElementRepere{
  Ttype valeur;
   struct ElementRepere *suivant;
 }Element;
```

Structure de contrôle de la pile

```
typedef struct PileRepere{
    Element *tete;
    int nef;
 }Pile;
```


Créer un élément et une structure de contrôle de la pile

- Comme pour les listes chaînées, nous aurons besoin des fonctions **CréerElement()** et **CreerPile()**:

Fonction CreerElement()

```
Element* CreerElement( ){
  Element* El=(Element*)malloc(sizeof(Element));
  El→valeur=0;
  El→suivant=NULL;
  return(El);
 }
```

Fonction CreerPile()

```
Pile* CreerPile( ){
  Pile* Pi=(Pile*)malloc(sizeof(Pile));
  Pi→tete=NULL;
  Pi→nef=0;
  return(Pi);
 }
```

Empiler un élément dans la pile

- Voici l'algorithme d'insertion d'un élément dans la pile :
  - Déclaration de l''élément à insérer;
  - Allocation de la mémoire pour le nouvel élément;
  - Remplir le contenu du champ de donnée;
  - Mettre à jour le pointeur **tete** vers le 1er élément (le sommet de la pile);
  - Mettre à jour la taille de la pile.

Empiler un élément dans la pile

- Voici le code de la fonction **Empiler():**

Fonction Empiler()

```
void Empiler(Pile* Pi, Ttype vad){
  Element* El;
  El = CreerElement();  //appel à la fonction CreerElement()
  if(Pi == NULL || El == NULL){
    exit(EXIT_FAILURE);
  }
  El→valeur = vad;

  El→suivant = Pi→tete;
  Pi→tete = El;
  Pi→nef++;
}
```

Dépiler un élément de la pile

- Les étapes de suppression d'un élément de la pile sont:
  - Le pointeur **ptr** contiendra l'adresse du 1er élément à supprimer;
  - Le pointeur **tete** pointera vers le 2ème élément (après la suppression du 1er élément, le 2ème sera au sommet de la pile);
  - Mettre à jour la taille de la pile (la taille de la pile sera décrémentée d'un élément).

Dépiler un élément de la pile

- Voici le code de la fonction **Dépiler():**

Fonction Dépiler()

```
Ttype Depiler(Pile* Pi){
Ttype vsd;
Element* ptr;
if(Pi == NULL){
   exit(EXIT_FAILURE);
}

 //cas d'une pile vide
if(Pi→tete == NULL){
  printf("Impossible de dépiler");
}
```

Fonction Dépiler() suite

```
 //cas d'une pile non vide
 else{
   ptr = Pi→tete ;
   Pi→tete = Pi→tete→suivant;
   vsd=ptr →valeur;
   free(ptr);
   Pi→nef--;
   return(vsd);
 }
}
```

Afficher une pile

- La fonction **AfficherPile()** va nous afficher une pile.

Fonction AfficherPile ()

```
void AfficherPile (Pile* Pi) {
  Element* ptr;
  ptr= Pi→tete ;
  while(ptr  != NULL){          /*condition d'arrêt: fin de la pile*/
      printf("%f\n", ptr→valeur );
       ptr=ptr→ suivant;
  }
  printf("NULL\n");             // pour la forme
}
```


Vider une pile

- Voici les étapes pour vider une pile:
  - On dépile l'élément au sommet de la pile tant que la taille est non nulle;
  - À la sortie de la fonction on doit récupérer la mémoire occupée par la pile elle-même via la fonction **free().**

Vider une pile

- La fonction **ViderPile()** va nous vider une pile.

Fonction ViderPile()

```
void ViderPile (Pile* Pi) {
  while(Pi→nef >0){          /*condition d'arrêt*/
        Depiler(Pi);
   }
   free(Pi);
}
```

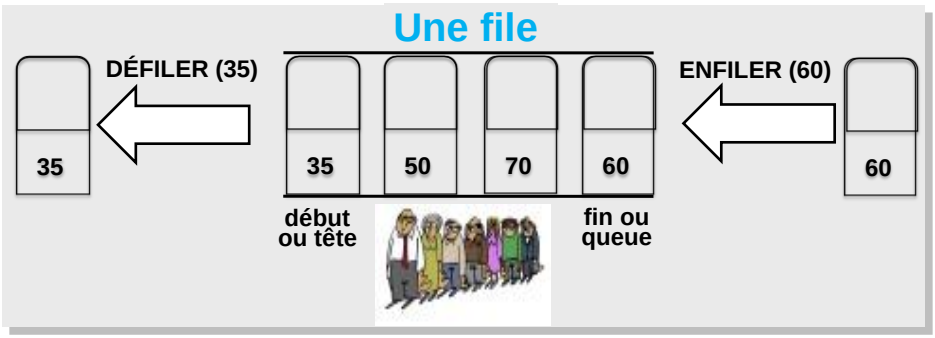

Partie II: Les files

Définition d'une file

- Une file est une structure de donnée linéaire permettant de stocker et de restaurer des données selon un ordre bien déterminé.
- 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.
- Les files sont très utilisées : File d'attente à un guichet, file de documents à imprimer, …

Principe d'une file

- Voyez la figure suivante:



- **ENFILER** (ENTRER): insérer un élément à la file.
- **DÉFILER** (SORTIR): supprimer un élément de la file.

- Une **file** est une **structure** dont la loi d'évolution respecte la règle **FIFO** (**F**irst **I**n **F**irst **O**ut) (Premier Entrée Premier Sortie).

Intérêt et applications

■ **Intérêt :**

- Type Abstrait de Données (**TAD**) linéaires et asymétrique.
- Permet de manipuler des ensembles dynamiques.

■ **Applications :**

- Les transactions bancaires sont mémorisées temporairement dans une **file** avant d'être traitées;
- Les requêtes dans les serveurs d'impression sont traitées dans l'ordre d'arriver dans une **file** d'attente;
- La création de toutes sortes de mémoires tampons (buffers) utilise les **files**.
- Un algorithme de parcours d'arbres en largeur utilise une **file** pour mémoriser les nœuds visités.
- etc.

Structure d'un élément et structure de contrôle de la file

■ Comme pour les piles, il faut le créer les structures d'une file:

Structure d'un élément de la file

```
typedef float Ttype;
typedef struct ElementRepere{
  Ttype valeur;
  struct ElementRepere *suivant;
 }Element;
```

Structure de contrôle de la file

```
typedef struct FileRepere{
    Element *tete;
    Element *queue;
    int nef;
 }File;
```

Créer un élément et une structure de contrôle de la file

- Nous aurons besoin des fonctions **CréerElement()** et **CreerFile()**:

Fonction CreerElement()

```
Element* CreerElement( ){
   Element* El=(Element*)malloc(sizeof(Element));
   El→valeur=0;
   El→suivant=NULL;
   return(El);
 }
```

Fonction CreerFile()

```
File* CreerFile( ){
   File* Fi=(File*)malloc(sizeof(File));
   Fi→tete=NULL;
   Fi→queue=NULL;
   Fi→nef=0;
   return(Fi);
 }
```


Insérer un élément dans la file

- Voici l'algorithme d'insertion d'un élément dans la file :
  - Déclaration de l''élément à insérer;
  - Allocation de la mémoire pour le nouvel élément;
  - Remplir le contenu du champ de donnée;
  - Mettre à jour le pointeur **queue** vers le dernier élément (la queue de la file);
  - Mettre à jour la taille de la file.

Insérer un élément dans la file

- Voici le code de la fonction **Enfiler():**

Fonction Enfiler()

```
void Enfiler(File* Fi, Ttype vaf){
  Element* El;
  El = CreerElement();
 if(Fi == NULL || El == NULL){
    exit(EXIT_FAILURE);
  }
  El→valeur = vaf;
  //cas d'une file vide
  if(Fi→tete==NULL){
     Fi→tete = El;
     Fi→queue = El;
  }
```

Fonction Enfiler() suite

```
  //cas d'une file non vide
  else{
     Fi→queue→suivant = El;
     Fi→queue = El;
  }
  Fi→nef++;
}
```


Supprimer un élément de la file

- Voici l'algorithme de suppression d'un élément de la file :
  - Le pointeur **ptr** contiendra l'adresse du 1er élément;
  - On retourne l'information à la tete;
  - Le pointeur **tete** pointera vers le 2ème élément (après la suppression du 1er élément, le 2ème sera au début de la file);
  - Mettre à jour la taille de la file (la taille de la file sera décrémentée d'un élément).

Supprimer un élément de la file

■ Voici le code de la fonction **Défiler():**

Fonction Defiler()

```
Ttype Defiler(File* Fi){
Ttype vsd;
Element* ptr;
if(Fi == NULL){
   exit(EXIT_FAILURE);
}

 //cas d'une file vide
if(Fi→tete == NULL){
  printf("Impossible de défiler");
}
```

Fonction Defiler() suite

```
//cas d'une file non vide
else{
   ptr = Fi → tete ;
   Fi→tete = Fi →tete→suivant;
   vsd=ptr →valeur;
   free(ptr);
   Fi → nef--;

   if(Fi→tete==NULL){
      Fi→queue=NULL;
   }
   return(vsd);
 }
}
```


Afficher une file

■ La fonction **AfficherFile()** va nous afficher une file.

Fonction AfficherFile ()

```
void AfficherFile (File* Fi) {
   Element* ptr;
   ptr = Fi→tete ;
   printf("Fi = ");                  // pour la forme
   while(ptr  != NULL){      /*condition d'arrêt: fin de la file*/
        printf("%f →", ptr→valeur);
        ptr = ptr→suivant;
    }
   printf("NULL\n");
 }
```

Vider une file

- Voici les étapes pour vider une file:
  - On défile l'élément en tête de la file tant que la taille est non nulle;
  - À la sortie de la fonction, on doit récupérer la mémoire occupée par la file elle-même via la fonction **free().**

Vider une file

- La fonction **ViderFile()** va nous vider une file.

Fonction ViderFile()

```
void ViderFile (File* Fi) {
  while(Fi→nef >0){          /*condition d'arrêt*/
        Defiler(Fi);
   }
   free(Fi);
}
```