

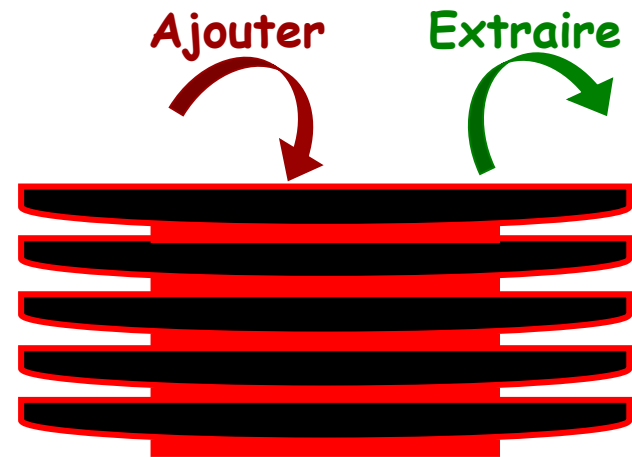
# Chapitre4. LES PILES ET FILES

- LES PILES : DÉFINITION
- REPRÉSENTATION DES PILES
- OPÉRATIONS DE BASE SUR LES PILES
- LES FILES : DÉFINITION
- REPRÉSENTATION DES FILES
- OPÉRATIONS DE BASE SUR LES FILES



# 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)** : en français *Dernier Entré Premier Sorti*
  - **Exemple : une pile d'assiette**
    - 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



## 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**

□ **listes chaînées ?**



## 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

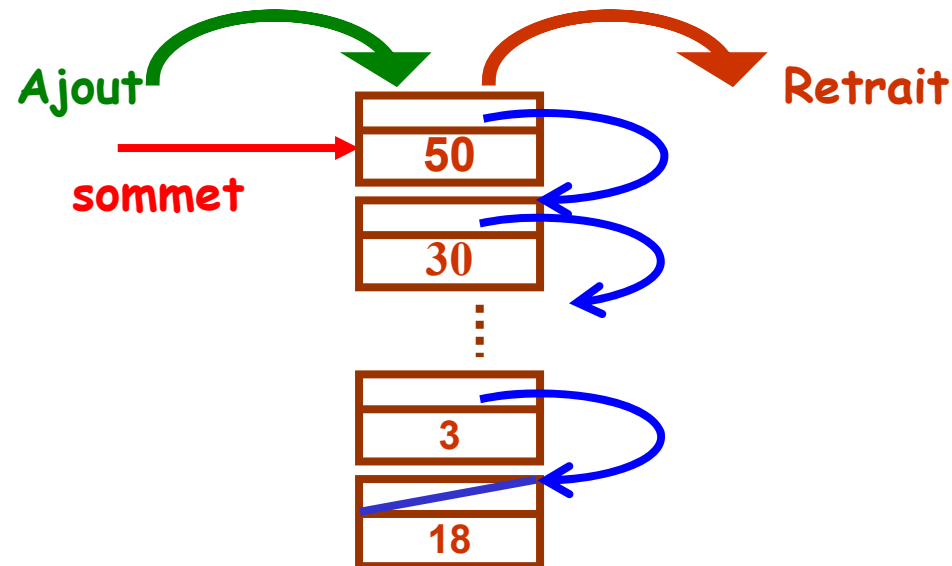
```
typedef struct pile {  
    int donnee; /*La donnée que notre pile stockera*/  
    struct pile *precedent; /*Pointeur vers l'élément précédent de la  
                           pile */  
} Pile;
```



## 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 en quelque sorte *emboîtées* les unes sur les autres

□ Le pointeur *sommet* devra toujours pointer vers le sommet de la pile

### 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 pile {  
    char *donnee;  
    struct pile *precedent;  
} Pile;  
  
/* les variables globales*/  
Pile *sommet;  
int taille;
```

# 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;
```

```
}
```

### 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 :

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

```
int pile_push(char *donnee);
```

Algorithme de la fonction :

- déclaration d'élément(s) à insérer
- allocation 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





### 3. OPÉRATIONS DE BASE SUR LES PILES

#### □ Ajout d'un nouvel élément

**Fonction** /\* empiler (ajouter) un élément dans la pile \*/

```
int pile_push (char *donnee) {  
    Pile *nouvel_element;  
    if ((nouvel_element = (Pile *) malloc(sizeof(Pile))) == NULL) return -1;  
    nouvel_element->donnee = (char *) malloc(50*sizeof(char));  
    strcpy (nouvel_element->donnee, donnee);  
    nouvel_element->precedent = sommet;  
    sommet = nouvel_element;  
    taille++;  
    return 0;  
}
```



### 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 trouvant au sommet de la pile)

Prototype de la fonction :

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

```
int pile_pop();
```

Algorithme de la fonction :

- le pointeur `supp_elem` 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

### 3. OPÉRATIONS DE BASE SUR LES PILES

#### □ Retrait d'un élément

**Fonction** /\*dépiler (supprimer) un élément de la pile\*/.

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

### 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

```
affiche () {  
    Pile *courant; int i;  
    courant = sommet;  
    for(i=0;i<taille;++i) { printf("  %s\n", courant->donnee);  
                           courant = courant->precedent;  
    }  
}
```



### 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 : `pile_clear( );`

Algorithme de la fonction

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

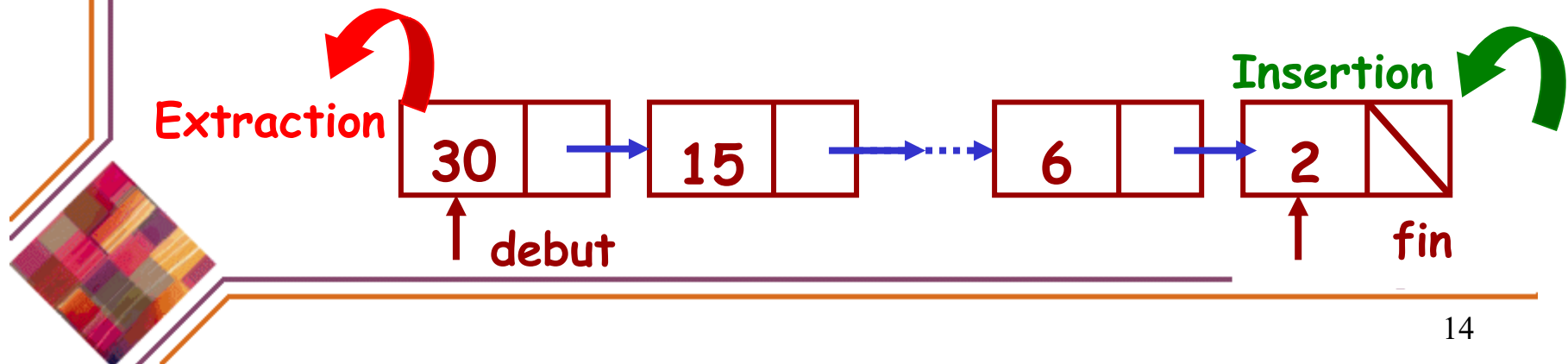
Fonction

```
pile_clear() { while (sommet != NULL) pile_pop(); }
```



## 4. 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**) en français *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)



## 5. 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 explique par le fait que nous pointons à la base de la file)

```
typedef struct file {
```

```
    int info;    /*La donnée que notre file stockera */
```

```
    struct file *suivant; /*Pointeur vers l'élément suivant de la
```

```
    } File;                                           file */
```

## 5. 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 de suppression des données `File *debut`
- le dernier élément : se trouve à la fin de la file et il permet de réaliser l'opération d'insertion des données `File *fin`
- le nombre d'éléments : `int taille`

□ On suppose que la File est déclarée de la façon suivante :

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



## 6. LES OPÉRATIONS DE BASE SUR LA FILE

### □ Initialisation

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`

### Fonction

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

```
}
```

## 6. LES OPÉRATIONS DE BASE SUR LA FILE

### □ Insertion d'un nouvel élément

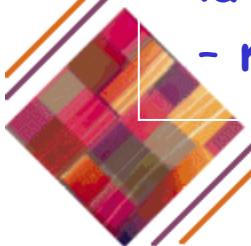
Prototype de la fonction :

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

```
int file_push(char *donnee);
```


Algorithme de la fonction :

- déclaration d'élément(s) à insérer
- allocation 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



## 6. LES OPÉRATIONS DE BASE SUR LA FILE

### □ Insertion d'un nouvel élément

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

```

## 6. 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



## 6. LES OPÉRATIONS DE BASE SUR LA FILE

### □ Suppression d'un élément

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

```
int file_pop () {  
    File *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;  
}
```



## 6. LES OPÉRATIONS DE BASE SUR LA FILE

### □ Affichage de la file

#### Algorithme de la fonction

- 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 affiche () {  
    File *courant; int i;  
    courant = debut;  
    for(i=0;i<taille;++i) { printf("  %s\n", courant->donnee);  
                           courant = courant->suivant; }  
}
```



## 6. LES OPÉRATIONS DE BASE SUR LA FILE

### □ Vidage de la file

□ Il s'agit d'une fonction permettant d'effacer la File

### Algorithme de la fonction

Tant que le pointeur **fin** n'est pas **NULL**

Effacer le premier élément de la file

### Fonction

```
file_clear() { while (fin != NULL) file_pop(); }
```



## 7. APPLICATIONS

□ **Exercice 1** : Afficher le nombre d'éléments pairs et impairs dans une pile d'entiers implémentée sous forme d'une liste simplement chaînée.

□ **Exercice 2** : Rechercher le maximum dans une file d'entiers implémentée sous forme d'une liste simplement chaînée.

