

# Chapitre1: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



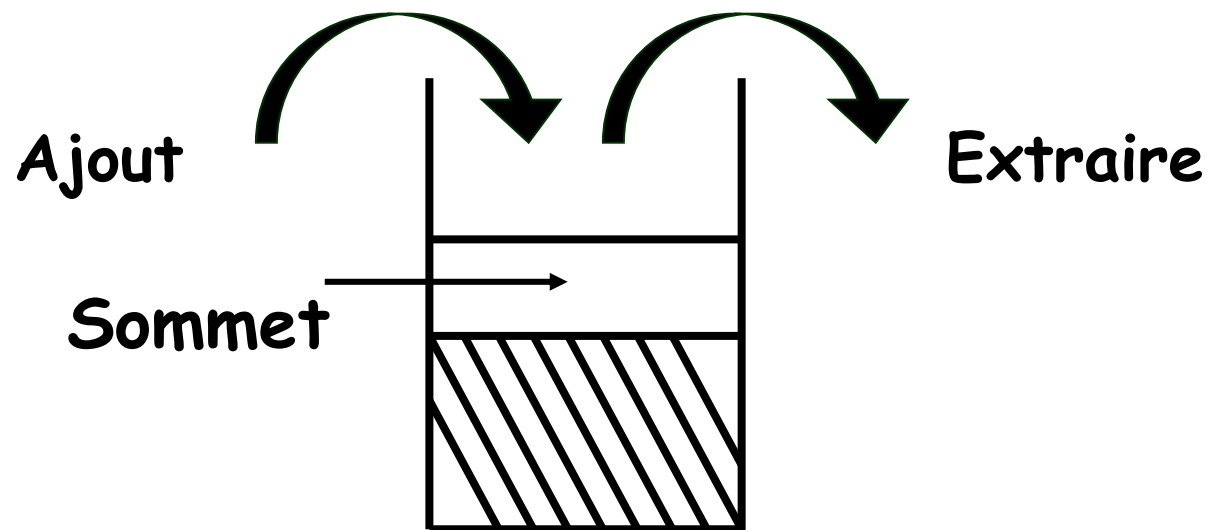
# Chapitre1:LES PILES ET LES FILES

## 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, une variable globale indiquant le sommet

- Un enregistrement avec deux champs

- **représentation dynamique**

- **listes chaînées ?**

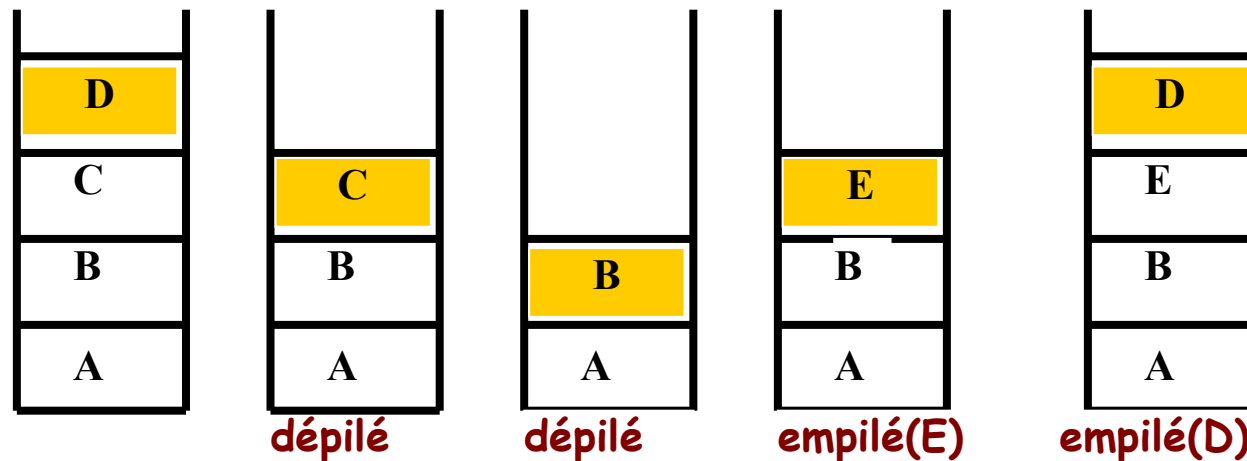


# 1. PILE: DÉFINITION ET REPRÉSENTATION

□ **Exemple : une pile d'assiette.** 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 Push : insérer l'élément au sommet de la pile

```
void Push(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 Pop : extraction (dépilement) de l'élément qui est au sommet de la pile et le retourne comme valeur de la fonction

element Pop(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 : initialiser 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);
```

```
}
```

### 3. EXEMPLE D'APPLICATION

#### □ traitement des expressions mathématiques

□ Si on a plus d'un délimiteur par exemple : ( [ {

□ Pour chaque type de délimiteur, il faut:

□ Empiler : un délimiteur ouvert

□ Dépiler : un délimiteur fermé

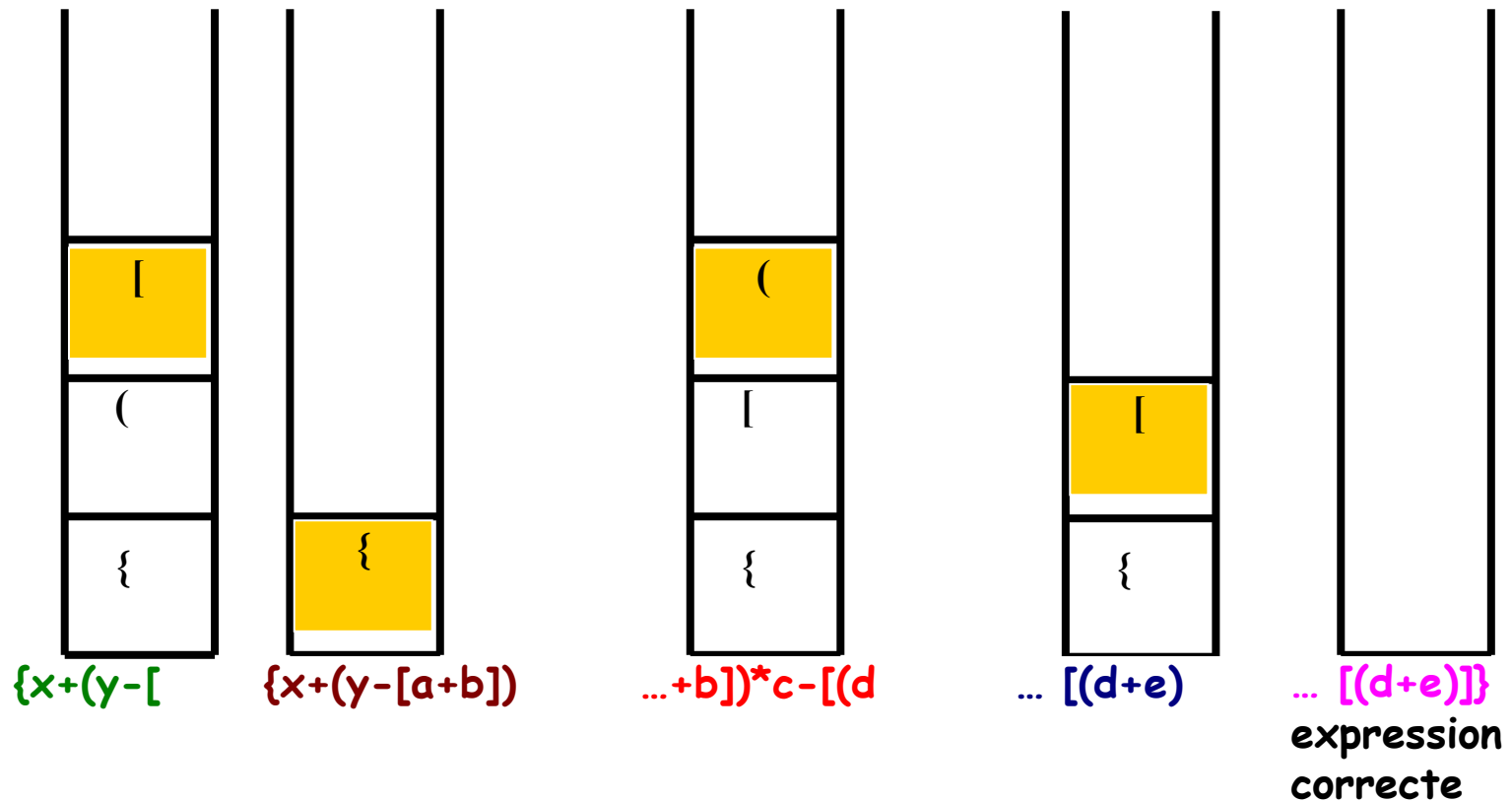
□ le dernier délimiteur ouvert et le premier à être fermé (LIFO).

□ La résolution de ce problème peut être donc simulée par la structure de la pile



### 3. EXEMPLE D'APPLICATION

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



### 3. EXEMPLE D'APPLICATION

- **Pb d'underflow** : Pile\_vide et on essaie de dépiler : correspond à une fermeture de plus
- **Pb d'overflow** : Dépassement de la capacité en nombre de délimiteur ouvrant et fermant
- 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** qu'un symbole est valide (expression correcte) **faire**

si (symbole) est un délimiteur ouvrant alors empiler (symbole)  
**finsi**

si (symbole) est un délimiteur fermant alors  $S \leftarrow$  **dépiler**

si  $S$  ne correspond pas au délimiteur fermant alors  
l'expression est incorrecte (invalid)

**finsi**

**fsi**

passer au symbole suivant

**fin de tant que**

si la pile est vide alors l'expression est correcte

sinon l'expression est incorrecte

**fsi**

### 3. EXEMPLE D'APPLICATION

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

typedef char element;
typedef element Pile[Max];

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



### 3. EXEMPLE D'APPLICATION

□ Fonction associe 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;}
```



### 3. EXEMPLE D'APPLICATION

```
main() {
```

```
    element expression[4*Max];
```

```
    element symbole, S;
```

```
    int lon,i, Valide;
```

```
    printf("Entrer une expression mathématique");
```

```
    gets(expression);
```

```
    initialisation();
```





### 3. EXEMPLE D'APPLICATION

```
lon=strlen(expression); i=0; Valide=1;
while ((i<=lon-1) && (Valide==1)) {
    symbole=expression[i];
    if ((symbole=='[' )||(symbole=='(' )||(symbole=='{' )) Push(symbole);
    if ((symbole==']' )||(symbole==')' )||(symbole=='}' ))
        { S=Pop();
          if (S!= ouvrant(symbole)) Valide=0;
        };
    i++;
} /*fin de while*/
if (Pile_vide() && (Valide==1)) printf("expression correcte");
else printf("expression incorrecte");
}
```



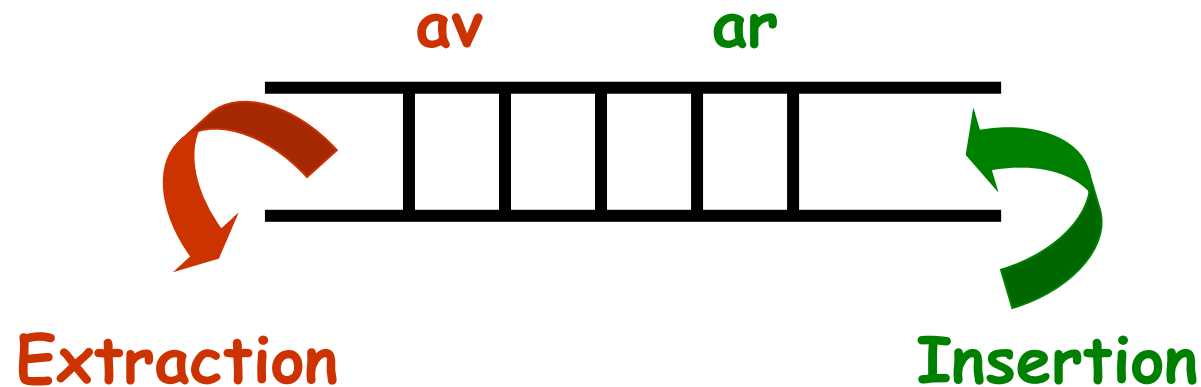
# Chapitre1:LES PILES ET LES FILES

## 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



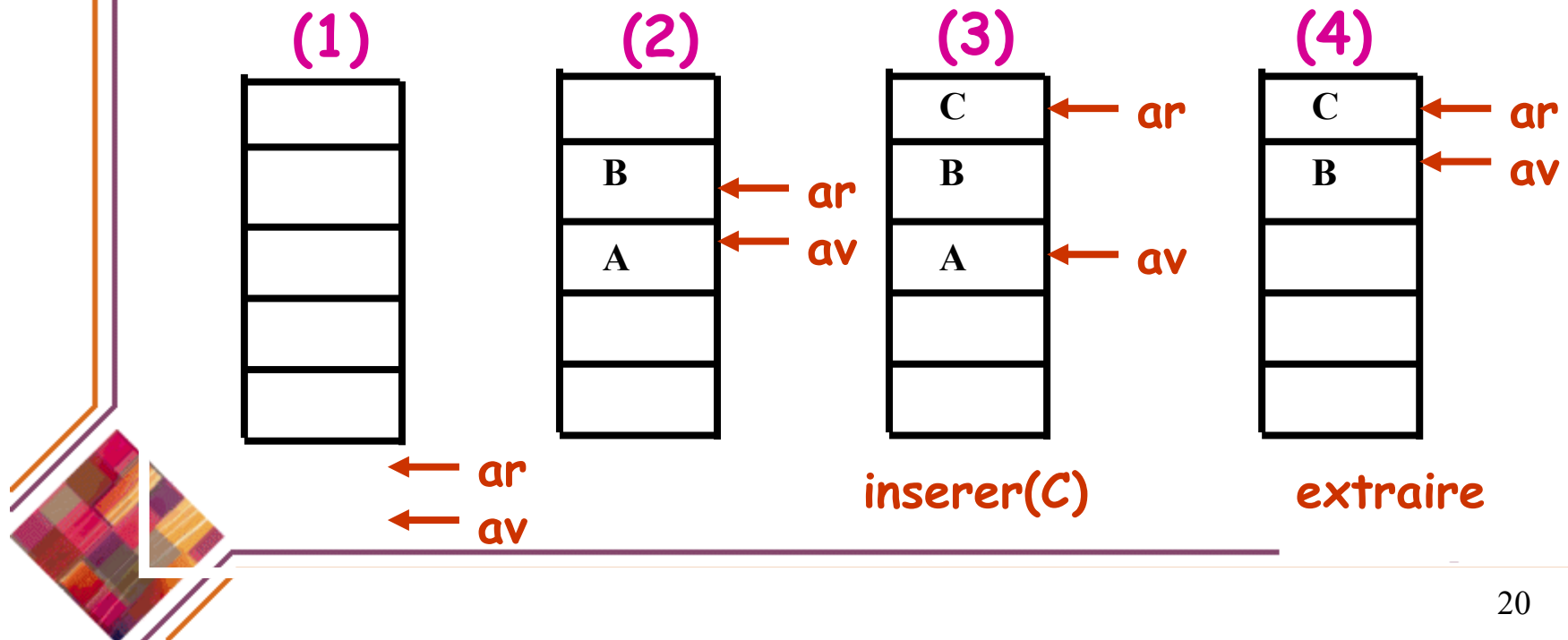
## 4. FILE : DÉFINITION ET REPRÉSENTATION

□ **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**, **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

$$ar \leftarrow -1 \quad av \leftarrow -1$$

□ **insertion** de l'élément de  $x$      $ar \leftarrow ar+1$      $F[ar] \leftarrow x$

□ **extraction** de l'élément de  $x$      $x \leftarrow F[av]$      $av \leftarrow av+1$

□ **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

□ une première solution serait de modifier **extraire** de sorte lorsqu'un élément est modifié 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

□ initialisation :  $ar = -1$

□ file est vide :  $ar < 0$

□ file est pleine :  $ar \geq n - 1$

element extraire() {

Element x; int i;

if file\_vide() printf("Underflow");

else {

$x = F[0];$

    for( $i = 0; i \leq ar - 1; i++$ )  $F[i] = F[i + 1];$

$ar = ar - 1;$

    return x;     }

}

## 5. LES OPÉRATIONS SUR LES FILES

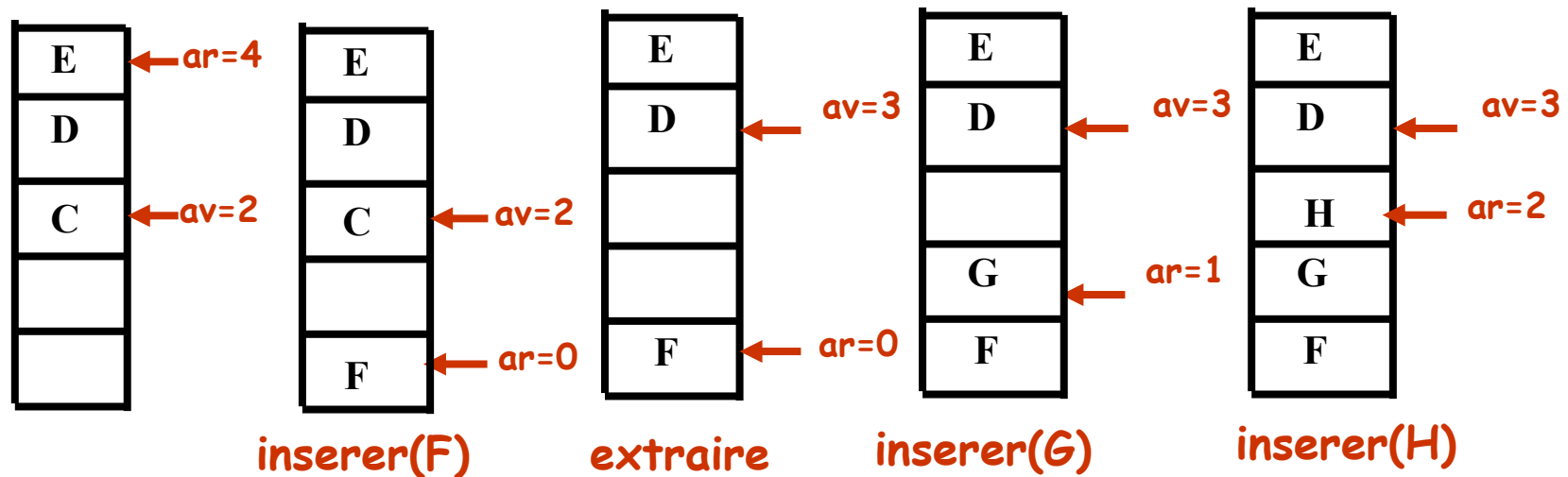
```
void inserer(element x) {  
    if (file_pleine()) printf("Overflow");  
    else  
        {F[ar]=x;  
         ar=ar+1;  
        }  
}
```





## 5. LES OPÉRATIONS SUR LES FILES

□ une **seconde solution** consiste à voir la **FILE** comme un ensemble circulaire d'éléments  $F[0], F[1], F[2], \dots, F[n-1], F[0]$



□ il n'y a pas perte de place, on exploite tout le tableau. On aura un **Overflow** lorsque  $(ar+1) \text{ modulo } n == av$

## 5. LES OPÉRATIONS SUR LES FILES

- **initialisation** :  $av = -1$  et  $ar = -1$
- **file est vide** :  $ar \leq av$
- **file est pleine** :  $(ar+1) \% n == av$

```
void extraire(element *x) {  
    if file_vide() printf("Underflow");  
    else {  
        x=F[av];  
        if( av==n-1) av=0; else av=av+1;  
    }  
}
```



## 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 sans 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<=av);
```

```
}
```

```
int file_pleine()
```

```
{
```

```
    return ((ar+1)%max==av);
```

```
}
```



## 6. EXEMPLE D'APPLICATION

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



## 6. EXEMPLE D'APPLICATION

```
void enfiler(int x)
{   printf("ok enfiler \n");
    if (ar==max-1) ar=0; else ar=ar+1;
    if (av==ar) {
        printf("Overflow");
        if (ar==0) ar=max-1; else ar=ar-1;
    }
    else 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. 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 par décalage)

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

- initialisation()
- int file\_vide()
- int file\_pleine()
- void enfiler(int)
- int defiler()
- main()

