STRUCTURES DE DONNÉES
MIP — S4

Pr. K. Abbad & Pr. Ad. Ben Abbou & Pr. A. Zahi
Département Informatique
FSTF
2019-2020

Séance 5 - Les Files

- Définition
- Exemple
- Opérations de base
- Représentation contigüe
- Représentation chainée(non contigüe)

Les Files — *Définition*

- Notion intuitive — *FILE d'objets*

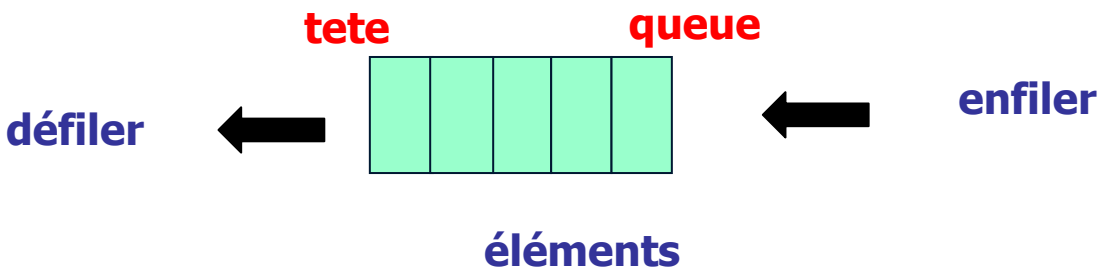
➡ l'ajout d'un élément se fait à la queue.
le retrait d'un élément se fait à la tête .

Les Files — *Définition*

- Une **FILE** est une **structure** destinée au stockage des données en cours de traitement par un programme.

- Une **file** est une **liste particulière** fondée sur le principe premier entrée premier sorti (**F**irst **I**n **F**irst **O**ut) **.**

L'ajout à **la fin(queue)** et le retrait se fait au **début (tete).**



Les Files — *Utilisation*

- Exemple:
  - Les fichiers à imprimer se mettent en file d'attente, de façon que le premier fichier envoyé soit le premier à être imprimé.

Les Files — *Opérations de base*

- **Enfiler** — *Ajoute un élément à la fin de file.*
- **Defiler** — *Supprime l'élément du début de la file* ***si la file est non vide.***
- **Queue** — *Récupère l'élément qui se trouve en queue de la file* ***si la file est non vide.***
- **Tete** — *Récupère l'élément qui se trouve dans la tête de la file* ***si la file est non vide.***
- **InitFile** — *Crée une file vide.*
- **FileVide** — *Détermine si une file est vide ou non.*

Les files — *Représentation*

- Représentation contiguë
  - Implémentation par un tableau.
- Représentation chainée
  - Ensemble d'éléments liés séquentiellement (par des pointeurs)

Les Files — *Représentation contigüe*

- Implémentation par un tableau.

- La file est représentée par une structure définie par:
  - tete — un entier qui représente l'indice de l'élément qui se trouve en tète de la file .
  - queue — un entier qui représente l'indice de l'élément qui se trouve en queue de la file .
  - Tab — un tableau d'éléments de la file .

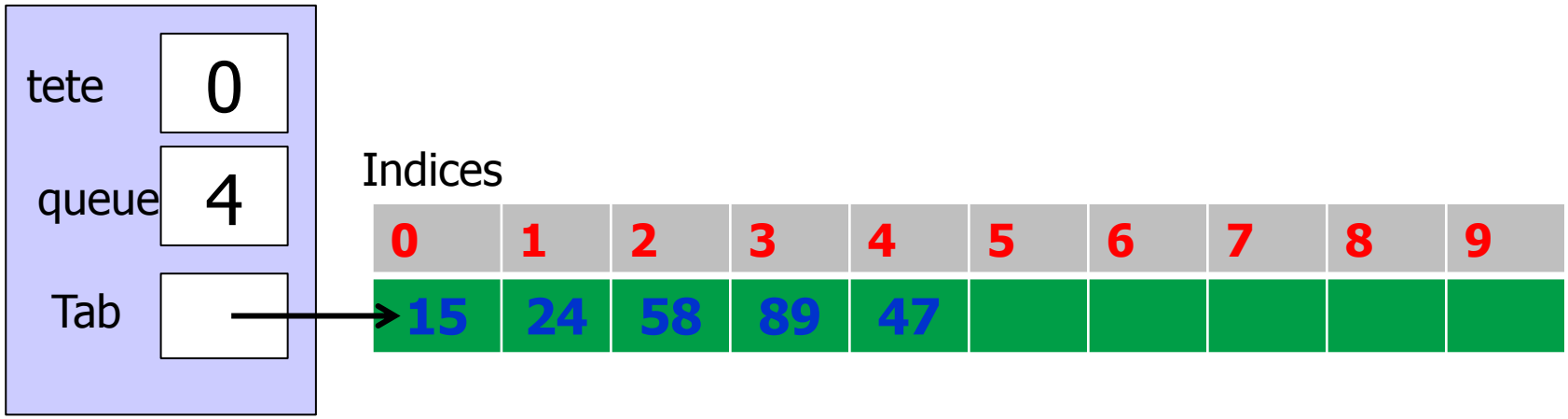
Les Files — *Représentation contigüe*

- Définition d'une FILE d'entiers.

```
#define CAP 10
typedef struct {
        int tete;
        int queue;
        int Tab[CAP];
 } FILE;
```

Les Files — *Représentation contigüe*

- **Exemple**:



Les Files — *Représentation contigüe*

- Le tableau peut être :
  - statique — *la taille maximale est fixée.*
  - dynamique — *réservation dynamique de la mémoire et la taille maximale peut être modifiée.*

- Il faut faire un contrôle de taille lors de l'enfilement pour gérer le dépassement de capacité.

Utilisation de *realloc* pour redimensionner le tableau

Les Files — *Représentation contigüe*

- Initialisation de la File —*la queue et la tete est mis à l'indice -1*

```
FILE InitFile  ()
  {
       FILE F;
       F.queue =-1;
       F.tete  =-1;
       return F;
  }
```

- File vide ? —*la tete est à l'indice -1*

```
int FileVide(FILE F)
  {
       if(F.tete  ==-1)
              return 1;
       return 0;
  }
```

Les Files — *Représentation contigüe*

- Récupérer la valeur du Tete
  - Il faut vérifier si la file n'est pas vide avant de prendre la valeur du Tete.
  - La fonction retourne 1 si la valeur est prise et 0 sinon

```
int Tete(FILE F, int *x)
  {
       if (!FileVide(F)) {
             *x = F.Tab[F.tete];
             return 1;
       }
       return 0;
  }
```

Les Files — *Représentation contigüe*

- Récupérer la valeur en Queue
  - Il faut vérifier si la file n'est pas vide avant de prendre la valeur du Queue.
  - La fonction retourne 1 si la valeur est prise et 0 sinon

```
int Queue(FILE F, int *x)
  {
       if (!FileVide(F)) {
              *x = F.Tab[F.queue];
              return 1;
       }
       return 0;
  }
```

Les Files — *Représentation contigüe*

- Enfiler un élément
  - Il faut vérifier si la file n'est pas pleine avant d'insérer l'élément.
  - La fonction retourne 1 si la valeur est insérée et 0 sinon

```
int Enfiler(FILE *F, int x)
  {
       if (F->queue!= CAP-1) {
              F->tete=0; /*si la file est vide*/
              F->queue++;
              F->Tab[F->queue]=x;
              return 1;
       }
       return 0;
  }
```

Les Files — *Représentation contigüe*

- Défiler un élément
  - Il faut vérifier si la file n'est pas vide avant de retirer l'élément au début .
  - La fonction retourne 1 si la valeur est retirée et 0 sinon.

```
int Defiler(FILE *F) {
int i;
if (!FileVide(*F)) {
  if(F->queue== F->tete) F->queue= F->tete=-1;/*un seul elt*/
  else{
        for(i=1;i<=F->queue ; i++)
              F->Tab[i-1]= F->Tab[i];
              F->queue--;
   }
   return 1;
 }
     return 0;
}
```

Les Files — *Représentation contigüe*

- Exemple

```
#define CAP 10
typedef struct {
         int queue, tete;
         int Tab[CAP];
} FILE;

PILE InitFile  ()
  {...}
int FileVide(FILE F)
  {...}
int Queue(FILE P,int *x)
  {...}
int Enfiler(FILE *F,int x)
  {...}
int Defiler(FILE *F)
  {...}
```

```
int main(){
        FILE F;
        int a,b,c,x;
        F= InitFile();
        int i=0;
        while(i<4){
             scanf("%d",&x);
             Enfiler(&F,x);
             i++;
        }
        printf("\n %d  ",F.tete);
        printf("\n %d  ",F.queue);
         while(!FileVide(F)) {
             b= Tete(F,&x);
             c= Defiler(&F);
              printf("\n X : %d \n",x);
        }
    getch();     return 0;
}
```

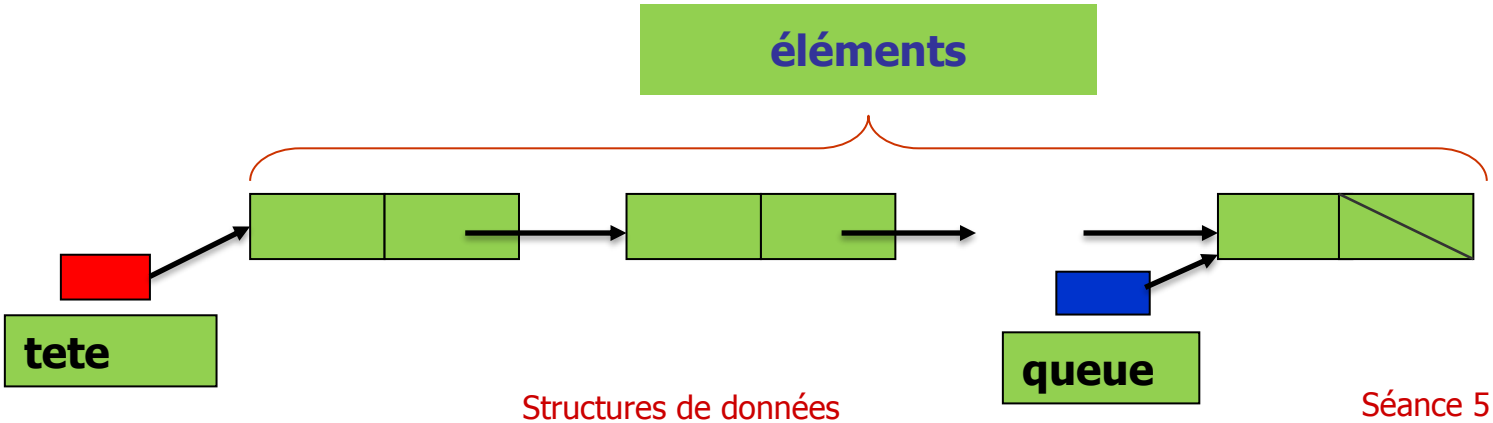
Les Files — *Représentation contigüe*

- La représentation contigüe par un tableau présente les inconvénients suivants :
  - La taille maximale doit être fixée.
  - les éléments du tableau sont stockés dans des cases mémoires adjacentes.
  - La difficulté de redimensionner le tableau.
    - La fonction *realloc* est très coûteuse en temps .
  - Le décalage des éléments de la file après le retrait de l' élément de la tête.

Représentation chaînée

Les Files — *Représentation Chaînée*

- Une file est une suite de maillons distribués et organisés séquentiellement.
- Un maillon possède :
  - les données d'un élément.
  - un lien (un pointeur) vers son successeur.
- La File est identifiée par sa tête (adresse du premier maillon) et par sa queue (adresse du dernier maillon)
- Les opérations sur la file sont basées sur la manipulation des liens (pointeurs) .



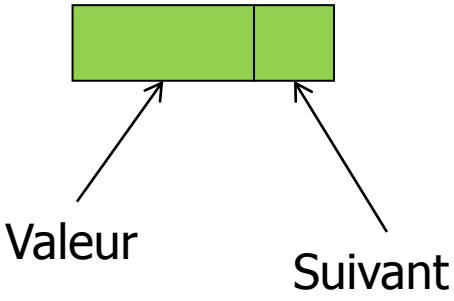
Les Files — *Représentation Chaînée*

- Définition d'une File d'entiers

```
/* type maillon */
   typedef struct Maillon{

          int valeur;
          struct Maillon * suivant;
    } maillon;

/* type FILE */
typedef struct {
        maillon *tete;
        maillon *queue;
} FILE;
```



tete : pointe sur le premier élément de la file

queue : pointe sur le dernier élément de la file

Les Files — *Représentation Chaînée*

- Initialisation de la File — *tete et queue de file sont à NULL*

```
FILE InitFile()
  {    FILE F;
       F.tete=NULL;
       F.queue=NULL;
       return F;
  }
```

- File vide ? —*tete et queue de file sont à NULL*

```
int FileVide(FILE F)
  {
       if(F.tete=== NULL && F.queue==NULL )
              return 1;
       return 0;
  }
```

Les Files — *Représentation chaînée*

- Récupérer la valeur de la tete
  - Il faut vérifier si la file n'est pas vide avant de prendre la valeur de la tete.
  - La fonction retourne 1 si la valeur est prise et 0 sinon

```
int TETE(FILE F, int *x)
  {
       if (!FileVide(F)) {
              *x = F.tete->valeur;
              return 1;
       }
       return 0;
  }
```

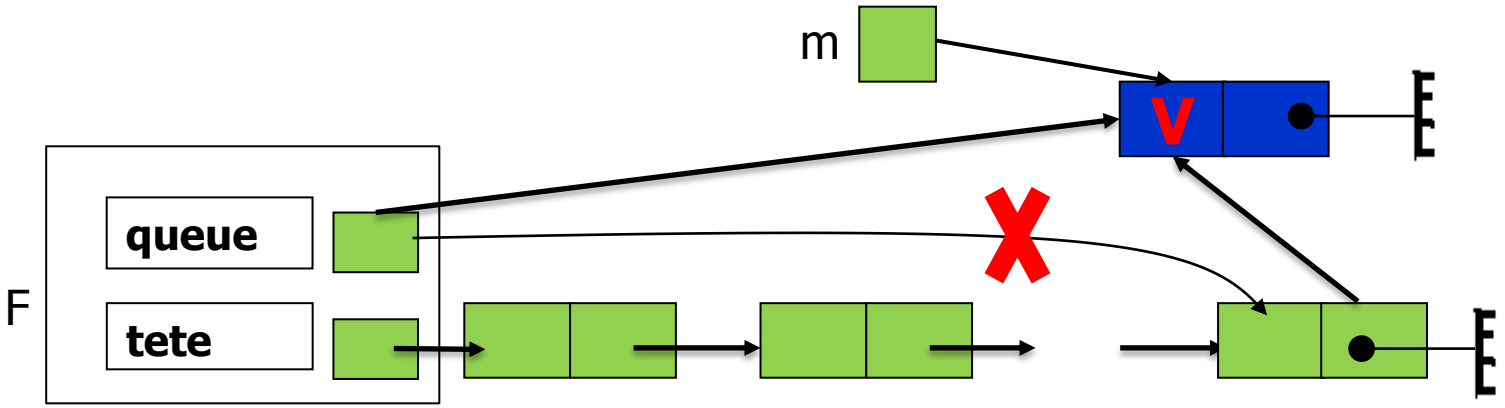
Les Files — *Représentation chaînée*

- Récupérer la valeur de la queue
  - Il faut vérifier si la file n'est pas vide avant de prendre la valeur de la queue.
  - La fonction retourne 1 si la valeur est prise et 0 sinon

```
int Queue(FILE F, int *x)
  {
       if (!FileVide(F)) {
              *x = F.queue->valeur;
              return 1;
       }
       return 0;
  }
```

Les Files — *Représentation chaînée*

- Enfiler un élément V dans la FILE tete
  - créer un maillon avec la valeur de l'élément
  - Ajouter l'élément à la queue de la liste .

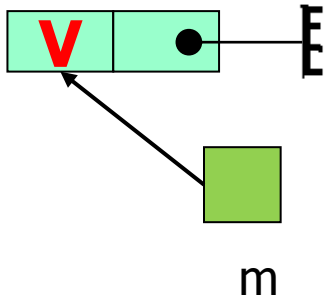



```
maillon * m;
1) M= creerMaillon(V);
2) F.queue->suivant=m;
3) F.queue=m;
```

Les Files — *Représentation chaînée*

- Créer maillon d'une file
  - Réserver l'espace mémoire pour l'élément
  - Remplir les champs du maillon : valeur par V et le pointeur par NULL
  - La fonction retourne l'adresse de l'espace réservé et NULL si l'espace n'est pas réservé

```
maillon* creerMaillon(int V)
{
  maillon *m;
  m=(maillon *)malloc(sizeof(maillon);
     if(m!=NULL){
               m->valeur = V;
               m->suivant = NULL;
     }
        return m;
  }
```



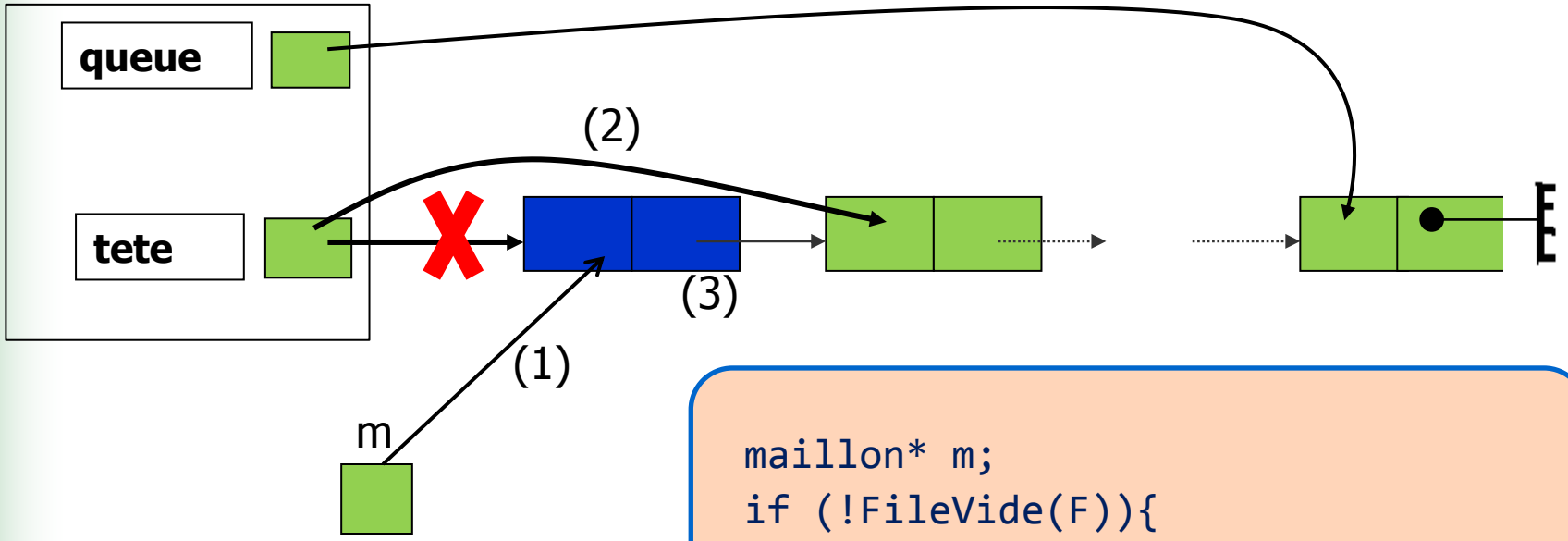
Les Files — *Représentation chaînée*

- **Enfiler un élément**
  - créer un maillon avec la valeur de l'élément
  - Ajouter l'élément à la queue de la liste .
  - La fonction retourne 1 si la valeur est insérée et 0 sinon

```
int Enfiler(FILE *F, int V) {
      maillon* m;
      m=creerMaillon(V);
      if(m!=NULL){
            creerMaillon(V);
            if(!FileVide(*F)){
                  F->queue->suivant = m;
                  F->queue = m ;
            }
            else   F->tete =F->queue=m;
            return 1
      }
      return 0;
}
```

Les Files — *Représentation chaînée*

- Défiler un élément dans la FILE tete




```
maillon* m;
if (!FileVide(F)){
        m=F.tete; //1
        F.tete=F.tete->suivant; //2
        free(m); //3
}
```

Remarque:

Si la File est vide on fait rien

Les Files — *Représentation chaînée*

- Défiler un élément
  - créer un pointeur m, tel que m et ~~p~~ **tete** pointent sur la même zone
  - Pointer la tête sur la zone pointée par le suivant de tête
  - Libérer la zone pointée par m
  - Si la file contient un seul élément, tête et queue doivent être à NULL
  - La fonction retourne 1 si l'élément de la tête est supprimé et 0 sinon

```
int defiler(FILE *F){
 maillon* m;
 if (!FileVide(*F)){
         m=F->tete;
         if(F->tete=F->queue){
                  F->tete=NULL; F->queue=NULL; }
         else     F->tete=F->tete->suivant;
         free(m);
         return 1
  }
  return 0;
}
```

Les Files — *Représentation chaînée*

- Exemple

```
typedef struct {
        int valeur;
        struct maillon *suivant;
} maillon;
typedef struct {
        maillon *tete;
        maillon *queue;
} FILE;

PILE InitFile  ()
  {...}
int FileVide(PILE P)
  {...}
int tete(FILE F,int *x)
  {...}
int Enfiler(FILE *F,int x)
  {...}
int Defiler(FILE *F)
  {...}
```

```
int main(){
        FILE F;
        int a,b,c,x;
        F= InitFile();
        int i=0;
        while(i<8){
             Enfiler(&F,i+1);
             i++;
        }
        while(!FileVide(F)) {
             b= Tete(F,&x);
             c= defiler(&F);
             printf("\n X : %d",x);
    }
        getch();
        return 0;
}
```