Chapitre II

Structure de données
Les Piles et les Files

1. Introduction

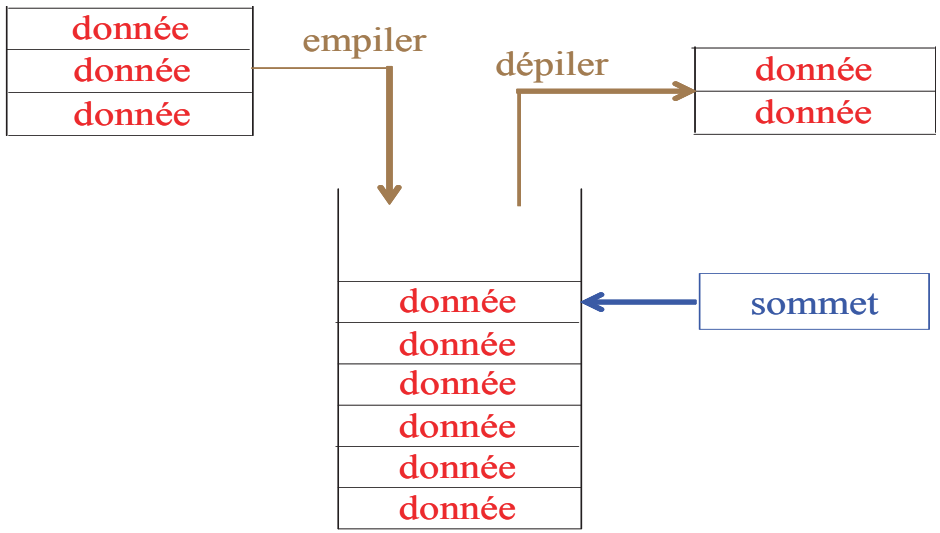
- Les *piles* (en anglais ***stack***) et les *files* (en anglais ***queue***) sont des types de listes (ou de structures de données) particuliers qui servent à stocker et manipuler des données dans des ensembles dynamiques.
- Leur particularité résident dans leur gestion des opérations d'insertion et de suppression des éléments dans les ensembles qui manipulent. En effet,
  - dans une pile, l'élément supprimé est le dernier inséré,
  - dans une file, l'élément supprimé est toujours le plus ancien.
- Par conséquent, dans une pile ou une file on ne peut accéder qu'à un seul élément de l'ensemble traité.
- Les piles et les files ont beaucoup d'applications dans différentes domaines notamment en informatique.
- Il existe plusieurs manières efficaces d'implantation, en langage C, des piles et des files.
- Dans ce chapitre, nous allons présenter comment les implanter à l'aide des pointeurs, des tableaux et des listes chaînées (seulement pour les files) .

2. Piles : 2.1 présentation générale

- Une pile est une structure de donnée où il n'est possible d'ajouter ou de supprimer des éléments qu'à une seule extrémité appelée ***sommet***. En effet,
- Le dernier élément ajouté devient le sommet de la pile et il sera le seul élément accessible.
- Lorsque on désire supprimer un élément de la pile, on doit supprimer le sommet, et le dernier élément ajouté avant lui devient alors le sommet de la pile.
- C-à-d, *le premier élément supprimé est le dernier élément ajouté dans la pile*. Cette structure est connue sous le nom de ***LIFO*** (***Last In, First Out***).
- Pour résumer, une pile permet la mis en attente d'informations ou de données dans le but de les récupérer plus tard, dans l'ordre inverse d'arrivée.
- La notion de pile est une structure très utilisée en informatique, surtout dans les langages de programmation, en effet
  - elle sert à implémenter les appels de fonctions (sous-programmes),
  - gère la récursivité,
  - permet le calcul des expressions arithmétiques,
  - etc.
- Généralement, il y a deux façons d'implanter une pile en langage C :
  - soit en utilise un pointeur indiquant le sommet de la pile,
  - soit on utilise un tableau.

Opérations sur les piles :

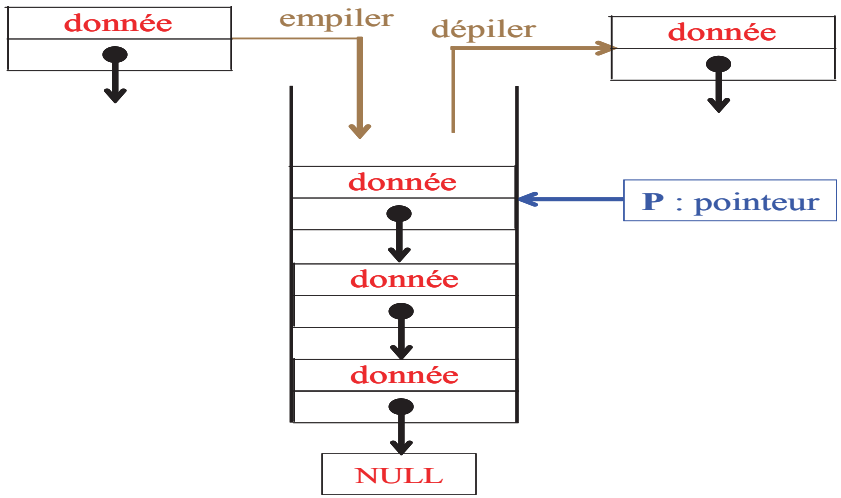
- Dans une pile, on ne peut autoriser que 5 opérations :
  1. *tester* si la pile est vide,
  2. *empiler* un élément, c-à-d l'ajouter au sommet de la pile,
  3. *dépiler* un élément, c-à-d le supprimer du sommet de la pile,
  4. *récupérer* le dernier élément de la pile, c-à-d acceder au sommet de la pile,
  5. *vider* le contenue de la pile.



Pile (LIFO)

Déclaration d'une pile :

- Comme le cas des listes, le langage C permet de réaliser des piles à l'aide des structures.
- En effet, chaque cellule de la pile est un objet constitué :
  - d'un champ de données,
  - d'un pointeur vers la cellule précédente.
- La déclaration correspondante est la suivante, où l'on suppose ici que les valeurs stockées dans la pile sont de type entier.



Pile (LIFO)

Code en C

```
typedef int element;
struct cellulePile
 {
    element valeur;
    struct cellulePile * precedent;
 };
typedef struct cellulePile cellulePile, * Pile;
```

Création et initialisation d'une pile :

- Pour créer une pile, on doit l'initialiser à NULL, cela va nous permettre d'allouer la première cellule.
- L'initialisation d'une pile, ou la création d'une pile vide, est faite par la fonction suivante :

```
Pile pileVide( )
  {
   return NULL;
  }
```

- Pour tester si une pile est vide, on utilise la fonction suivante :

```
 int testerPileVide(Pile p)
 {
   return p==NULL;
 }
```

Ajouter un élément dans la pile (empiler) :

- Il n'est possible d'insérer qu'une seule valeur en tête (ou sommet) de la pile.
- La valeur, une fois insérée, devient le sommet de la pile.

Code en C

```
Pile empiler(element val, Pile p)
  {
    Pile pn;
    pn=malloc(sizeof(cellulePile));
    pn->valeur=val;
    if(p==NULL)
     {
        pn->precedent=NULL;
        return pn;
     }
    else
     {
        pn->precedent=p;
        p=pn;
        return p;
     }
  }
```

Supprimer un élément de la pile (dépiler) :

- Il n'est possible de supprimer qu'une seule valeur dans une pile.
- Cette valeur n'est que la dernière ajoutée dans la pile, c-à-d le sommet de la pile.
- Ainsi, après cette suppression, le dernier élément inséré devient le sommet de la pile.

Code en C

```
Pile depiler(Pile p)
  {
   Pile pPile;
   if(p==NULL)
    return NULL;
   else
    {
     pPile=p->precedent;
     free(p);// ou p=NULL;
     return pPile;
    }
  }
```

Retourner l'élément supprimer de la pile :

- Puisque une pile permet la mise en attente d'informations dans le but de les récupérer après, la fonction suivante permet de retourner l'élément (ou le sommet) retiré de la pile.

Code en C

```
element valeurSommet(Pile *p)
  {
    element vdep;
    Pile pPile;
    if(*p==NULL)
     return -1;
    else
     {
      vdep=(*p)->valeur;
      pPile=(*p)->precedent;
      free(*p);
      *p=pPile;
      return vdep;
     }
  }
```

Code en C

```
Pile valeurSommet2(Pile p,element *vdep)
  {
    Pile pPile;
    if(p==NULL)
     return NULL;
    else
     {
      *vdep=p->valeur;
      pPile=p->precedent;
      free(p);// ou p=NULL;
      return pPile;
     }
  }
```

récupérer le sommet de la pile :

- Puisque chaque pile est définie par son sommet, la fonction suivante permet de retourner la valeur du sommet d'une pile.

Code en C

```
element sommetPile(Pile p)
  {
   return p->valeur;
  }
```

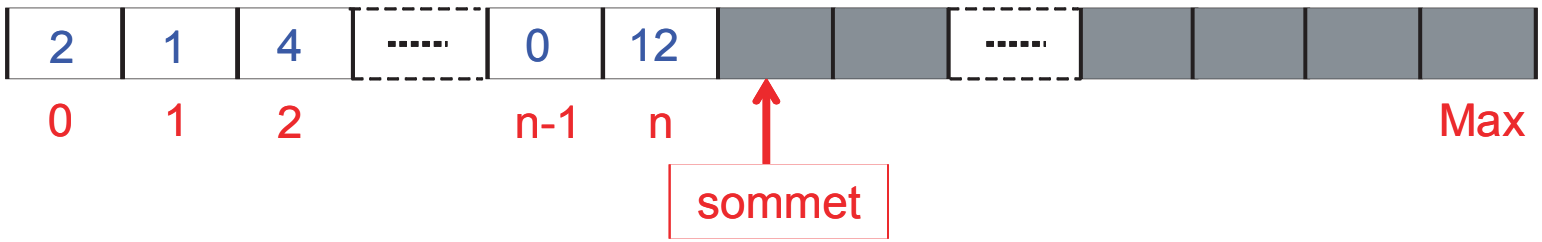
Vider la pile :

- La fonction suivante permet de vider le contenue d'une pile.

Code en C

```
Pile viderPile(Pile p)
  {
   while(p!=NULL)
     p=depiler(p);
   return p;
  }
```

- Comme le cas des listes, une pile peut être aussi implantée à l'aide d'un tableau.
- Les valeurs de la pile seront donc stockés dans les cases du tableau.
- Comme les piles gèrent des ensembles dynamiques, une taille maximum pour le tableau doit être donc définie lors des déclarations.
- C-à-d, on ne peut pas modifier la taille maximum du tableau lors de l'exécution d'un programme.
- Et si on désire modifier la taille maximum du tableau, il faut modifier aussi le programme.
- Par consequent, il faut définir un tableau dont la taille sera suffisante pour accueillir la plus grande pile pouvant être utilisée.



Déclaration d'une pile :

- En langage C, pour implanter une pile à l'aide d'un tableau, on utilise une structure constituée de deux champs,
  - un champs contenant un tableau, de taille suffisante pour accueillir la plus grande pile à traiter.
  - un autre champs contenant un entier représentant le sommet de la pile. En fait, cet entier contient l'indice+1 du sommet de la pile.

Code en C

```
typedef int element;
struct pileTab
 {
   element valeur[Max];
   int sommet;
 };
typedef struct pileTab pileTab;
```

Création et initialisation d'une pile :

- Pour créer une pile, on doit l'initialiser à un tableau de taille nulle.
- L'initialisation d'une pile, ou la création d'une pile vide, est généralement faite par la fonction suivante :

```
void pileVide(pileTab *p)
  {
   p->sommet=0;
  }
```

- Pour tester si une pile est vide, on utilise la fonction suivante :

```
 int testerPileVide(pileTab p)
 {
   return p.sommet==0;
 }
```

Ajouter un élément dans la pile (empiler) :

- Comme le cas d'implantation avec les pointeurs, il n'est possible d'insérer qu'une seule valeur en tête (ou sommet) de la pile.
- La valeur, une fois insérée, devient le sommet de la pile.

Code en C

```
pileTab empiler(element val, pileTab p)
  {
   p.valeur[p.sommet]=val;
   (p.sommet)++;
   return p;
  }
```

Supprimer un élément de la pile (dépiler) :

- De même que l'implantation avec les pointeurs, Il n'est possible de supprimer qu'une seule valeur dans une pile.
- Cette valeur n'est que la dernière ajoutée dans la pile, c-à-d le sommet de la pile.
- Ainsi, après cette suppression, le dernier élément inséré devient le sommet de la pile.

Code en C

```
pileTab depiler(pileTab p)
  {
   p.sommet=p.sommet-1;
   return p;
  }
```

Retourner l'élément supprimer de la pile :

- La fonction suivante permet de retourner le dernier élément (ou le sommet) retiré de la pile.

Code en C

```
int valeurSommet(pileTab *p)
  {
    int vdep;
    vdep=p->valeur[p->sommet-1];
    p->sommet=p->sommet-1;
    return vdep;
  }
```

récupérer le sommet de la pile :

- La fonction suivante permet de renvoyer la valeur du sommet d'une pile.

Code en C

```
element sommetPile(pileTab p)
  {
   return p.valeur[p.sommet-1];
  }
```

Vider la pile :

- La fonction suivante permet de vider le contenue d'une pile.

Code en C

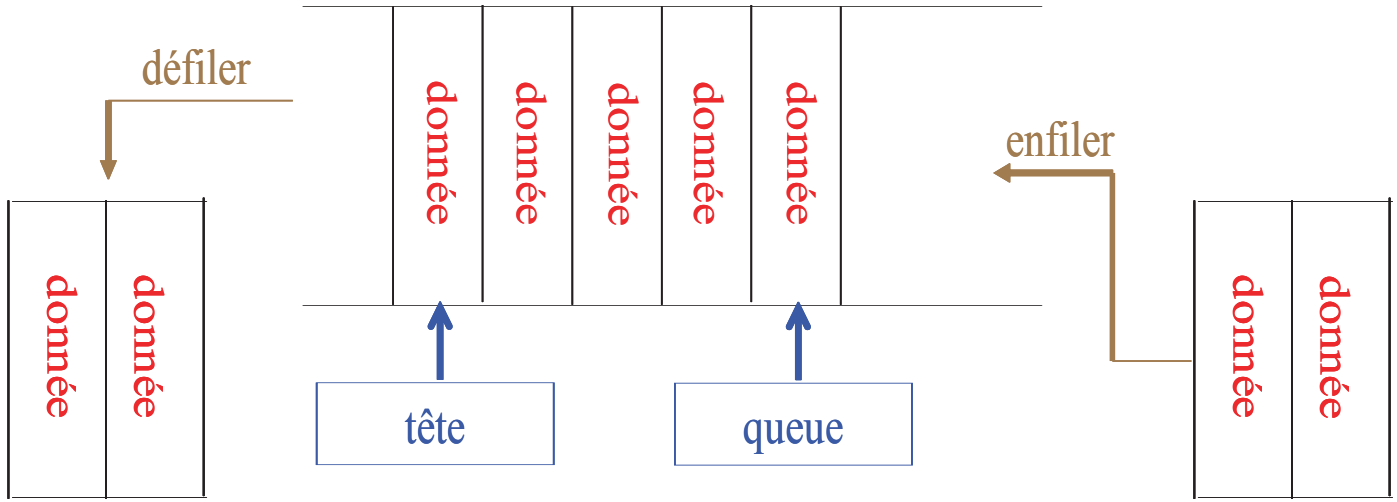
```
pileTab viderPile(pileTab p)
  {
   while(p.sommet!=0)
     p=depiler(p);
   return p;
  }
```

3. Files : 3.1 présentation générale

- Une file (connue aussi sous le nom de ***file d'attente***) est une structure de donnée identique à une pile sauf,
- les insertions se font à une extrémité appelée ***queue*** (fin de la file), les suppressions à l'autre appelée ***tête*** (debut de la file).
- Notons que dans une file, on a l'accès qu'à un seul élément, la ***tête*** de la file.
- Le dernier élément ajouté sera le dernier élément accessible.
- Lorsque on retire un élément de la file, on doit retirer toujours la tête, et cet élément étant le premier qui a été inséré dans la file.
- C-à-d, *le premier élément ajouté dans la file est le premier élément à en être supprimé*. Cette structure est connue sous le nom de ***FIFO*** (***First In, First Out***).
- Pour résumer, une file permet la mis en attente d'informations ou de données dans le but de les récupérer plus tard, dans l'ordre d'arrivée.
- La notion de file est également une structure utilisée en informatique :
  - la mise en attente de programme pour exécution,
  - l'accès à une périphérique, par exemple une imprimante.
  - etc.
- Généralement, il y a trois façons d'implanter une file en langage C :
  - soit en utilise un pointeur indiquant la queue de la file,
  - soit on utilise un tableau.
  - ou on peut utiliser des listes chaînées.

Opérations sur les files :

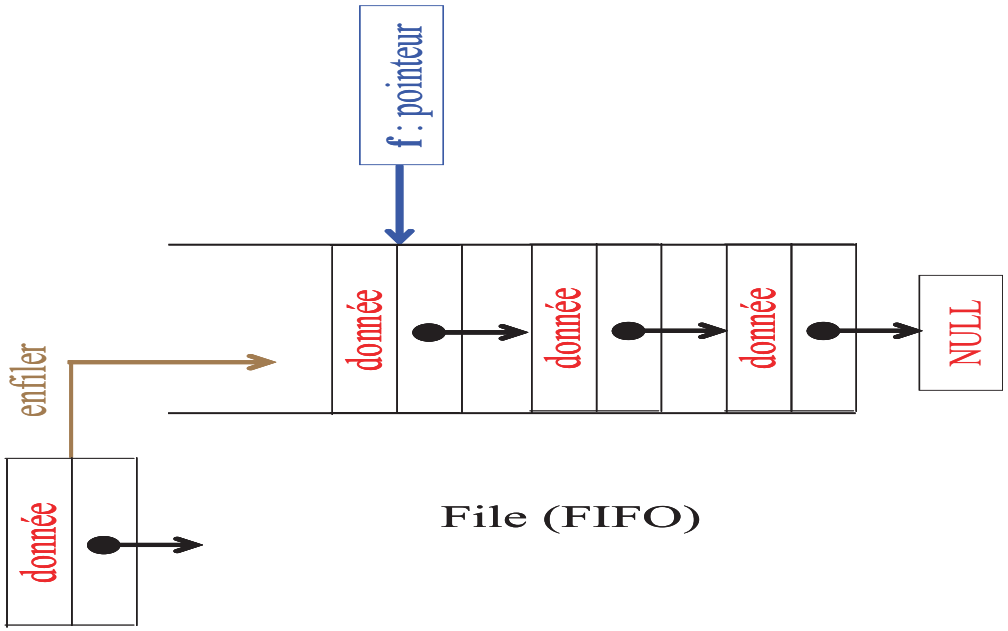
- Comme le cas des piles, dans une file, on ne peut autoriser que 5 opérations :
  1. *tester* si la file est vide,
  2. *enfiler* un élément, c-à-d l'ajouter à la queue de la file,
  3. *défiler* un élément, c-à-d le supprimer de la tête de la file,
  4. *récupérer* le premier élément de la file, c-à-d acceder à la tête de la file,
  5. *vider* le contenue de la file.



File (FIFO)

Déclaration d'une file :

- Comme le cas des piles t les listes, le langage C permet de réaliser des files à l'aide des structures.
- En effet, chaque cellule de la file est un objet constitué :
  - d'un champ de données,
  - d'un pointeur vers la cellule suivante.
- La déclaration correspondante est la suivante, où l'on suppose ici que les valeurs stockées dans la file sont de type entier.



File (FIFO)

Code en C

```
typedef int element;
struct celluleFile
 {
    element valeur;
    struct celluleFile * suivant;
 };
typedef struct celluleFile celluleFile, * File;
```

Création et initialisation d'une file :

- Pour créer une file, on doit l'initialiser à NULL, cela va nous permettre d'allouer la première cellule.
- L'initialisation d'une file, ou la création d'une file vide, est faite par la fonction suivante :

```
File fileVide( )
  {
   return NULL;
  }
```

- Pour tester si une file est vide, on utilise la fonction suivante :

```
 int testerFileVide(File f)
 {
  return f==NULL;
 }
```

Ajouter un élément dans la file (enfiler) :

- Il n'est possible d'insérer qu'une seule valeur en queue de la file.
- La valeur, une fois insérée, devient la queue de la file.

Code en C

```
File enfiler(element val, File f)
  {
    File fn;
    fn=malloc(sizeof(celluleFile));
    fn->valeur=val;
    if(f==NULL)
     {
        fn->suivant=NULL;
        return fn;
     }
    else
     {
        fn->suivant=f;
        f=fn;
        return f;
     }
  }
```

Supprimer un élément de la file (défiler) :

- Il n'est possible de supprimer qu'une seule valeur dans une file.
- Cette valeur n'est que la première entrée dans la file, c-à-d la tête de la file.

Code en C

```
File defiler(File f)
  {
    File pfile1, pfile2;
    if(f==NULL)
     return NULL;
    if(f->suivant==NULL)
     {
       free(f);
       return NULL;
     }
    pfile1=f;
    while(pfile1->suivant->suivant!=NULL)
      pfile1=pfile1->suivant;
    pfile2=pfile1->suivant;
    pfile1->suivant=NULL;
    free(pfile2);
    return f;
  }
```

Retourner l'élément supprimer de la pile :

- Puisque une pile permet la mise en attente d'informations dans le but de les récupérer après, la fonction suivante permet de retourner l'élément (ou le sommet) retiré de la pile.

Code en C

```
int valeurTete(File *f)
  {
    int vdef;
    File pfile1, pfile2;
    if((*f)->suivant==NULL)
     {
       vdef=(*f)->valeur;
       free(*f);
       return vdef;
     }
    pfile1=*f;
    while(pfile1->suivant->suivant!=NULL)
      pfile1=pfile1->suivant;
    pfile2=pfile1->suivant;
    pfile1->suivant=NULL;
    vdef=pfile2->valeur;
    free(pfile2);
    return vdef;
  }
```

Code en C

```
File valeurTete(File f, int *vdef)
  {
    File pfile1, pfile2;
    if(f->suivant==NULL)
     {
       *vdef=f->valeur;
       free(f);
       return NULL;
     }
    pfile1=f;
    while(pfile1->suivant->suivant!=NULL)
      pfile1=pfile1->suivant;
    pfile2=pfile1->suivant;
    *vdef=pfile2->valeur;
    pfile1->suivant=NULL;
    free(pfile2);
    return f;
  }
```

récupérer la queue de la file :

- Puisque chaque file peut être définie par sa queue, la fonction suivante permet de retourner la valeur de la queue d'une file.

Code en C

```
element queueFile(File f)
  {
   return f->valeur;
  }
```

récupérer la tête de la file :

- Puisque chaque file peut être définie aussi par sa tête, la fonction suivante permet de retourner la valeur de la tête d'une file.

Code en C

```
element teteFile(File f)
  {
   while(f->suivant!=NULL)
     f=f->suivant;
   return f->valeur;
  }
```

Vider la file :

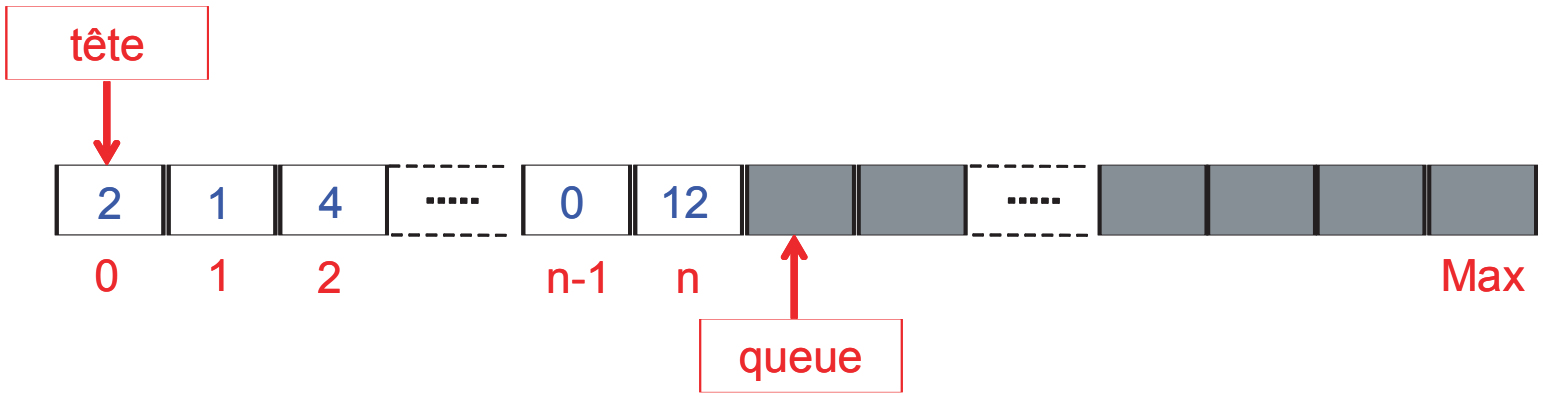
- La fonction suivante permet de vider le contenue d'une file.

Code en C

```
File viderFile(File f)
  {
   while(f!=NULL)
     f=defiler(f);
   return f;
  }
```

3. Files : 3.3 Implantation par tableau

- Comme le cas des piles et des listes, une file peut être aussi implantée à l'aide d'un tableau.
- Les valeurs de la file seront donc stockés dans les cases du tableau.
- Comme les files gèrent des ensembles dynamiques, une taille maximum pour le tableau doit être donc définie lors des déclarations.
- C-à-d, on ne peut pas modifier la taille maximum du tableau lors de l'exécution d'un programme.
- Et si on désire modifier la taille maximum du tableau, il faut modifier aussi le programme.
- Par consequent, il faut définir un tableau dont la taille sera suffisante pour accueillir la plus grande pile pouvant être utilisée.



Déclaration d'une file :

- En langage C, pour implanter une file à l'aide d'un tableau, on utilise une structure constituée de deux champs,
  - un champs contenant un tableau, de taille suffisante pour accueillir la plus grande file à traiter.
  - un autre champs contenant un entier représentant la queue de la file. En fait, cet entier contient l'indice+1 de la queue de la file.

Code en C

```
typedef int element;
struct fileTab
 {
   element valeur[Max];
   int queue;
 };
typedef struct fileTab fileTab;
```

Création et initialisation d'une file :

- Pour créer une file, on doit l'initialiser à un tableau de taille nulle.
- L'initialisation d'une file, ou la création d'une file vide, est généralement faite par la fonction suivante :

```
void fileVide(fileTab *f)
  {
   f->queue=0;
  }
```

- Pour tester si une file est vide, on utilise la fonction suivante :

```
 int testerFileVide(fileTab f)
 {
  return f.queue==0;
 }
```

Ajouter un élément dans la file (enfiler) :

- Comme l'implantation avec les pointeurs, il n'est possible d'insérer qu'une seule valeur en queue de la file.
- La valeur, une fois insérée, devient la queue de la file.

Code en C

```
fileTab enfiler(element val, fileTab f)
  {
   f.valeur[f.queue]=val;
   (p.queue)++;
   return f;
  }
```

Supprimer un élément de la file (défiler) :

- Comme l'implantation avec les pointeurs, il n'est possible de supprimer qu'une seule valeur dans une file.
- Cette valeur n'est que la première entrée dans la file, c-à-d la tête de la file.

Code en C

```
fileTab defiler(fileTab f)
  {
   int i;
   for(i=0;i<=f.queue-2;i++)
    f.valeur[i]=f.valeur[i+1];
   f.queue=f.queue-1;
   return f;
  }
```

Retourner l'élément supprimer de la file :

- La fonction suivante permet de retourner le dernier élément (ou la tête) retiré de la file.

Code en C

```
int queueFile(fileTab *f)
  {
   int vdef,i;
   vdef=f->valeur[0];
   for(i=0;i<=f.queue-2;i++)
     f.valeur[i]=f.valeur[i+1];
   f.queue=f.queue-1;
   return vdef;
  }
```

récupérer la queue de la file :

- La fonction suivante permet de renvoyer la valeur de la queue d'une file.

Code en C

```
element queueFile(fileTab f)
  {
    return f.valeur[f.queue-1];
  }
```

récupérer la tête de la file :

- Puisque chaque file peut être définie aussi par sa tête, la fonction suivante permet de retourner la valeur de la tête d'une file.

Code en C

```
element teteFile(fileTab f)
  {
   return f.valeur[0];
  }
```

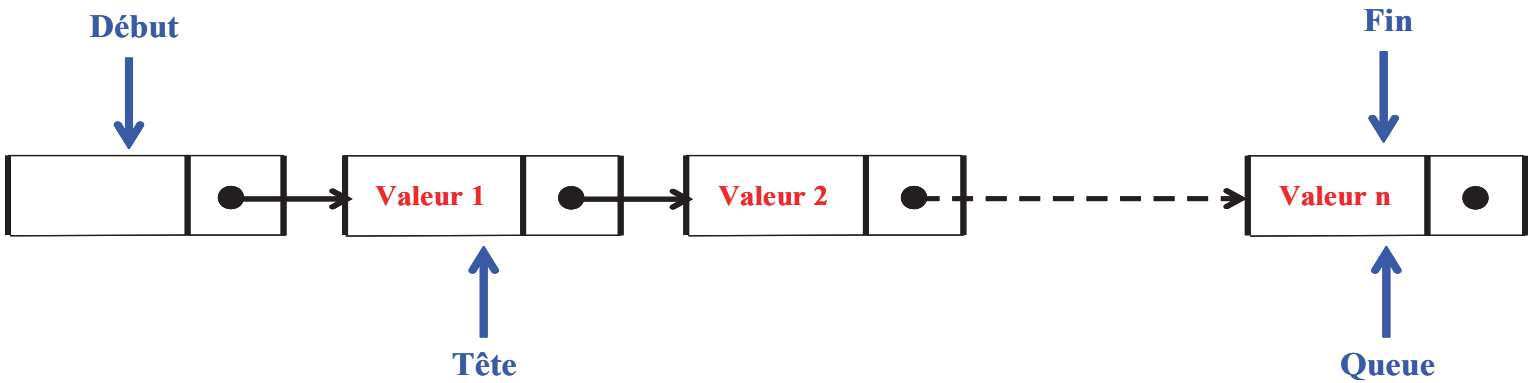
Vider la file :

- La fonction suivante permet de vider le contenue d'une file.

Code en C

```
fileTab viderFile(fileTab f)
  {
   while(f.queue!=0)
    f=defiler(f);
   return f;
  }
```

- Une autre façon de gérer des files consiste à utiliser des listes chaînées.
- En effet, comme une file est un cas particulier de liste chaînée, chaque file sera définie par deux cellules : une cellule qui pointe vers le début (la tête) de la file et une autre qui pointe sur sa fin (queue).



File représenté par une liste chaînée

Déclaration d'une file :

- En langage C, pour implanter une file à l'aide d'une liste chaînée, on utilise une structure constituée de deux champs contenant des pointeurs de type liste,
  - le premier pointe vers la cellule représentant la tête de la file.
  - le deuxième pointe sur la cellule représentant la queue de la file.

Code en C

```
struct fileListe
{
  liste deb;
  liste fin;
};
typedef struct fileListe fileListe;
```

Création et initialisation d'une file :

- Pour créer une file, la cellule qui pointe sur la queue de la file doit pointer sur la même adresse du pointeur qui pointe vers la cellule début.
- À noter que cette adresse ne contient aucun valeur de type `element`.
- L'initialisation d'une file, ou la création d'une file vide à l'aide d'une liste est faite par la fonction suivante :

```
fileListe fileVide()
  {
   fileListe f;
   f.deb=(liste)malloc (sizeof (cellule));
   f.fin=f.deb;
   return f;
  }
```

- Pour tester si une file est vide, on utilise la fonction suivante :

```
 int testerFileVide(fileListe f)
 {
  return f.deb==f.fin;
 }
```

Exercice :

Écrire les fonctions suivantes en utilisant une implantation des files par les listes chaînées :

- *enfiler* un élément,
- *défiler* un élément,
- *récupérer* la tête de la file,
- *vider* le contenue de la file.