Chapitre 5: Les Files

1. Définition

On appelle file d'attente (ou tout simplement file) un ensemble formé d'un nombre variable, éventuellement nul de données, sur lequel les opérations suivantes peuvent être effectuées :

creer_file : permet de créer une file vide (création).
file_vide : permet de tester la vacuité d'une file, ou si la file est vide ou non (consultation).
enfiler : permet d'ajouter une donnée de type T à la file (modification).
defiler : permet d'obtenir une nouvelle file (modification).
premier : permet d'obtenir l'élément le plus ancien dans la file (consultation).

Opérations illégales : il y a des opérations définies sur la SD file d'attente exigent des pré conditions :

défiler exige que la file soit non vide.
premier exige que la file soit non vide.

2. Propriétés

Dans ce paragraphe on va citer (énumérer) les propriétés qui caractérisent la sémantique des opérations applicables sur la SD file d'attente.

F1 : creer_file permet de créer une file vide.
F2 : si un élément entré (enfiler) dans la file résultante est non vide.
F3 : un élément qui entre (grâce à enfiler) dans la file d'attente devient immédiatement le premier : si la file vide sinon (file non vide) le premier reste inchangé.
F4 : une entrée et une sortie successive sur une file vide la laissent vide.
F5 : Une entrée et une sortie successive sur une file non vide peuvent être effectuées dans n'importe quel ordre.

Illustration :

File non vide : 5 6 1

Cas 1 : 5 6 1

enfiler 8 →5 6 1 8
defiler →6 1 8

Cas 2 : 5 6 1

defiler → 6 1
enfiler 8→6 1 8

Remarques :

- La structure de File obéit à la loi FIFO : First In, First Out.
- La SD file d'attente est structurée à deux points d'entrée. Par contre la SD pile est une structure à un seul point d'entrée.

3. Représentation physique

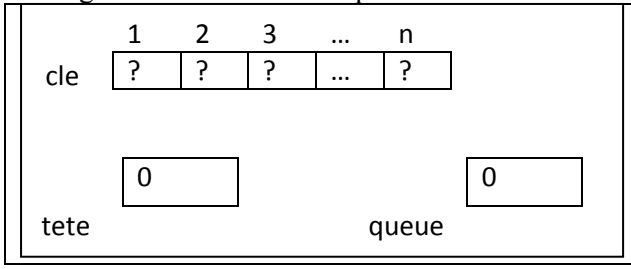
On distingue deux types de représentations physiques :

-représentation contiguë
-représentation chaînée

3.1 Représentation contiguë

TDA FILE concrétisé par une représentation contiguë

Il s'agit d'un tableau à deux points d'entrée : deux indices tête et queue.



Constante N 100
Type
File= struct
Cle :tableau de N entier
Tete :entier
Queue : entier
FinStruct

➔**Opération ou services exportés :**

Convention :

- ✓ On enfile après queue.
- ✓ On defile : après tete

```
Procedure creer_file (var  F:File)
début
        F.Tete   ←0
        F.Queue ←0
Fin Proc

Fonction file_vide (F:File) : boolean
début
        retourner  ( F.Tete = F.Queue)
Fin Fn
```

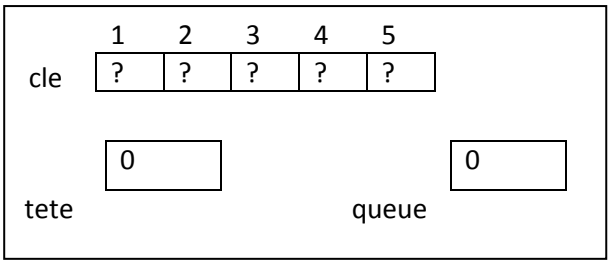
```
Procedure enfiler (x: entire ; var F:File)
début
        assurer (F.Queue <N)
        F.Queue ←  F.Queue+1
        F.Cle [F.Queue]← x
Fin Proc

Procedure defiler (var  File F)
début
        assurer (Non file_vide (F))
        F. Tete ←  F.Tete +1
Fin Proc

Fonction premier (F:File) : entire
début
        assurer (Non file_vide (F))
        retourner  ( F.Cle [ F.Tete+1] )
Fin Fn
```

➔Un problème posé par la représentation contigüe de la SD file :

Situation initiale : file vide

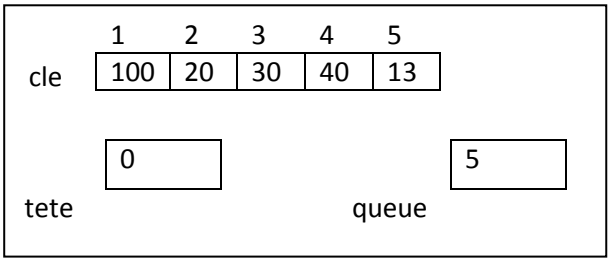


Question : En partant de cette file vide (de taile n=5) , exécuter la séquence suivante :

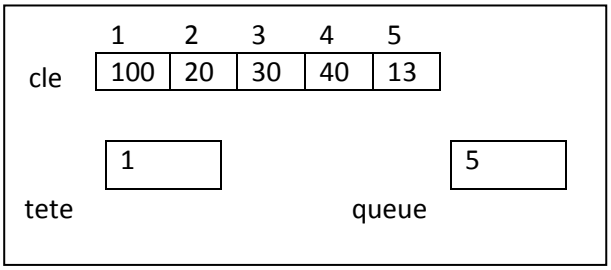
a) enfiler les éléments : 100 puis 20 puis 30 puis 40 puis 13.
b) defiler
c) enfiler 120 ;

Résultat :

a) Situation après les 5 enfilements



b) Situation après le défilement

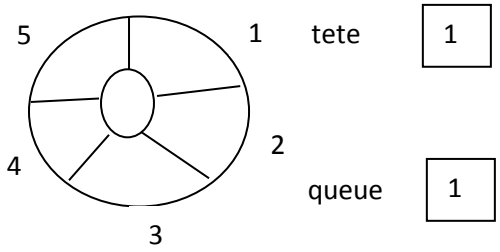


c) L'état en position 1 est disponible, mais on ne peut pas enfiler de nouveau !!!

Prob : On ne peut enfiler 120 après queue (en effet l'élément de position queue+1 (5+1=6) n'appartient pas au tableau clé. Et pourtant la file n'est pas pleine ?? Car le tableau est perçu d'une façon linéaire, il est parcourue de gauche à droite. Au bout de n enfilements, on ne peut plus ajouter des nouveaux éléments, **même si on fait des défilements**. Sachant que la valeur n est la taille du tableau cle.

- ⇨ **Remède :** un tableau circulaire c'est-à-dire : tete et queue **modulo n ;** avec n est la taille du tableau.
- ⇨ Il s'agit d'une perception logique et non physique. On va appliquer la séquence des actions a, b et c vue précédemment sur un tableau sur perçu d'une façon circulaire.

Situation initiale : file vide



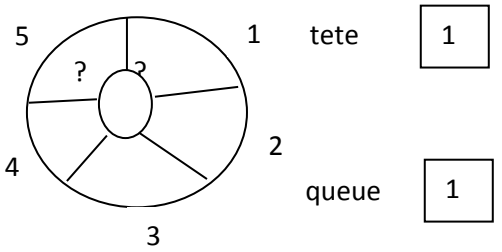
Convention :

- ✓ On enfile après queue.
- ✓ On defile : après tete

enfilement 100 → queue=2
enfilement 20 → queue=3
enfilement 30 → queue=4
enfilement 40 → queue=5
enfilement 13 → queue+1 =6 ; or 6>n donc ➔ queue= 6 mod 5 = 1

defiler → tete=2
enfiler 120 → queue+1=1+1=2➔cette position est disponible. L'élément dans cette position est déjà traitée en (b)

Constatation :



tete=queue →file vide ou file pleine ??

➔d'où un deuxième problème

➔**Idée :** Au lieu de réserver un tableau (ici cle) de n éléments, on prévoit un tableau de taille n+1, en respectant la propriété suivante :

- ✓ On utilise au plus n éléments : lorsque la file contient n éléments, la file est logiquement pleine mais pas physiquement.
- ✓ La proposition tete=queue caractérise une file vide.

Implémentation :

```
Constante N 100
Type
File= struct
        cle : tableau de N entier
        tete : entier
        queue : entier
FinStruct


Procedure creer_file ( var F: File)
debut
        F.tete    ←1
        F. queue ←1
finProc
```

RQ : n'importe quel indice compris entre 1 et n pourra faire l'affaire.

```
Fonction file_vide (F: File) : Boolean
debut
        file_vide ← (F.tete=F.queue)
FinFn


Fonction premier (F: File) : entier
Var
    i: entier
debut
        assure ( Non file_vide(F)  )
        i←F.tete+1
        si (i>n) alors
                i←1
        finSi
        premier← (F.cle[i])
finFn
```

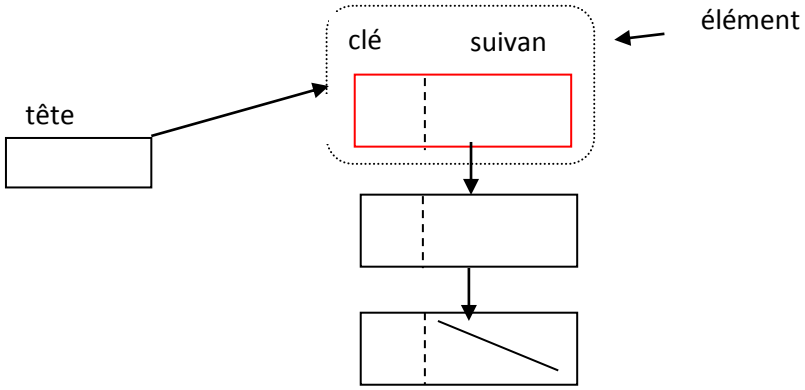
```
Procedure enfiler (info: entire ; var F : File)
debut
    F. queue ← F. queue+1

    Si (F.queue > n) alors
        F. queue ← 1
    finsi
    assure (f.tete != f.queue)
    F.cle [F.queue]←info
finProc


Procedure defiler  (var F : File)
debut
   assure ( Non file_vide(F))
    F.tete← F.tete+1

    Si (F.tete > n) alors
        F.tete ←1
     finSi
FinProc
```

3.2 Représentation chaînée



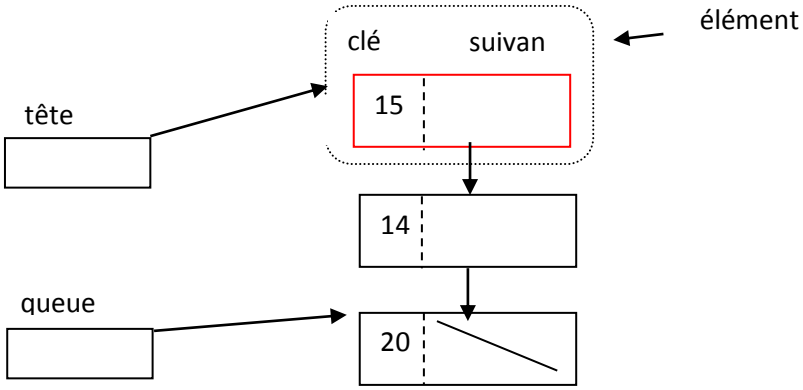
premier : coût une indirection en partant du pointeur tête

defiler : coût une indirection en partant du pointeur tête

enfiler : coût il faut parcourir toute la file en partant du pointeur tête.

Ceci nécessite plusieurs indirections. Ainsi, **il ne faut pas retenir la solution proposée**
Remède : on a besoin d'une représentation physique à deux points d'entrées : tête et queue.

Le pointeur **tête** favorise l'implémentation efficace des opérations : premier et defiler.
Le pointeur **queue** favorise l'implémentation efficace de l'opération enfiler.



Remarque :
La SD file d'attente est structurée à deux points d'entrée. Par contre la SD pile est une structure à un seul point d'entrée.

4Matérialisation de la SD File comme type de donnée abstrait

```
/*représentation chainée*/
Types
        Cellule = Struct
          cle   : entier
          Suiv : ^Cellule
        FinStruct

        File = Struct
          tete    : ^Cellule
          queue : ^Cellule
        FinStruct
```

```
Procedure creer_File (var F :File )
Début
        F.tete←Nil
        F.queue←Nil
Fin Proc

Fonction File_vide (F :File): boolean
Debut
        File_vide←( F.queue=Nil)
fin Fn

Procédure enfiler (x : Entier ; var F : File)
Var
        P : ^Cellule
Début
        Allouer (P)
        P^.cle← x
        P^.Suiv←Nil

        Si File_vide(F) alors
                F.tête← P
                F.queue← P
        sinon
                F.queue^.Suiv←P
                F.queue← P
        FinSi
Fin Proc

Procédure défiler (Var F : File)
Var
        Q : ^Cellule
Début
        Assure (Non File_vide(F))
        Q←F.Tête
        F.Tête←F.Tête^.Suiv
        Libérer(Q)
        Si F.Tête = NIL alors
           F.queue ← NIL
        FinSi
Fin Proc

Fonction premier (F : File) : entier
Début
        Assure (Non File_vide(F))
        premier←F.tête^.cle
Fin Fn
```