Chapitre 4 : Piles et Files

Les piles et files ne sont pas de nouveaux types de données mais plutôt une manière de gérer un ensemble de données. Elles sont très souvent utiles et servent, entre autres, à mémoriser des évènements en attente de traitement.

Il n'y a pas de structures spécifiques prévues dans les langages de programmation pour les piles ou files. Il faut donc les créer de toute pièce sachant que la représentation en mémoire de ces structures de données peut être, selon le besoin, statique (utilisation des tableaux) ou dynamique (utilisation des listes).

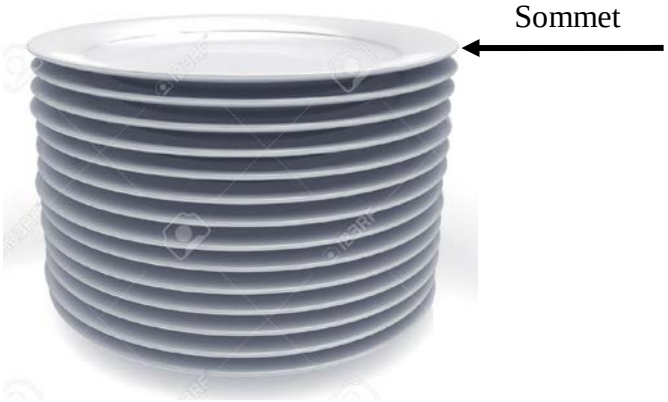
I- Pile

Quand on vous dit pile penser directement à une pile d'assiettes qu'il faut manipuler avec attention pour éviter les dégâts.

Une pile est un ensemble de valeurs ne permettant des insertions ou des suppressions qu'a une seule extrémité, appelée ***sommet de la pile***.

Empiler un objet sur une pile P consiste à insérer cet objet au sommet de P (dans la pile d'assiettes une nouvelle assiette ne peut être ajoutée qu'au dessus de celle qui se trouve au sommet) ;

Dépiler un objet de P consiste à supprimer de P l'objet placé au sommet (dans la pile d'assiettes seule peut être retirée celle qui se trouve au sommet). L'objet dépilé est retourné comme résultat du traitement.



Une propriété remarquable des piles est qu'un objet ne peut être dépilé qu'après avoir dépilé tous les objets qui sont placés "au dessus" de lui, ce qui fait que les objets quittent la pile dans l'ordre inverse de leur ordre d'arrivée. Pour cette raison, une pile est aussi appelée structure LIFO (Last In, First Out) ou (dernier arrivé, premier sorti)

En informatique une pile sert essentiellement à stocker des données qui ne peuvent pas être traitées immédiatement, car le programme a une tâche plus urgente ou préalable à accomplir auparavant. En particulier les appels et retours de fonctions sont gérés grâce à une pile appelée pile d'exécution.

En termes de programmation, une pile est un enregistrement avec :
- Une structure de données pour enregistrées les valeurs (elle peut être statique ou dynamique)
- Une variable sommet qui indique le sommet de la pile.

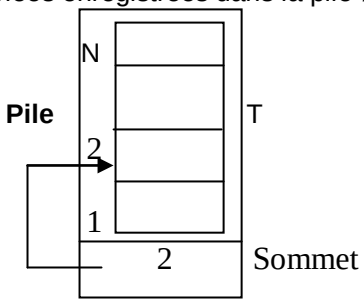
La manipulation d'une pile revient à l'appel de fonctions et procédures dites de bases définies une seule fois et utilisées autant de fois qu'il est nécessaire.
Ces sous-algorithmes sont :
- Init_Pile : permet d'initialiser une pile à vide lors de sa création ;
- Pile_vide : pour vérifier si une pile est vide ou non et savoir alors s'il reste des valeurs à traiter ou non ;
- Pile_pleine : pour vérifier s'il est possible de rajouter ou non un nouveau élément (utilisée dans le seul cas des piles statiques) ;
- Empiler : permet d'ajouter une nouvelle valeur (envoyé en paramètre par l'appelant) à la pile (au dessus du sommet et dans le cas d'une pile non pleine) ;
- Depiler : permet de supprimer une valeur (se trouvant au sommet de la pile) et de la renvoyer en paramètre. Cette opération n'est possible que si la file n'est pas vide.

I-1- Pile statique

Une pile statique est un enregistrement à 2 cases : un tableau à hauteur maximale prévisible et un indice entier qui pointe la dernière valeur ajoutée à la pile (sommet).

I-1-*1- Déclaration*

```
Constante
    N = …. ; /* taille du tableau*/
Type
    Tab = Tableau de N Type_C ; /* Type_C est le type des données enregistrées dans la pile*/
    Pile = Enregistrement
            T : Tab ;
            Sommet : Entier
    Fin ;
```



Note : Les données enregistrées dans la pile peuvent être des entiers, des réels, des caractères, des chaînes de caractères, des booléens, des tableaux, des pointeurs de listes ou encore des piles ou files.

I-1-*2- Initialisation d'une pile*

```
Procedure Init_Pile (Var P : Pile) ;
Debut
     P. Sommet ← 0
Fin ;
```

I-1-*3- Vérification de pile vide*

```
Fonction Pile_vide (P : Pile) : Booleen ;
Debut
   Si P. Sommet = 0 Alors
          Pile_vide ← vrai
   Sinon
          Pile_vide ←faux
   FinSi ;
Fin ;
```

Une autre façon plus compacte d'écrire cette fonction est la suivante :

```
Fonction Pile_vide (P : Pile) : Booleen ;
Debut
    Pile_vide ← P. Sommet = 0;
Fin ;
```

Le nom de la fonction recevra le résultat de la comparaison (vrai ou faux) entre le sommet et le zéro.

I-1-*4- Vérification de pile pleine*

```
Fonction Pile_pleine (P : Pile) : Booleen ;
Debut
   Si P. Sommet = N Alors
        Pile_pleine ← vrai
   Sinon
        Pile_pleine ←faux
   FinSi ;
Fin ;
```

Une autre façon plus compacte d'écrire cette fonction est la suivante :

```
Fonction Pile_pleine (P : Pile) : Booleen ;
Debut
    Pile_pleine ← P. Sommet = N;
Fin ;
```

I-1-*5- Ajout d'une nouvelle valeur à une pile*

```
Procedure Empiler (Var P : Pile ; X : Type_C) ;
Debut
   Si Pile_pleine (P) Alors
        Ecrire('Impossible la pile est pleine')
   Sinon
     Debut
        P. Sommet ←P. Sommet + 1 ;
        P.T (P. Sommet) ← X ;
     Fin
   FinSi ;
Fin ;
```

I-1-*6- Suppression d'une valeur de la pile*

```
Procedure Depiler (Var P : Pile, X : Type_C) ;
Debut
   Si Pile_vide (P) Alors
        Ecrire('Impossible la pile est vide')
   Sinon
     Debut
        X ← P.T (P. Sommet) ;
        P. Sommet ←P. Sommet - 1 ;
      Fin
   FinSi ;
Fin ;
```

I- 2- Pile dynamique

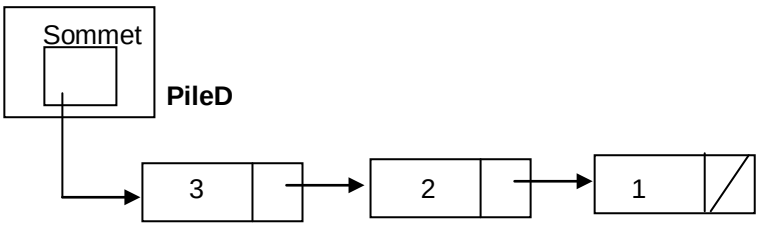
Une pile dynamique est une liste à la quel on attache un pointeur sommet. C'est un enregistrement à une seule case : pointeur qui pointe la dernière valeur traitée dans la liste (sommet).

De même que pour les piles statiques nous présentons la déclaration et les sous algorithmes de bases détaillés pour le type statique. Et dans un souci d'uniformisation nous utilisons les mêmes noms mais en ajoutant un D (pour rappeler Dynamique) à la fin de chaque nom pour faire la différence.

Note : Il n'y a pas de pile pleine pour les piles dynamique. La seule possibilité de ne pas pouvoir ajouter un élément c'est d'avoir une mémoire pleine, cas que l'on ne prend pas en considération ici.

I-2-1- Déclaration d'une pile dynamique

```
Type
    Liste = ^Elem
    Elem = Enregistrement
          Info : Type_C ;
          Suiv : ^Elem
    Fin ;
    PileD = Enregistrement
          Sommet : Liste
    Fin ;
```



Note : Les données enregistrées dans la pile peuvent être des entiers, des réels, des caractères, des chaînes de caractères, des booléens, des tableaux, des pointeurs de listes ou encore des piles ou files.

I-2-*2- Initialisation d'une pile dynamique*

```
Procedure Init_PileD (P : PileD) ;
Debut
     P. Sommet ← Nil
Fin ;
```

I-2-*3- Vérification de pile vide dynamique*

```
Fonction Pile_videD (P : PileD) : Booleen ;
Debut
   Si P. Sommet = Nil Alors
         Pile_videD ← vrai
   Sinon
         Pile_videD ←faux
   FinSi ;
Fin ;
```

Une autre façon plus compacte d'écrire cette fonction est la suivante :

```
Fonction Pile_videD (P : PileD) : Booleen ;
Debut
    Pile_videD ← P. Sommet = Nil;
Fin ;
```

Le nom de la fonction recevra le résultat de la comparaison (vrai ou faux) entre le sommet et le zéro.

I-2-*4- Ajout d'une nouvelle valeur à une pile dynamique*

```
Procedure EmpilerD (Var P : PileD ; X : Type_C) ;
Variable
   Pt : Liste ;
Debut
        Allouer (Pt) ;
        Pt^.Info ← X ;
        Pt^. Suiv ← P. Sommet ;
        P. Sommet ← Pt ;
Fin ;
```

Note : L'ajout d'une valeur à une pile dynamique revient à une insertion en début de liste si l'on considère que le sommet est la tête de la liste.

I-2-*5- Suppression d'une valeur de la pile dynamique*

La suppression d'une valeur dans une pile dynamique revient à effectuer une suppression physique d'un nœud (le premier de liste) et à récupérer dans un paramètre, passé par variable, la donnée enregistrée.

```
Procedure DepilerD (Var P : Pile, X : Type_C) ;
Variable
   Pt : Liste ;
Debut
   Si Pile_videD (P) Alors
        Ecrire('Impossible la pile est vide')
   Sinon
     Debut
        Pt ← P. Sommet ;
        X ← P.Sommet^.Info ;
        P. Sommet ← P. Sommet^.Suiv ;
        Liberer (Pt)
      Fin
   FinSi ;
Fin ;
```

II- File

Quand on vous dit file penser directement à une file d'attente où chacun à son tour qu'il doit respecter.

Une file est un ensemble de valeurs qui a un début (Debut) et une fin (Queue).

Enfiler un objet sur une file F consiste à insérer cet objet à la fin de la file F (dans la file d'attente un nouvel arrivant se met à la queue c.-à-d., après la personne arrivée juste avant lui) ;

Défiler un objet de F consiste à supprimer de F l'objet placé en début de file (dans la file d'attente seule peut être servie la personne qui se trouve en début de file). L'objet défilé est retourné comme résultat du traitement.

En informatique une file sert essentiellement à stocker des données qui doivent être traitées selon leur ordre d'arrivée. L'exemple le plus connu est celui de l'impression de documents reçus par une imprimante qui imprime le premier document arrivé et termine par le dernier. Ce qui fait que les objets quittent la pile dans l'ordre de leur ordre d'arrivée. Pour cette raison, une file est aussi appelée structure FIFO (First In, First Out) ou (premier arrivé, premier sorti)

En termes de programmation, une file est un enregistrement avec :
- Une structure de données pour enregistrées les valeurs (elle peut être statique ou dynamique)
- Une variable Debut qui indique le premier élément de la file.
- Une variable Queue qui indique le dernier élément de la file.

Comme pour les piles, la manipulation d'une file revient à l'appel de fonctions et procédures dites de bases définies une seule fois et utilisées autant de fois qu'il est nécessaire.
Ces sous-algorithmes sont :
- Init_File : permet d'initialiser une file à vide lors de sa création ;
- File_vide : pour vérifier si une file est vide ou non et savoir alors s'il reste des valeurs à traiter ou non ;
- File_pleine : pour vérifier s'il est possible de rajouter ou non un nouveau élément (utilisée dans le seul cas des files statiques) ;
- Enfiler : permet d'ajouter une nouvelle valeur (envoyé en paramètre par l'appelant) à la file (après le dernier élément de la file qui se trouve au niveau de sa queue et dans le cas d'une file non pleine) ;
- Defiler : permet de supprimer une valeur (se trouvant au début de la file) et de la renvoyer en paramètre. Cette opération n'est possible que si la file n'est pas vide.

II-1- File statique

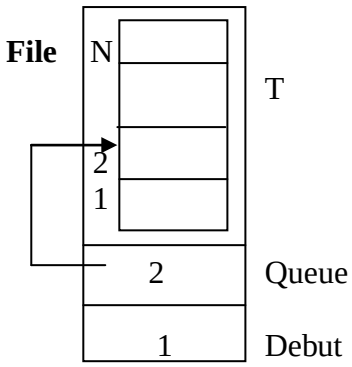
Une file statique est un enregistrement à 3 cases : un tableau à hauteur maximale prévisible, un indice entier qui pointe le premier élément insérer dans la file (Debut) et un deuxième indice entier qui pointe la dernière valeur ajoutée (Queue).

Exemples :
Sachant que le tableau est indicé de 1 à N et pour une gestion simpliste des files :

1- Une file a un seul élément → Debut = 1 et Queue = 1
2- Une file a 3 éléments → Debut = 1 et Queue = 3
3- une file qui vient d'être déclarée (et non encore utilisée) → Debut = 1 et Queue = 0
4- une file complètement vidée → Debut = Queue +1

II-1-*1- Déclaration*

```
Constante
    N = …. ; /* taille du tableau*/
Type
   Tab = Tableau de N Type_C ; /* Type_C est le type des données enregistrées dans la pile*/
   File = Enregistrement
         T : Tab ;
         Debut, Queue : Entier
   Fin ;
```



Note : Les données enregistrées dans la pile peuvent être des entiers, des réels, des caractères, des chaînes de caractères, des booléens, des tableaux, des pointeurs de listes ou encore des piles ou files.

! Dans certain langages de programmation le nom "File" désigne un type de données appelé fichier. Dans ce cas ne pas utiliser ce terme comme identifiant de la file.

II-1-*2- Initialisation d'une file*

```
Procedure Init_File (Var F : File) ;
Debut
     F. Debut ← 1 ;
     F. Queue ← 0
Fin ;
```

II-1-*3- Vérification de File vide*

```
Fonction File_vide (F : File) : Booleen ;
Debut
   Si F. Debut > F. Queue Alors
         File_vide ← vrai
   Sinon
         File_vide ←faux
   FinSi ;
Fin ;
```

Une autre façon plus compacte d'écrire cette fonction est la suivante :

```
Fonction File_vide (F : File) : Booleen ;
Debut
     File_vide ← F. Debut > F. Queue ;
Fin ;
```

Le nom de la fonction recevra le résultat de la comparaison (vrai ou faux) entre les indices début et Queue.

II-1-*4- Vérification de File pleine*

```
Fonction File_pleine (F : File) : Booleen ;
Debut
   Si (F. Queue = N) et (F.Debut < F. Queue) Alors
         File_pleine ← vrai
   Sinon
         File_pleine ← faux
   FinSi ;
Fin ;
```

Une autre façon plus compacte d'écrire cette fonction est la suivante :

```
Fonction File_pleine (F : File) : Booleen ;
Debut
     File_pleine ← (F. Queue = N) et (F. Debut < F. Queue) ;
Fin ;
```

Note : **Bien que non utilisée dans ce cours,** il existe une autre façon de considérer les piles (en circulaire). En effet, vu qu'en dépilant c'est le Debut qui se rapproche de la Queue et que les cases en dessous du Debut sont supposées devenues vides, il est possible de les réutiliser si la file est pleine d'en haut. Il suffirait que Queue est remise à 1 et s'incrémente jusqu'à comblement des cases vides en dessous de Debut.

II-1-*5- Ajout d'une nouvelle valeur à une File*

```
Procedure Enfiler (Var F : File ; X : Type_C) ;
Debut
   Si File_pleine (F) Alors
         Ecrire('Impossible la file est pleine')
   Sinon
     Debut
         F. Queue ← F. Queue + 1 ;
         F.T (F. Queue) ← X ;
     Fin
   FinSi ;
Fin ;
```

II-1-*6- Suppression d'une valeur de la File*

```
Procedure DeFiler (Var F : File, X : Type_C) ;
Debut
   Si File_vide (F) Alors
         Ecrire('Impossible la File est vide')
   Sinon
     Debut
         X ← F.T (F. Debut) ;
         F. Debut ← F. Debut + 1 ;
      Fin
   FinSi ;
Fin ;
```

II- 2- File dynamique

Une File dynamique est une liste à la quel on attache deux (2) pointeurs Debut et Queue. C'est un enregistrement à deux cases : pointeur qui pointe le premier élément de la liste (Debut) et un autre qui pointe la dernière valeur ajoutée dans la liste (Queue).

Exemples :
1- Une file vide → Debut = Queue = Nil
2- Une file a un seul élément → Debut = Queue ≠ Nil
3- Une file a plus d'un élément → Debut ≠ Queue

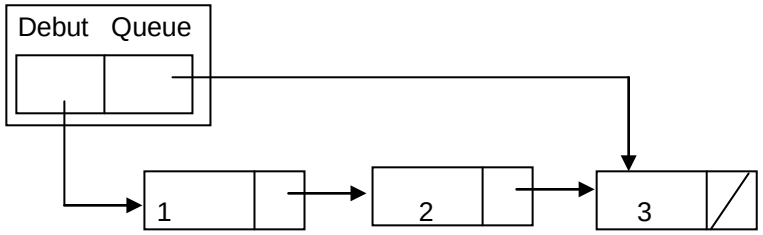
De même que pour les files statiques nous présentons la déclaration et les sous algorithmes de bases détaillés pour le type statique. Et dans un souci d'uniformisation nous utilisons les mêmes noms mais en ajoutant un D (pour rappeler Dynamique) à la fin de chaque nom pour faire la différence.

Note : Il n'y a pas de file pleine pour les files dynamique. La seule possibilité de ne pas pouvoir ajouter un élément c'est d'avoir une mémoire pleine, cas que l'on ne prend pas en considération ici.

II-2-1- Déclaration d'une file dynamique

```
Type
   Liste = ^Elem
   Elem = Enregistrement
         Info : Type_C ;
         Suiv : ^Elem
   Fin ;
   FileD = Enregistrement
         Sommet, Queue : Liste
   Fin ;
```

FileD



Note : Les données enregistrées dans la file peuvent être des entiers, des réels, des caractères, des chaînes de caractères, des booléens, des tableaux, des pointeurs de listes ou encore des piles ou files.

II-2-*2- Initialisation d'une file dynamique*

```
Procedure Init_FileD (F : FileD) ;
Debut
      F. Debut ← Nil ;
      F. Queue ← Nil
Fin ;
```

II-2-*3- Vérification de File vide dynamique*

```
Fonction File_videD (F : FileD) : Booleen ;
Debut
   Si F. Queue = Nil Alors
         File_videD ← vrai
   Sinon
         File_videD ← faux
   FinSi ;
Fin ;
```

Une autre façon plus compacte d'écrire cette fonction est la suivante :

```
Fonction File_videD (F : FileD) : Booleen ;
Debut
    File_videD ← F. Queue = Nil;
Fin ;
```

Le nom de la fonction recevra le résultat de la comparaison (vrai ou faux) entre la queue et le Nil.

II-2-*4- Ajout d'une nouvelle valeur à une file dynamique*

L'ajout d'une valeur à une File dynamique revient à une insertion à la fin de liste avec l'adresse du dernier élément dans Queue. Le cas d'un enfiler sur file vide nécessite l'initialisation du Debut à l'adresse du nouveau nœud.

```
Procedure EnfilerD (Var F : FileD ; X : Type_C) ;
Variable
   Pt : Liste ;
Debut
        Allouer (Pt) ;
        Pt^.Info ← X ;
        Pt^. Suiv ← Nil ;
        Si File_videD (F) Alors
           F. Debut ← Pt
        Sinon
          F. Queue^.Suiv ← Pt
        FinSi ;
        F. Queue ← Pt ;
Fin ;
```

II-2-*5- Suppression d'une valeur de la file dynamique*

La suppression d'une valeur dans une file dynamique revient à effectuer une suppression physique d'un nœud (le premier de liste) et à récupérer dans un paramètre, passé par variable, la donnée enregistrée.

```
Procedure DefilerD (Var F: FileD, X : Type_C) ;
Variable
   Pt : Liste ;
Debut
   Si File_videD (F) Alors
        Ecrire('Impossible la file est vide')
   Sinon
     Debut
        Pt ← F. Debut ;
        X ← F.Debut^.Info ;
        F. Debut ← F. Debut^.Suiv ;
        Si F. Debut = Nil Alors
           F. Queue ← Nil
        FinSi  /* si debut devient Nil cela veut dire que la file a été vidée et Queue doit devenir Nil*/
        Liberer (Pt)
      Fin
   FinSi ;
Fin ;
```

III- Exercices

Dans les exercices avec piles et files il est suffit de faire appel aux sous algorithmes de base définis dans les sections précédentes.

Exercice 1 :

Ecrire une procédure afficher (F1) qui affiche tous les éléments d'une file de mots et une autre défilerJusqua (F1, elt) qui défile la file jusqu'à l'élément elt. L'élément elt n'est pas défilé. Si l'élément n'appartient pas à la file, alors la fonction défile toute la file.

```
Procedure afficher (F1 : File) ;
Variable
   Mot : Chaine de caractere ;
Debut
    Tantque Non (File_vide (F1)) Faire
     Debut
          Defiler (F1, Mot) ;
          Ecrire (Mot)
     Fin ;
   FinTantque
Fin ;
```

Si l'on considère que la file est dynamique il suffit de faire appelle aux sous-algorithmes de base des files dynamiques comme suit :

```
Procedure afficher (F1 : FileD) ;
Variable
   Mot : Chaine de caractere ;
Debut
    Tantque Non (File_videD (F1)) Faire
     Debut
          DefilerD (F1, Mot) ;
          Ecrire (Mot)
     Fin ;
    FinTantque
Fin ;
```

```
Procedure défilerJusqua (Var F1 : File ; elt : chaine de caractere) ;
Variable
   Mot : Chaine de caractere ;
   B : Booleen ;
Debut
    B ← Vrai ;
    Tantque (B ) et Non (File_vide (F1)) Faire
     Debut
          Defiler (F1, Mot) ;
          Si Mot = elt Alors
            Debut
              B ← Faux ;
              Enfiler (F1, elt)
            Fin
          FinSi
      Fin ;
Fin ;
```

Exercice 2 :

Ecrire une procédure qui inverse une pile P1de réels. Doit-on utiliser une pile ou une file ?

Pour inverser une pile on aurait besoin d'utiliser une file où l'on enfile ce qui a été dépilé puis empilant à partir de la file les éléments seront installés dans l'ordre inverse.

```
Procedure inver (var P1 : Pile);
Variable
     Y: Reel;
     F: File;
Debut
      Init_File (F);
      Tantque Non (Pile_vide (P1)) Faire
        Debut
           Depiler (P1, Y) ;
           Enfiler (F, Y)
        Fin ;
      FinTantque ;
      Init_Pile (P1);
      Tantque Non (File_vide (F)) Faire
        Debut
           Defiler (F, Y) ;
           Empiler (P1, Y)
         Fin ;
     FinTantque
Fin ;
```

Question Supplémentaire : Que faire si l'utilisation des files était interdite ?

Exercice 3 :

On se donne une pile P1 contenant des entiers positifs.
Ecrire un algorithme pour déplacer les entiers de P1 dans une pile P2 de façon à avoir dans P2 tous les nombres pairs en dessus des nombres impairs en gardant l'ordre d'apparition des nombre pairs et en inversant l'ordre d'apparition des nombres impairs. Au retour à l'algorithme appelant on doit retrouver P1 initiale.

```
Procedure Pair (P1 : Pile; Var P2: Pile);
Variable
     Y: Entier;
     P3: Pile;
Debut
      Init_Pile (P2);
      Init_Pile (P3);
      Tantque Non (Pile_vide (P1)) Faire
        Debut
           Depiler (P1, Y) ;
           Si Y Mod 2 = 0 Alors
              Empiler (P3, Y)
           Sinon
              Empliler (P2, Y)
           FinSi ;
         Fin
      FinTantque ;
      Tantque Non (Pile_vide (P3)) Faire
        Debut
           Depiler (P3, Y) ;
           Empiler (P2, Y)
         Fin ;
     FinTantque
Fin ;
```

Exercice Supplémentaire :

- Montrer comment implémenter une file en utilisant deux piles ; écrire les opérations enfiler, défiler de ce cas de figure.
- Montrer comment implémenter une pile en utilisant deux files ; écrire les opérations empiler, dépiler de ce cas de figure.

Pour les questions supplémentaires vous pouvez envoyez vos réponses à ilabed8@gmail.com si vous voulez les faire corriger.