

Chapitre 11

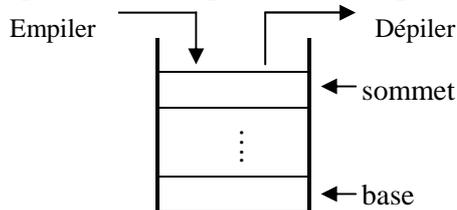
Piles et files

1. Piles

Une pile est une liste chaînée d'informations dans laquelle :

- Un élément ne peut être ajouté qu'au sommet de la pile,
- Un élément ne peut être retiré que du sommet de la pile.

Il s'agit donc d'une structure de type LIFO (Last In First Out). On ne travaille que sur le sommet de la pile. Les piles sont comparables à des piles d'assiettes.



On associe à une pile les termes de :

- PUSH pour empiler c'est-à-dire ajouter un élément,
- POP pour dépiler c'est-à-dire supprimer un élément.

Les piles servent à revenir à l'état précédent et sont utilisées pour :

- implanter les appels de procédures (pour revenir à l'état d'avant l'appel),
- annuler une commande,
- évaluer des expressions arithmétiques,
- etc.

1.1. Opérations autorisées

Les opérations autorisées avec une pile sont :

- empiler, toujours au sommet, et jusqu'à la limite de la mémoire,
- dépiler, toujours au sommet, si la pile n'est pas vide,
- vérifier si la pile est vide ou non.

On peut implémenter une pile dans un tableau (pile statique) ou dans une liste chaînée (pile dynamique). C'est l'implémentation en liste chaînée qui est présentée ici. Le sommet de la pile est le premier élément et le pointeur de tête pointe sur ce sommet. Il faut commencer par définir un type de variable pour chaque élément de la pile. La déclaration est identique à celle d'une liste chaînée, par exemple pour une pile de chaînes de caractères :

```
Type Pile = ^Element
Type Element = Structure
                Info : chaîne de caractères
                Suivant : Pile
            Fin Structure
```

1.1.1. Empiler

Empiler un élément revient à faire une insertion en tête dans la liste chaînée.



Procédure Empiler (*Entrée/Sortie* Tête : Pile, *Entrée* Valeur : chaîne de caractères)

/ Ajout d'un élément dans une pile passée en paramètre */*



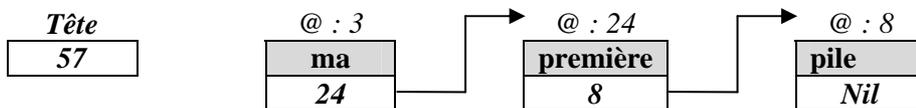
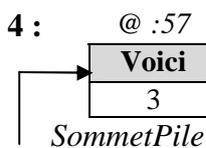
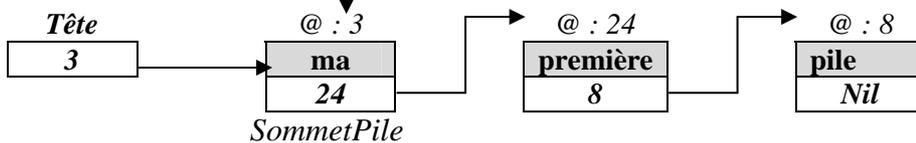
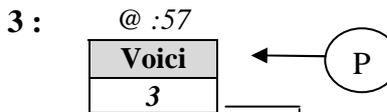
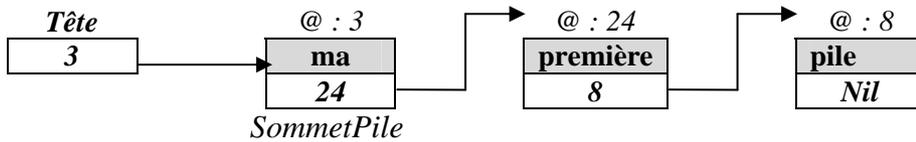
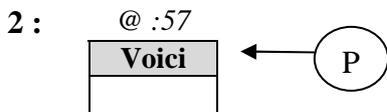
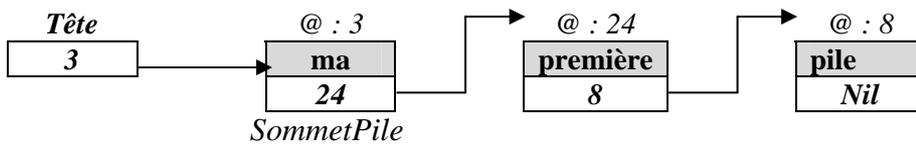
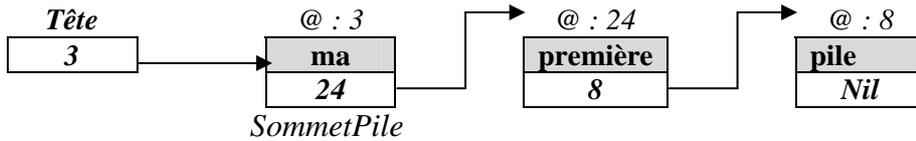
variable locale

P : Pile */* pointeur auxiliaire */*

DEBUT

- 1 Allouer(P) */* Réserve un espace mémoire pour le nouvel élément */*
- 2 P^.Info = Valeur */* stocke dans l'Info de l'élément pointé par P la valeur passée en paramètre */*
- 3 P^.Suivant = Tête */* stocke dans l'adresse du Suivant l'adresse de Tête */*
- 4 Tête ← P */* Tête pointe maintenant sur le nouvel élément */*

FIN



En faisant référence à la procédure *InsererEnTete* dans les exercices à propos des listes chaînées simples, nous pouvons écrire :

Procédure Empiler (Entrée/Sortie Tête : Pile; Entrée Element : variant)
/* Ajout de l'élément au sommet de la pile P passée en paramètre */



DEBUT

| InsererEnTete(Tête, Element)

IFIN

Puisqu'il s'agit d'une gestion dynamique de la mémoire, il n'est pas nécessaire de vérifier si la pile est pleine.

1.1.2. Dépiler

Dépiler revient à faire une suppression en tête.

Procédure Dépiler (Entrée/Sortie Tête : Pile) 

/* Suppression de l'élément au sommet de la pile passée en paramètre */

Variable locale

P : Pile /* Pointeur nécessaire pour libérer la place occupée par l'élément dépilé */

DEBUT

| /* Vérifier si la pile est vide */

| **SI** Tête <> NIL **ALORS** /* la pile n'est pas vide donc on peut dépiler */

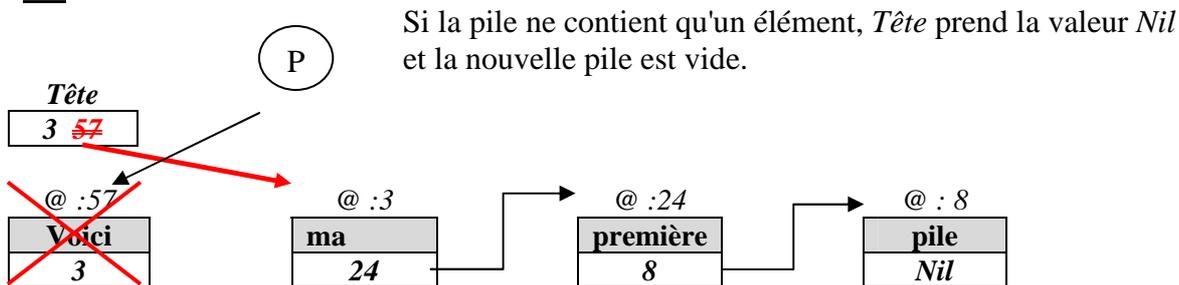
| | P ← Tête /* on garde l'adresse du sommet pour désallouer */

| | Tête ← Tête^.Suivant /* P va pointer sur le 2ème élément de la pile qui devient le sommet */

| | Désallouer(P)

| **FINSI**

FIN



1.1.3. Procédures et fonctions de base

Procédure InitPile (Sortie Tête : Pile)

/* Initialise la création d'une pile */

DEBUT

| Tête ← Nil

FIN

Fonction PileVide (Tête : Pile) : booléen

/* indique si la pile est vide ou non */

DEBUT

| Retourner (Tête = Nil)

FIN

Fonction SommetPile (Tête : Pile) : variant

/* renvoie la valeur du sommet si la pile n'est pas vide */

DEBUT

| SommetPile ← Tête^.Info

FIN

2. Files



Une file, ou file d'attente, est une liste chaînée d'informations dans laquelle :

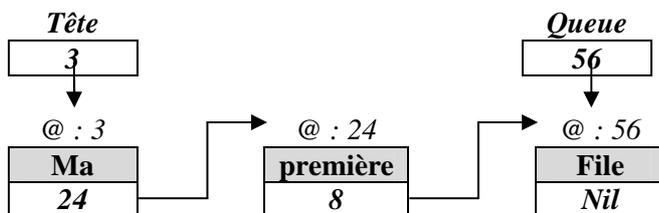
- Un élément ne peut être ajouté qu'à la queue de la file,
- Un élément ne peut être retiré qu'à la tête de la file.

Il s'agit donc d'une structure de type FIFO (First In First Out). Les données sont retirées dans l'ordre où elles ont été ajoutées. Une file est comparable à une queue de clients à la caisse d'un magasin.

Les files servent à traiter les données dans l'ordre où on les a reçues et permettent de :

- gérer des processus en attente d'une ressource système (par exemple la liste des travaux à éditer sur une imprimante)
- construire des systèmes de réservation
- etc.

Pour ne pas avoir à parcourir toute la liste au moment d'ajouter un élément en queue, on maintient un pointeur de queue. Attention une file peut très bien être simplement chaînée même s'il y a un pointeur de queue.



2.1. Opérations autorisées

Les opérations autorisées avec une file sont :

- enfiler toujours à la queue et jusqu'à la limite de la mémoire,
- défiler toujours à la tête si la file n'est pas vide,
- vérifier si la file est vide ou non.

On peut implémenter une file dans un tableau (file statique) ou dans une liste chaînée (file dynamique). C'est l'implémentation en liste chaînée qui est présentée ici. Le pointeur de tête pointe sur le premier élément de la file, et le pointeur de queue sur le dernier. Il faut commencer par définir un type de variable pour chaque élément de la file. La déclaration est identique à celle d'une liste chaînée, par exemple pour une file de chaînes de caractères :

```

Type File = ^Element
Type Element = Structure
                Info : chaîne de caractères
                Suivant : File
            Fin Structure
  
```

2.1.1. Enfiler

Enfiler un élément consiste à l'ajouter en queue de liste. Il faut envisager le cas particulier où la file était vide. En effet, dans ce cas, le pointeur de tête doit être modifié.



Procédure Enfiler (*Entrée/Sortie* : Tête, *Queue* : File, *Entrée Valeur* : variant)

/ Ajout d'un élément dans une file */*

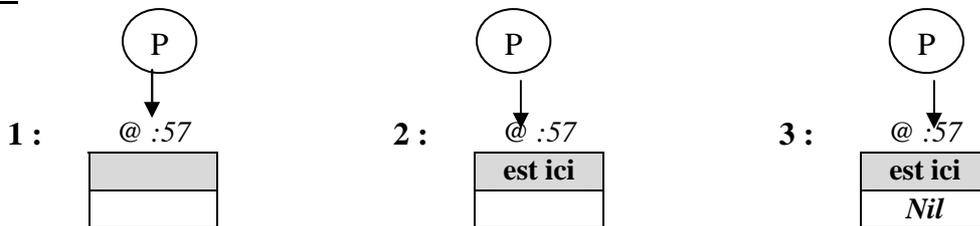
Variable locale

P : File */* Pointeur nécessaire pour allouer la place au nouvel élément */*

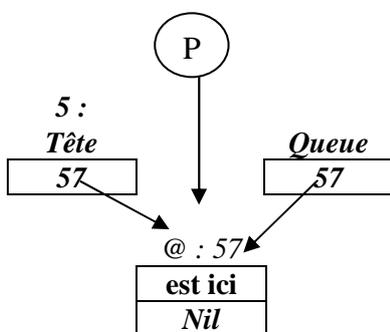
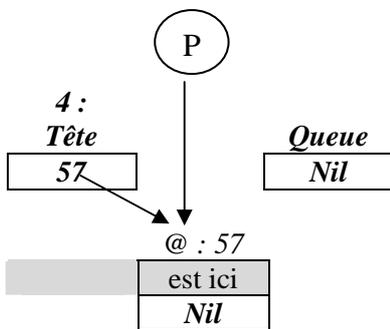
DEBUT

```

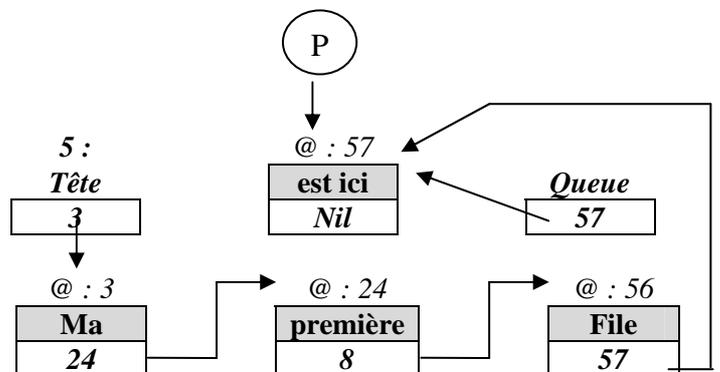
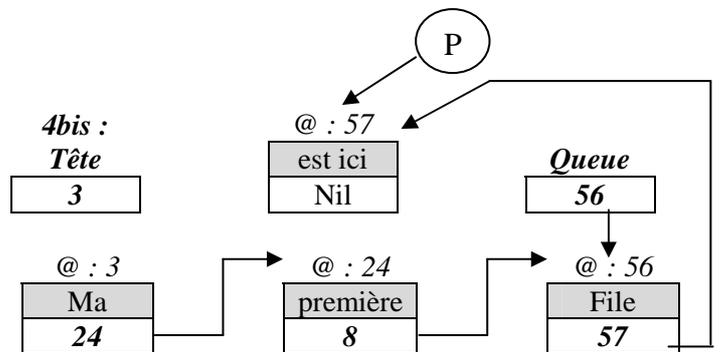
1 Allouer(P) /* Réserve un espace mémoire pour le nouvel élément */
2 P^.Info ← Valeur /* stocke la valeur dans l'Info de l'élément pointé par P */
3 P^.Suivant ← Nil /* stocke Nil (ce sera le dernier de la file) dans Suivant */
4 SI Tête = Nil ALORS /* file vide */
   Tête ← P /* Tête pointe maintenant sur l'élément unique */
4bis SINON /* il y a au moins un élément */
   Queue^.Suivant ← P /* le nouvel élément est ajouté au dernier */
5 FINSI
   Queue ← P /* Queue pointe maintenant sur l'élément ajouté */
FIN
    
```



Si Tête = Nil



Si Tête ≠ Nil



En faisant référence à la procédure *InsererEnTete* dans les exercices sur des listes chaînées simples, nous pouvons écrire :



Procédure Enfiler (*Entrée/Sortie* : Tête, Queue : File, *Entrée Valeur* : variant)
/* Ajout d'un élément dans une file */

DEBUT

SI Tete = Nil **ALORS** /* Tete = Nil et Queue = nil */

InsererEnTete(Tete, Valeur)

Queue ← Tete

SINON

InsererEnTete(Queue^.Suivant, Valeur)

Queue ← Queue^.Suivant

FINSI

FIN

2.1.2. Défiler

Défiler est équivalent à dépiler et consiste à supprimer l'élément de tête si **la file n'est pas vide**. Si la file a un seul élément, il faut mettre à jour le pointeur de queue car on vide la file. Il faut conserver l'adresse de l'élément qu'on supprime pour libérer sa place.

Procédure Défiler (*Entrée/Sortie* Tête, Queue : File, *Sortie Valeur* : chaîne de caractères)

/* Suppression de l'élément de tête de la file passée en paramètre */

Variable locale

P : File /* Pointeur nécessaire pour libérer la place de l'élément supprimé */

DEBUT

SI Tête <> NIL **ALORS** /* la liste n'est pas vide donc on peut défiler */

Valeur ← Tête^.info /* on récupère l'élément de tête */

P ← Tête /* on garde l'adresse du sommet pour désallouer */

Tête ← Tête^.Suivant /* P va pointer sur le 2ème élément de la pile qui devient le sommet */

Désallouer(P)

Si Tête = Nil **ALORS**

Queue ← Nil /* la file a été vidée */

FINSI

FINSI

FIN

En faisant référence à la procédure *SupprimerEnTete* dans les exercices à propos des listes chaînées simples, nous pouvons écrire :

Procédure Défiler (*Entrée/Sortie* Tête, Queue : File, *Sortie Val* : variant)

/* Suppression d'un élément dans une file */

DEBUT

SI Tete <> Nil **ALORS**

Val ← Tete^.Info

SupprimerEnTete(Tete)

SI tête = Nil

Queue ← Nil

FINSI

FINSI

FIN

WEBOGRAPHIE

<http://www.siteduzero.com/tutoriel-3-36245-les-listes-chaínees.html>

http://liris.cnrs.fr/pierre-antoine.champin/enseignement/algo/listes_chaínees/

<http://deptinfo.cnam.fr/Enseignement/CycleA/SD/cours/structuress%E9quentielleschain%E9es.pdf>

<http://pauillac.inria.fr/~maranget/X/421/poly/listes.html#toc2>

<http://wwwens.uqac.ca/~rebaine/8INF805/courslistespilesfiles.pdf>