CHAPITRE 2

Les Types Abstraits de Données

2.1 Introduction

Les Types Absrtaits de Données sont les types tels que on n'impose pas une mise-en-œuvre particulière dans la mémoire.

Un Type abstrait de données consiste en un ensemble de données (valeurs) plus une stratégie d'accès à ces données.

Un type abstrait de données peut être mis en œuvre de plusieurs mainères dans la mémoire.

2.2 Les listes

2.2.1 Définition

Une liste est une structure particulièrement souple : elle peut grandir ou retrécir à volonté. Ses éléments, qui sont accessibles à tout moment, peuvent être insérés ou supprimés à tout moment à n'importe quel endroit.

D'un point de vue mathématique, une liste est une suite, vide ou non, d'éléments d'un type donné (noté par la suite *TypeÉlément*). Il est habituel de représenter une telle suite par : $a_1, a_2, \ldots, a_n$ où n est un entier positif ou nul et où chaque a_i est de type *TypeÉlément*. Une liste peut être concatenée à une autre ou divisée en deux sous-`listes`. Le nombre n d'éléments s'appelle la longueur de la liste.

- Si $n \geq 1$, on dit que a_1 est le premier élément de la liste et a_n est le dernier.
- Si $n = 0$, on parle, alors, d'une liste vide.
- On dit que :
  - a_i précède a_{i+1}, $i = 1, \ldots, n-1$;
  - a_i suit $a_{i-1}, i = 2, \ldots, n$;
  - a_i se trouve à la position i.

2.2.2 Les opérations sur les listes

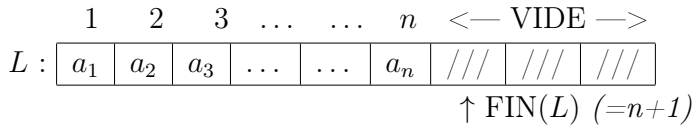
Notation

— L : une `listes` d'objets de type *TypeÉlément*;
— x : un objet e type *TypeÉlément*;
— p : de type entier (désigne la postion d'un élément).

FIN(L)

Cette fonction retourne la position suivant la n^{ieme} position d'une liste L de n éléments. Cette position varie suivant que la liste grandit ou rétricit.

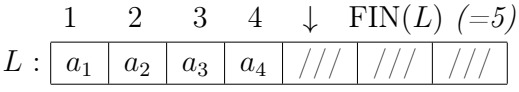
Exemple 1 ()



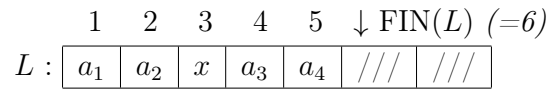
INSERER(x, p, L)

Cette procédure insére l'objet x à la position p dans la liste L, avec déplacement de l'élément précédemment situé en p et de tous les suivants d'une position vers la fin de la liste.

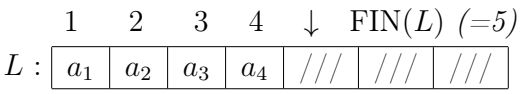
Exemple 2 ($1 \leq p \leq n$)
Avant Insertion :



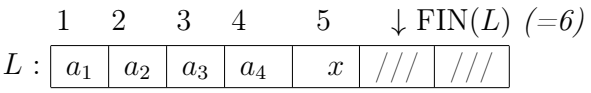
Après insertion de x à $p = 3$



Exemple 3 (p =Fin(L))
Avant Insertion :



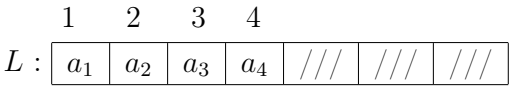
Après insertion de x à $p = 5$



LOCALISER(x, L)

Cette fonction retourne la position p de x dans la liste L. Si x apparaît plusieurs fois, c'est la position de la 1^{iere} occurrence qui est retournée. Si x n'est pas dans la liste, c'est la position FIN(L) qui est renvoyée.

Exemple 4 ()



LOCALISER(a_2, L) $\rightsquigarrow$?
LOCALISER(a_6, L) $\rightsquigarrow$?

ACCEDER(p, L)

Cette fonction retourne l'élément se trouvant à la position p de la liste L. L'opération est impossible si $p \geq$ FIN(L) ou si L est vide. Les éléments à extraire doivent avoir le même type que celui des objets retournés par la fonction ACCEDER().

Exemple 5 ()

L :

a_1	a_2	a_3	a_4	///	///	///

ACCEDER(3, L) $\rightsquigarrow$?
ACCEDER(5, L) $\rightsquigarrow$?

SUPPRIMER(p, L)

Elle permet de supprimer l'élément se trouvant à la position p de la `liste` L. L'opération est impossible si L est vide ou si $p \geq \text{FIN}(L)$.

Exemple 6 ()

Avant suppression :

	1	2	3	4	5	$\downarrow$ FIN(L) *(=6)*	
L :	a_1	a_2	a_3	a_4	a_5	///	///

Après suppression de l'élément à $p = 3$: SUPPRIMER(3, L)

	1	2	3	4	$\downarrow$	FIN(L) *(=5)*	
L :	a_1	a_2	a_4	a_5	///	///	///

SUIVANT(p, L) et PRECEDENT(p, L)

Ces deux fonctions retournent respecivement la position suivante et précédente à la position p dans la liste L.

— Si p est la dernière position de cette liste, SUIVANT(p, L) = FIN(L) ;
— Si p = FIN(L), SUIVANT() n'est pas définie ;
— Si $p = 1$, PRECEDENT() n'est pas définie ;
— Si L est vide, ces deux opérations n'ont pas de sens.

Exemple 7 ()

1	2	3	4

L :

a_1	a_2	a_3	a_4	///	///	///

SUIVANT(4, L) $\rightsquigarrow$?
SUIVANT(5, L) $\rightsquigarrow$?
PRECEDENT(3, L) $\rightsquigarrow$?
PRECEDENT(1, L) $\rightsquigarrow$?

Application 1 ()
Écrire une procédure permettant de prendre une liste L en argument et en élimine toutes les répétitions. Les éléments de la liste sont de type *TypeÉlément* et une liste de tels objets est du type `LISTE`. On suppose aussi l'existence d'une fonction Identique(x, y) où x et y sont de type *TypeÉlément*, qui retourne la valeur `VRAI` si x et y sont identiques et `FAUX` sinon.

2.2.3 Mise-en-œuvre des listes

Une liste peut être mise-en-œuvre par un tableau, le type LISTE consiste, alors, en un enregistrement à 2 champs.

- Le premier champs est un tableau dont la taille est suffisante pour contenir la liste.
- Le second champs est un entier « dernier »indiquant la position du dernier élément du tableau.

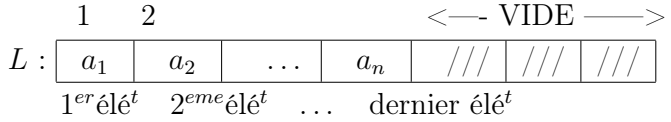
La foncition FIN(L) ne fait que renvoyer la valeur (Dernier+1).

Déclaration

Les déclarations suivantes sont nécessaires :

```
const  LongMax = 100
Type  LISTE = enregistrement
                    Eléments : Tableau [1..LongMax] de TypeElement
                    Dernier :entier
              finenregistrement
```

Exemple 8 ()



Exemple 9 (La fonction FIN())

```
fonction  FIN (L : liste) : entier
début
        FIN ←L.Dernier +1
```

```
fin
```

Application 2 ()
Écrire les sous-programmes suivants : SUIVANT(), PRECEDENT(), INSERER(), SUPPRIMER() et LOCALISER().

2.3 Les piles

2.3.1 Définition

Une pile est un type de liste particulière dans laquelle toute suppression d'éléments se fait à une extrimité appelée le dessus ou le « *sommet* »de la pile.

Dans une pile, l'élément supprimé est celui le plus récemment inséré : la pile met-en-oeuvre le principe « dernier entré, premier sortant », ou LIFO (Last In, First Out).

2.3.2 Les opérations sur les piles

L'opération « INSERER »dans une pile est souvent appelée « EMPILER »et l'opération « SUPPRIMER », qui ne prend pas d'argument, est souvent appelée « DEPILER ».

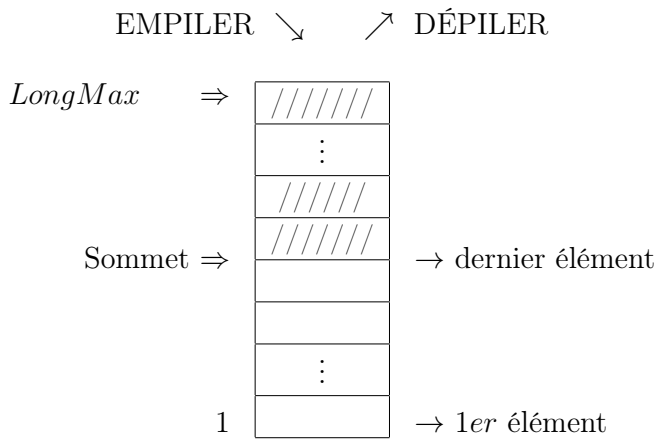
Les opérations possibles sur les piles sont souvent les suivants :

— RAZ(P) : vider le contenu de la pile P ;
— VIDE(P) : cette fonction retourne `VRAI` si P est vide et `FAUX` sinon ;
— SOMMET(P) : retourne (sans le dépiler) l'élément au sommet de la pile P ;
— DEPILER(P) : supprime physiquement l'élément au sommet de P ;
— EMPILER(x, P) : insère l'élément x au sommet de P. L'ancien élément le plus haut devient le 2^{eme} et ainsi de suite...

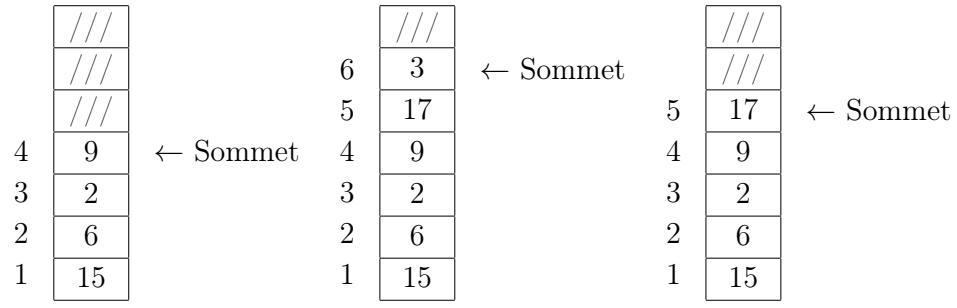
2.3.3 Mise en oeuvre des piles par les tableaux

D'une façon générale, une pile peut être mise-en-oeuvre par un tableau de taille $LongMax$, comme suit :

Exemple 10 ()



Exemple 11 ()



P — après : EMPILER(17, P), EMPILER(3, P) — après : DÉPILER(P)

SOMMET(P) $\rightsquigarrow$?

VIDE(P) $\rightsquigarrow$?

Pour cette mise-en-oeuvre des piles à partir d'un tableau, il faut définir le type de données abstrait PILE par :

```
const  LongMax = ...
Type
PILE = enregistrement
                    Eléments : Tableau [1..LongMax] de TypeElement
                    Sommet :entier
       finenregistrement
```

Remarque

si le sommet = 0, alors la pile est vide.

Application 3 ()
Écrire les sous-programmes RAZ(P), VIDE(P), SOMMET(P), DEPILER(P) et EMPILER(x, P).

2.4 Les files

2.4.1 Définitions

Une file est un autre type particulier de liste, où les éléments sont insérés en queue et supprimés de la tête. Une file met en oeuvre le principe « *premier entré, premier sorti* »ou FIFO (First In, First Out). Les opérations sur les files sont analogues à celles définies sur les piles. La principale différence provient du fait que les insertions se font en fin de la liste (queue) plutôt qu'en tête.

2.4.2 Les opérations sur les files

— RAZ(F) : transforme la file en une file vide ;
— TETE(F) : cette fonction retourne l'élément en tête de la file F ;
— ENFILER(x, F) : Insère l'élément x à la fin de la file F ;
— DEFILER(F) : Supprime le 1^{er} élément de a file F ;
— VIDE(F) : cette fonction retourne `VRAI` si la file F est vide et `FAUX` sinon.

2.4.3 Mise-en-oeuvre des files par des pointeurs

Comme pour les piles, toute mise-en-oeuvre de liste est valable sur les files. On définit les nœuds de nos files par la déclaration suivante :

```
type NŒUD = enregistrement
                    val : TypeElement
                    suiv : ^NŒUD
            finenregistrement
```

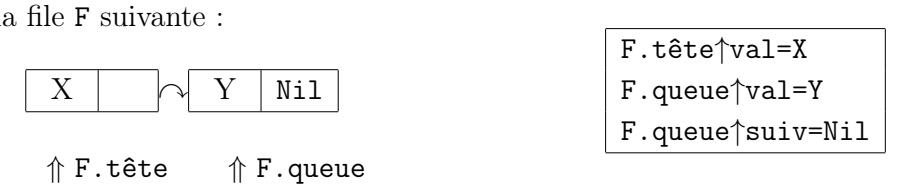
On peut ensuite définir une file comme la donnée de 2 pointeurs : un 1^{er} sur la tête de la file et un second sur la queue.

On définit alors le type `FILE` par :

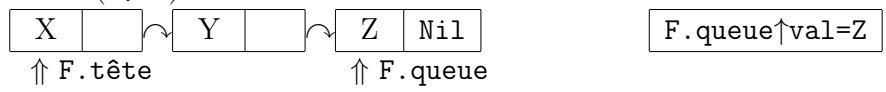
```
type FILE =enregistrement
                    tête, queue : ^NŒUD
           finenregistrement
```

Exemple 12 ()

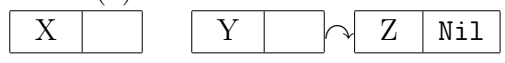
Soit la file `F` suivante :

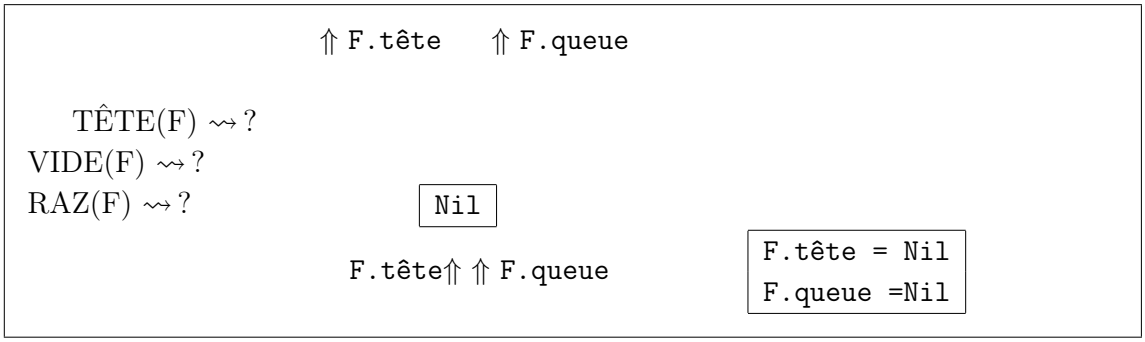


ENFILER(Z, F)



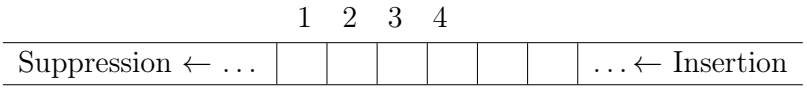
DEFILER(F)





Application 4 ()
Écrire les sous-programmes : RAZ(F), VIDE(F), TETE(F), ENFILER(x, F) et DEFILER(F).

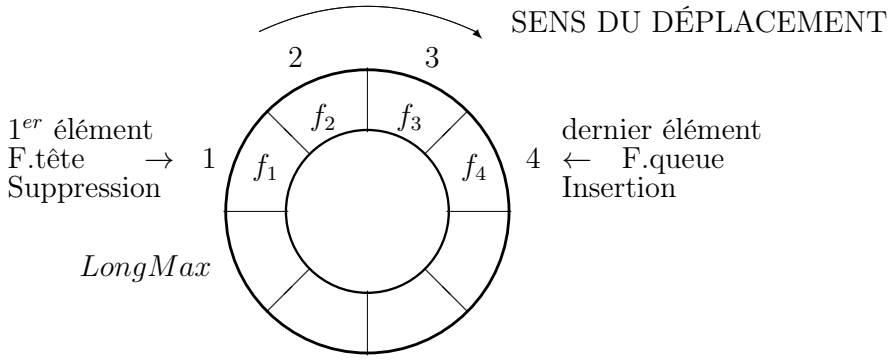
2.4.4 Mise-en-oeuvre des files par tableaux



Il faut noter que la procédure DEFILER, dont l'action est de supprimer le 1er élément, provoque le déplacement d'une position vers la tête de tous les éléments suivants de la file.

Pour éviter ces décalages successifs, on peut penser à une représentation de la file par un tableau circulaire où la dernière position est directement suivie par la première.

Exemple 13 ()



Ici la position `F.tête` du 1^{er} élément est variable dans le temps.

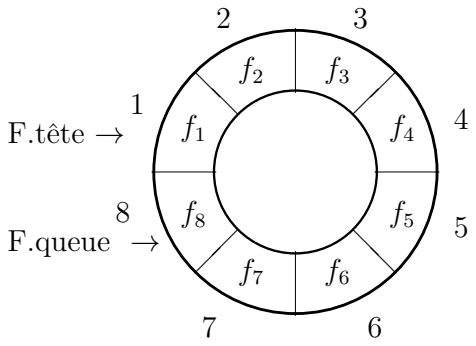
Quand une insertion est effectuée, la position `F.queue` est déplacée d'une position vers l'avant et l'élément est inséré à cette position.

Une suppression provoque simplement le déplacement de `F.tête` d'une position vers l'avant. Formellement, une file est définie par :

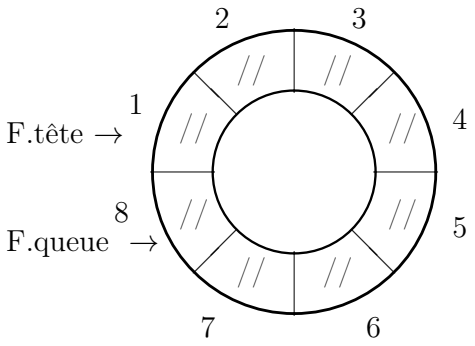
```
const  LongMax = ...
type  FILE = enregistrement
                          Elément : tableau [1... LongMax] de TypeElément
                          tête, queue : entier
             Finenregistrement
```

Exemple 14 ()

FILE PLEINE



FILE VIDE



FILE PLEINE :
F.tête = 1
F.queue = 8

FILE VIDE :
F.tête = 1
F.queue = 8

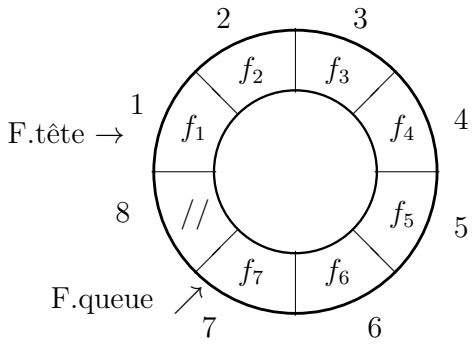
Il n'est pas possible de distiguer une file vide de la même file pleine.

Une manière de surmonter ce phénomène est de ne pas permettre le remplissement entier de la file. Autrement dit, la file ne peut pas contenir plus que (*LongMax* -1) éléments.

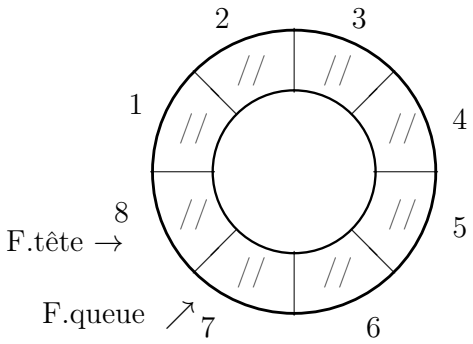
Dans ce cas la file est vide si les deux positions `F.tête` et `F.queue` sont adjacents.

Exemple 15 ()

FILE PLEINE



FILE VIDE



FILE PLEINE :
F.tête = 1
F.queue = 7

FILE VIDE :
F.tête = 8
F.queue = 7

Application 5 ()
Écrire une fonction AVANCER permettant d'incrémenter une position p dans un tableau ciculaire de taille *LongMax*.