CHAPITRE II

Structures de Données Elémentaires

Listes chainées

Représentation à l'aide de tableau et opérations élémentaires

La liste L = {a_1, a_2, …, a_n} est représentée à l'aide des tableaux *élément* et *suivant* comme suit :

	élément	**suivant**
tête=1	a_1	2
2		3
	…..	…..
n	a_n	-1
libre		
max		

Initialisation** des listes **tête** et **libre ;
(initialement la liste **tête** est vide et la liste **libre** est pleine *)*
*élément, suivant : tableau [1..max] d'**entier** ;*
***var** tête, libre, p :-1..max*

```
début
        pour p :=1 à max-1 faire
                début  élément[p] := blanc ;
                        suivant[p] := p+1 ;
                fin ;
        élément[max] := blanc ; suivant[max] := -1
        tête := -1 ;
        libre := 1 ;
fin
```

procédure liste-vide ;
***entrée** : tête, libre : -1..max ;*
***sortie** : vrai ou faux ;*

```
début
        si tête = -1 alors retourner (vrai) sinon retourner (faux) ;
fin
```

```
procédure liste-pleine ;
entrée : tête, libre : -1..max ;
sortie : vrai ou faux ;

début
        si libre = -1 alors retourner (vrai) sinon retourner (faux) ;
fin
```

```
procédure recherche ;
(* recherche d'un entier x dans une liste non triée *)
entrée : tête : -1..max ; x : entier ;
sortie : position de x dans la liste ;

var     p : -1..max ;
début
        si (non liste-vide) alors
          début  p := tête ;
                 tant que (élément[p] ≠ x) et (suivant[p] ≠ -1) faire
                          p := suivant[p] ;
                 si (élément[p]=x) alors retourner(p)
                          sinon signaler ('x n'existe pas dans la liste') ;
          fin
        sinon signaler ('liste vide') ;
fin
```

```
procédure suppression ;
(* suppression d'un entier x d'une liste non triée *)
entrée : tête, libre : -1..max ; x : entier ;
sortie : tête, libre : -1..max ;
var p, q : -1..max ;
début
        si (non liste-vide) alors
          début  p := tête ;
                 tant que (élément[p] ≠ x) et (suivant[p] ≠ -1) faire
                          début   q := p ;
                                  p := suivant[p] ;
                          fin ;
                 si (élément[p] = x) alors
                 début si (p = tête) alors tête := suivant[p]
                            sinon   suivant[q] :=  suivant[p];
                       suivant[p] := libre ;
                       libre := p ;
                 fin
           fin
           sinon signaler ('liste vide') ;
fin
```

```
procédure insertion ;
(* insertion d'un entier x dans une liste triée *)
entrée : tête, libre : -1..max ; x : entier ;
sortie : tête, libre : -1..max  ;

var p, q, temp : -1..max   ;
début
            p := tête ;
            élément [libre] := x ;
            temp := suivant[libre] ;
            si (non liste-vide) et (non liste-pleine) alors
              début tant que (élément[p]  < x) et (suivant[p] ≠ -1) faire
                         début   q := p ;
                                 p := suivant[p] ;
                        fin ;
                        si (élément[p] > x) alors
                              si (p = tête) alors début suivant[libre] := tête ;
                                                        tête := libre ;
                                                fin
                                       sinon début  suivant[libre] := p ;
                                                    suivant[q] := libre ;
                                               fin
                        sinon début suivant[libre]:=  suivant[p];
                                    suivant[p] := libre ;
                                  fin ;
                        libre := temp ;
               fin
               sinon si (liste-vide) alors début tête := libre ;
                                                 suivant[tête] := -1 ;
                                                 libre := temp ;
                                            fin
                                       sinon signaler ('liste pleine')
fin
```

Exemple 2.1

Considérer la liste triée suivante constituée de 5 nombres entiers relatifs ordonnés et représentée par les tableaux *élément* et *suivant :*

	élément	suivant
Tête=1	-32	2
2	0	3
3	12	4
4	78	5
5	341	-1
Libre=6		7
7		8
8		-1

L'insertion du nombre 45 engendre la configuration suivante de la liste et le contenu suivant des tableaux :

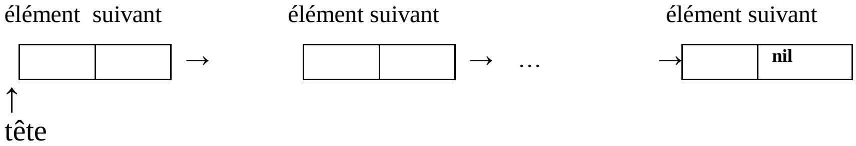
	élément	suivant
Tête=1	-32	2
2	0	3
3	12	6
4	78	5
5	341	-1
6	45	4
Libre=7		8
8		-1

La suppression de 78 de cette nouvelle configuration donne le tableau suivant :

	élément	suivant
Tête=1	-32	2
2	0	3
3	12	6
Libre=4	78	7
5	341	-1
6	45	5
7		8
8		-1

Représentation à l'aide de structure dynamique et opérations élémentaires

Tous les traitements vus dans la section précédente peuvent être conduits sur des structures de données dynamiques, en l'occurrence sur des listes dynamiques. La représentation graphique d'une liste dynamique est la suivante :



```
type
        liste :    enregistrement élément : entier ; suivant : ↑liste ; fin ;

var
        tête : ↑liste ;
```

```
Initialisation de la liste ;
(* initialement la liste tête est vide  *)
var tête : ↑liste ;

tête := nil ;
```

```
procédure liste-vide ;
entrée : tête : ↑liste ;
sortie : vrai ou faux ;

début   si tête = nil alors retourner (vrai) sinon retourner (faux) ;
fin
```

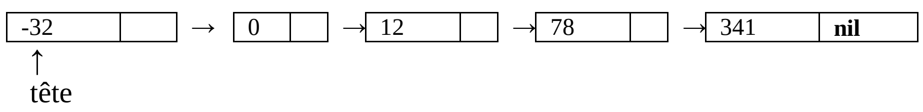
```
procédure recherche ; (* rechercher un entier x dans une liste non triée *)
entrée : tête : ↑liste ; x : entier ;
sortie : position de x dans la liste ;

var p : ↑liste ;
début   p := tête ;
        si (liste-vide) alors signaler ('liste vide')
        sinon début   tant que (p↑.élément ≠ x) et (p↑.suivant ≠ nil) faire
                                p := p↑.suivant;
                      si (p↑.élément = x) alors retourner(p)
                            sinon signaler ('x n'existe pas dans la liste') ;
              fin
fin
```

```
procédure insertion ; (* insertion de x dans une liste triée *)
entrée : tête : ↑liste ;  x : entier ;
sortie : tête : ↑liste ;

var p, q, libre : ↑liste;
début   p := tête ;
        nouveau (libre) ;
        libre↑.élément := x ;
        si (liste-vide) alors début tête := libre ;
                                    libre↑.suivant :=  nil ;
                              fin
         sinon début   tant que (p↑.élément < x) et (p↑.suivant ≠ nil) faire
                             début   q := p ;
                                     p := p↑.suivant;
                             fin ;
                            si (p↑.élément > x) alors début
                                                  libre↑.suivant := p ;
                                                  si p= tête alors tête := libre
                                                  sinon q↑.suivant := libre ;
                                                  fin
                             sinon début libre↑.suivant :=  p↑.suivant ;
                                         p↑.suivant := libre ;
                                   fin
                  fin
fin
```

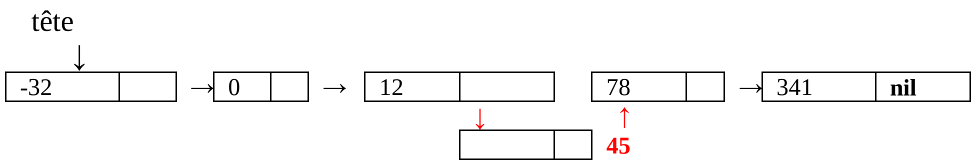
```
procédure suppression ;
(* suppression de x d'une liste non triée *)
entrée : tête : ↑liste ;  x : entier ;
sortie : tête : ↑liste ;

var p, q : ↑liste ;
début
        si (liste-vide) alors signaler ('liste vide')
        sinon début       p := tête ;
                             tant que (p↑.élément ≠ x) et (p↑.suivant ≠ nil) faire
                             début   q := p ;
                                     p := p↑.suivant;
                             fin ;
                  si (p↑.élément = x) alors
                       début   si p = tête alors tête := p↑.suivant
                                      sinon q↑.suivant :=  p↑.suivant ;
                               libérer (p) ;
                       fin  sinon signaler ('x n'existe pas dans la liste') ;
              fin
fin
```

Exemple 2.2

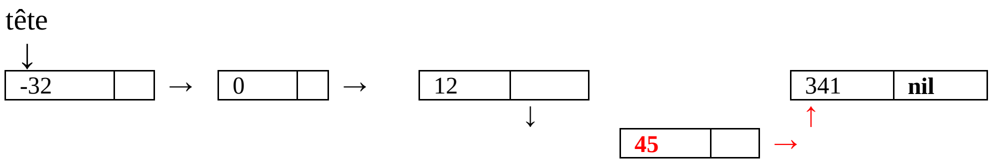
Si l'on considère la liste des 5 entiers {-32, 0, 12, 78, 341}, la structure schématique de la liste dynamique contenant ces entiers est la suivante :



L'insertion du nombre 45 engendre la configuration suivante de la liste :



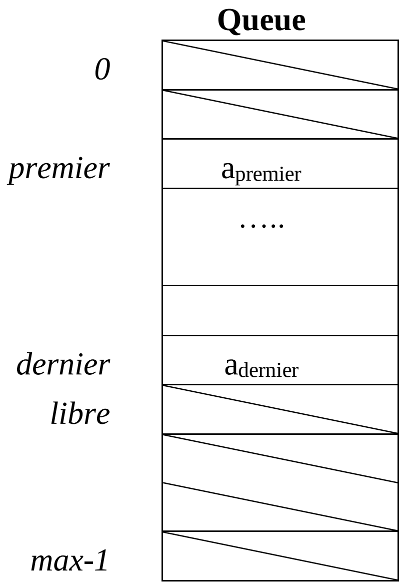
La suppression de 78 de cette nouvelle configuration donne la configuration suivante :



La complexité de chacune des procédures présentées est *O(n)* pour la même raison évoquée dans l'utilisation des tableaux statiques. Au pire cas, la liste est parcourue dans sa globalité et les *n* éléments sont visités.

Queues ou files d'attente

Représentation à l'aide de tableau et opérations élémentaires



Initialisation de la queue :
max := 200 ; premier := 0 ; dernier := 0 ; libre := 0 ;

procédure *queue-vide* ***;***
entrée *: queue : tableau [0..max-1] d'entier ;*
premier, dernier, libre : 0..max-1 ;
sortie *: vrai ou faux ;*

début ***si*** *(premier = libre) et (dernier = libre)*
alors retourner (vrai) sinon retourner (faux) *;*
fin

procédure *queue-pleine* ***;***
entrée *: queue : tableau [0..max-1] d'entier ;*
premier, dernier, libre :0..max-1 ;
sortie *: vrai ou faux ;*

début ***si*** *(libre = premier) et (dernier = (premier-1)* ***modulo*** *max)* ***alors***
retourner (vrai) sinon retourner (faux) *;*
fin

```
procédure recherche ;
entrée : queue : tableau [0..max-1] d'entier ;
         premier, dernier : 0..max-1 ;
         x : entier ;
sortie : position de x dans la queue ;

var      p : entier ;
         trouve : booléen ;

début si (non queue-vide) alors
         début   trouve := faux ;
                 p := premier ;
                 tant que (non trouve) et (p <= dernier) faire
                 si (queue[p] = x) alors trouve := vrai
                                   sinon  p :=( p+1)modulo max ;
                 si (trouve) alors retourner(p)
                             sinon signaler ('x n'existe pas dans la queue') ;
          fin sinon signaler ('queue vide')
fin
```

```
procédure enfiler ;
entrée : queue : tableau [0..max-1] d'entier ;
         premier, dernier, libre : 0..max-1 ;
         x : entier ;
sortie : queue, premier, dernier, libre;

début    si queue-pleine alors signaler ('queue pleine')
         sinon   début
                     si queue-vide alors premier := libre ;
                      queue [libre] := x ;
                     dernier := libre ;
                      libre := (libre +1) modulo max ;
                 fin
fin
```

```
procédure défiler;
entrée : queue : tableau [0..max-1] d'entier ;
          premier, dernier, libre : 0..max-1 ;
sortie : x, le premier élément de la queue ;

var x ;
début   si (queue-vide) alors  signaler ('queue vide')
                    sinon début x := queue[premier] ;
                                premier := (premier+1) modulo max ;
                                si (premier = libre) alors dernier := libre ;
                                retourner (x) ;
                          fin
fin
```

Exemple 2.3

Considérer la queue suivante constituée de 5 nombres entiers :

	Queue
premier= 0	-32
1	0
2	12
3	78
dernier= 4	341
libre=5	

L'insertion du nombre 45 engendre la configuration suivante de la queue :

	Queue
libre = premier = 0	-32
1	0
2	12
3	78
4	341
dernier= 5	45

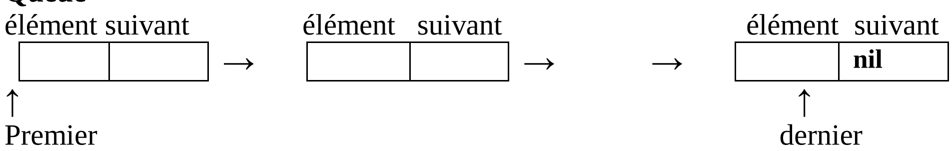
La suppression du premier élément de cette nouvelle configuration donne le tableau suivant :

	Queue
libre = 0	-32
premier = 1	0
2	12
3	78
4	341
dernier = 5	45

Représentation à l'aide de structure dynamique et opérations élémentaires

Les structures dynamiques peuvent être également utilisées pour représenter des queues. La figure suivante est une représentation schématique d'une queue à l'aide d'une liste dynamique.

Queue



La déclaration du type de queue est :
queue : ***enregistrement*** *premier, dernier : ↑liste* ***fin*** *;*

Initialisation de la queue :
queue.premier := ***nil*** *; queue.dernier :=* ***nil*** *;*

---

procédure *queue-vide* **;**
entrée *: queue :* ***enregistrement*** *premier, dernier : ↑liste* ***fin*** *;*
sortie *: vrai ou faux ;*

début ***si*** *(queue.premier =* ***nil****) et (queue.dernier =* ***nil****)* ***alors retourner*** *(****vrai****)*
sinon retourner *(****faux****) ;*
fin

```
procédure recherche ;
entrée : queue : enregistrement premier, dernier : ↑liste fin ; x : entier ;
sortie : position de x dans la queue ;

var       p : ↑liste ; trouve : booléen ;

début si (non queue-vide) alors
   début p := queue.premier ;
          trouve := faux ;
          tant que (non trouve) et (p ≠ nil) faire
             si ((p↑.élément=x) alors trouve := vrai sinon   p := p↑.suivant ;
         si trouve alors retourner(p) sinon signaler('x n'est pas dans la queue') ;
   fin sinon signaler ('queue vide')
fin
```

```
procédure enfiler ;
entrée : queue : enregistrement premier, dernier : ↑liste fin ; x : entier ;
sortie : queue : enregistrement premier, dernier : ↑liste fin ;

var p : ↑liste ;
début   nouveau (p) ;
        p↑.élément := x
        p↑.suivant := nil ;
        si (non queue-vide) alors queue.dernier↑.suivant := p
                           sinon début   queue.premier := p ;
                                         queue.dernier := p;
                                 fin
fin
```

```
procédure défiler ;
entrée : queue : enregistrement premier, dernier : ↑liste fin ;
sortie : le premier élément de la queue ;

var p:↑liste ;
début   si (non queue-vide) alors
            début p := queue.premier ;
                  queue.premier :=  queue.premier↑.suivant;
                  si (queue.premier = nil) alors queue.dernier := nil ;
                  retourner (p↑.élément) ;
                  libérer (p) ;
            fin
        sinon signaler ('queue vide') ;
fin
```

Remarquons que la notion de *queue-pleine* n'existe pas pour les structures dynamiques car on peut rajouter autant d'éléments dans la queue que l'on désire. Comme pour le cas d'utilisation de structure statique, la complexité de chacune de ces trois procédures est *O(n)*.

Exemple 2.4

Considérer la liste suivante constituée des 5 nombres entiers relatifs :

-32		→	0		→	12		→	78		→	341	**nil**

↑

↑

premier dernier

L'insertion du nombre 45 engendre la configuration suivante :

Premier dernier

↓ ↓

-32		→	0		→	12			78		→	341		→	**45**	**nil**

La suppression du premier élément de la queue de cette nouvelle configuration donne la configuration suivante :

x = -32

premier **dernier**

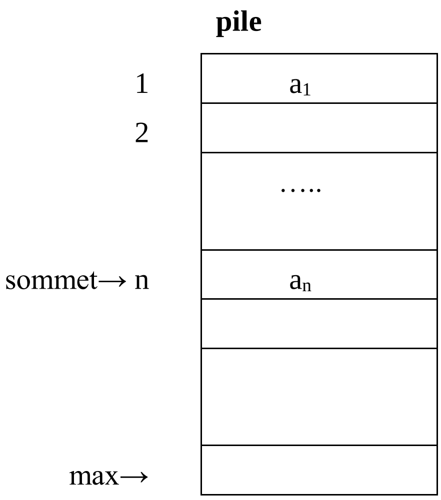
↓ ↓

0		→	12		→	78		→	341		→	45	**nil**

Piles

Représentation à l'aide de tableau et opérations élémentaires

La pile contenant les éléments a_1, a_2, …, a_n, est représentée à l'aide du tableau suivant :



Initialisation de la pile
max := 200 ; sommet := 0 ;

procédure *pile-vide*
entrée *: pile : tableau[1..max] d'****entier*** *; sommet :0..max;*
sortie *: vrai ou faux ;*

début
  si *(sommet = 0)* ***alors retourner (vrai) sinon retourner (faux)*** *;*
fin

procédure *pile-pleine*
entrée *: pile : tableau[1..max] d'****entier*** *; sommet :0..max;*
sortie *: vrai ou faux ;*

début
  si *(sommet = max)* ***alors retourner (vrai) sinon retourner (faux)*** *;*
fin

```
procédure empiler ;
entrée : pile : tableau[1..max] d'entier ; sommet :0 .. max; x : entier ;
sortie : pile : tableau[1..max] d'entier ; sommet :0 .. max;

début   si (non pile-pleine) alors
          début sommet := sommet + 1 ;
                pile[sommet] := x ;
          fin
        sinon signaler un débordement de la pile ;
fin
```

```
procédure dépiler ;
entrée : pile : tableau[1..max] d'entier ; sommet : 0 .. max;
sortie : l'élément qui se trouve au sommet de la pile ;

début   si (non pile-vide) alors
        début
              sommet := sommet-1 ;
              retourner (pile[sommet+1]);
        fin
        sinon signaler que la pile est vide
fin
```

Exemple 2.5
Considérer la pile suivante constituée des 5 nombres entiers relatifs et représentée par le tableau nommé *pile* suivant :

	pile
1	-32
2	0
3	12
4	78
Sommet= 5	341
max	

L'empilement du nombre 45 engendre la configuration suivante de la pile et le contenu suivant du tableau :

	pile
1	-32
2	0
3	12
4	78
5	341
sommet=6	45
max	

Le dépilement à partir de cette nouvelle configuration donne le tableau initial suivant :

	pile	
1	-32	
2	0	
3	12	
4	78	
sommet= 5	341	
	45	**x = 45**
max		

Ensembles

Représentation et opérations élémentaires

Une représentation possible d'un ensemble S, qui facilite l'écriture des opérations union et intersection entre deux ensembles, utilise deux tableaux TV (Tableau des Valeurs) et VR (Vecteur Représentatif) comme l'illustre l'exemple suivant :

Exemple 2.6 :

Soit S = {9, 11, 546, 0, 27, 47}, un ensemble de 6 entiers naturels. Les tableaux TV et VR se présentent comme suit :

31	5	9	87	11	546	0	27	8	47	71	-1	-1		-1	11
1	2	3	4	5	6	7	8	9	10	11					max

TV

0	0	1	0	1	1	1	1	0	1	0	0	0	0		6
1	2	3	4	5	6	7	8	9	10	11					max

VR

Le tableau TV contient toutes les valeurs susceptibles d'appartenir à un ensemble S. *TV[max]* contient le nombre de valeurs de TV. VR, le vecteur représentatif de S, est un vecteur de nombres booléens défini comme suit :

$$VR[i] = \begin{cases} 1 & si\ TV[i] \in S \\ 0 & sinon \end{cases}$$

VR[max] contient le nombre d'éléments de S. Les opérations qu'on peut mener sur les ensembles sont :

- Appartenance d'un élément à un ensemble.
- Union de deux ensembles.
- Intersection entre deux ensembles.

Ces trois opérations peuvent être implémentées comme suit :

***Procédure** appartenance ;*
***entrée** : TV : **tableau d'entiers** ; VR : **tableau de booléens** ; x : **entier** ;*
***sortie** : vrai ou faux ;*

***var** p : **entier** ;*
***début** p :=1;*
    ***tant que** (p <= TV[max]) **et** (TV[p] ≠ x) **faire** p := p+1 ;*
    si** (TV[p] = x) **et** (VR[p]=1) **alors retourner(vrai)
        ***sinon retourner(faux)** ;*
fin

***procédure** union ;*
***entrée** : TV : **tableau d'entiers** ; VR_1, VR_2 : **tableau de booléens** ;*
***sortie** : VR_3 : **tableau de booléens**;*

***var** i : 1..max ;*
début
    pour** (i := 1 **à** TV[max]) **faire
    *$VR_3[i]$:= $VR_1[i]$ **ou** $VR_2[i]$;*
    (**ou** est l'opérateur booléen de disjonction *)*
fin ;

---

procédure *intersection ;*
entrée *: TV :* ***tableau d'entiers*** *; VR_1, VR_2 :* **tableau de booléens** *;*
sortie *: VR_3 :* ***tableau de booléens;***

var *i : 1 ..max ;*
début
    pour *(i := 1* ***à*** *TV[max])* ***faire***
    VR_3[i] := VR_1[i] ***et*** *VR_2[i] ;*
    *(** ***et*** *est l'opérateur booléen de conjonction *)*
fin ;

La complexité de la procédure appartenance est *O*(*TV[max])* car au pire cas, *x* n'appartient pas à TV et la boucle aura un nombre d'itérations égal à *TV[max]*. Les procédures *union* et *intersection* ont chacune une complexité *O*(*TV[max])* car la boucle *pour* a un nombre d'itérations égal à *TV[max]*.