Les listes linéaires bidirectionnelles

3ème GI & TR

Sakka Rouis Taoufik & Ben Salah Kais

Les listes linéaires bidirectionnelles

1- Introduction

- ✓ Dans les listes vues précédemment le parcours ne peut se faire que dans un seul sens : de la tête vers la queue.
- ✓ Pour remédier à cette dissymétrie de l'opération et permettre un parcours aussi bien dans un sens que dans l'autre, on peut construire des listes linéaires bidirectionnelles (ou listes bilatères)

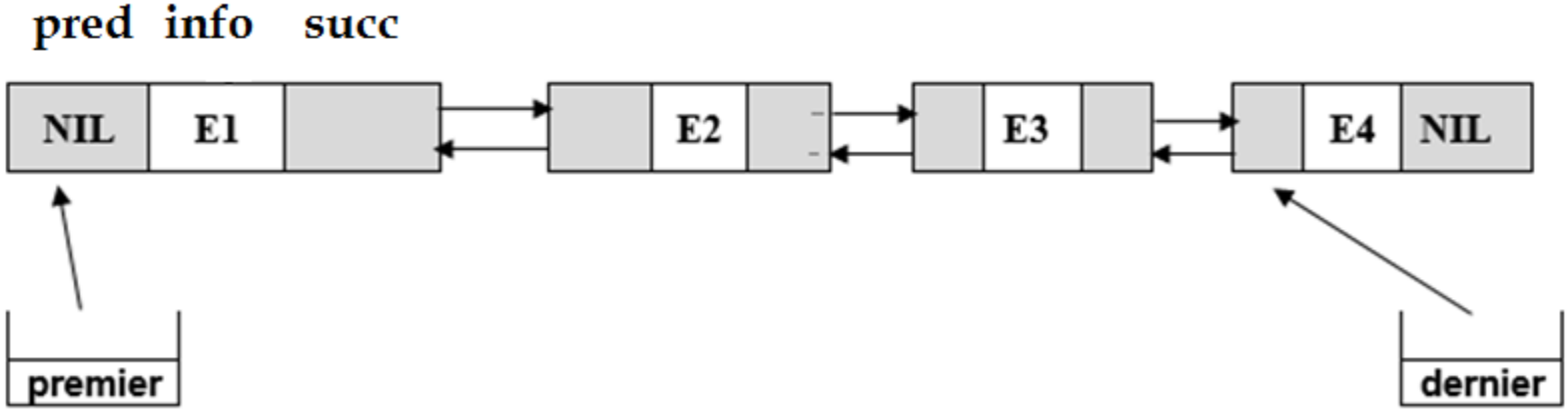
2- Définition

- ✓ Une liste linéaire bidirectionnelle (LLB) est une structure de données permettant de regrouper une suite d'éléments de même type et dont chacun possède deux liens :
  - un lien pour accéder à l'élément successeur
  - et un lien pour atteindre l'élément précédent.

Les listes linéaires bidirectionnelles

- ✓ A chaque élément de la liste est associé, en mémoire, un emplacement mémoire représentant une structure à trois champs
  - le premier contient l'adresse de l'emplacement mémoire de l'élément précédent,
  - le deuxième contient l'élément
  - et le troisième contient l'adresse de l'emplacement mémoire de l'élément successeur.
- ✓ Le premier et le dernier champs sont des pointeurs sur des structures de même type.

Exemple



Les listes linéaires bidirectionnelles

3- Caractéristiques d'une liste doublement chaînée

- ✓ Une liste linéaire bidirectionnelle est utilisée pour stocker des données qui doivent être traitées de manière séquentielle
- ✓ Les éléments de la liste appelés cellule, nœuds ou maillons, ne sont pas rangés les uns à côté des autres.
- ✓ On a besoin de connaître, pour chaque élément, l'adresse de l'élément précédent et l'adresse de l'élément successeur.
- ✓ On dispose une structure qui contient :
  - un pointeur « **premier** » qui contient l'adresse du premier élément de la liste
  - un pointeur « **dernier** » qui contient l'adresse du dernier élément de la liste
- ✓ Si **premier** = **dernier** = NIL → La liste est vide
- ✓ Si **premier** = **dernier** ≠ NIL → La liste à un seul élément

Les listes linéaires bidirectionnelles

4- Matérialisation d'une LLB à 2 points d'entrée

Syntaxe :

```
Nom_Cellule = structure
        // Nom_Pointeur sur l ' élément précédent : ^ Nom_Cellule
        // Déclaration des champs de la cellule
        //  Nom_Pointeur  sur l ' élément successeur     : ^ Nom_Cellule
Fin structure

Nom_Liste = Structure
      // Nom_Pointeur_premier     : ^ Nom_Cellule
      //  Nom_Pointeur_dernier     : ^ Nom_Cellule
Fin structure
```

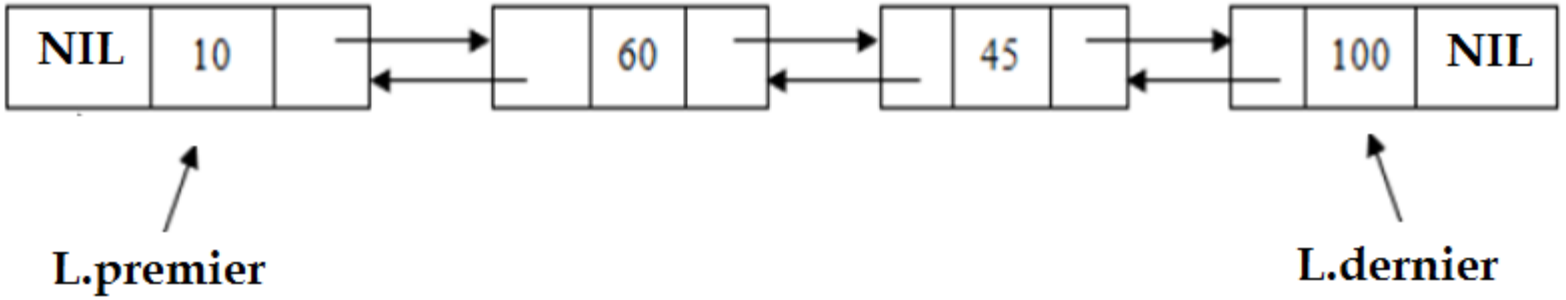
Les listes linéaires bidirectionnelles

Exemple :

En supposant que les éléments de la liste sont des entiers, celle-ci se définie de la façon suivante :

```
cellule = structure
            pred    : ^cellule
            info    :  entier
            succ    : ^cellule
 Fin structure
 liste = structure
            premier : ^cellule
            dernier  : ^cellule
Fin structure

L : liste
```



Les listes linéaires bidirectionnelles

5-Les opérations sur les listes linéaires bidirectionnelles

5-1-Création d'une liste chaînée bidirectionnelle

```
Procédure creer_liste (VAR L : Liste)
DÉBUT
    L.premier ← NIL
    L.dernier ← NIL
FIN
```

5-2-Tester la liste est vide ou pas

```
Fonction tester_liste_vide (L : Liste):booléen
DÉBUT
     retourner  L.premier = NIL
FIN
```

Les listes linéaires bidirectionnelles

5-3-Ajout d'un élément à une liste chaînée bidirectionnelles

- On distingue quatre types d'insertions :

  - insérer avant premier
  - insérer après dernier
  - insérer après un élément référencé
  - insérer avant un élément référencé

Les listes linéaires bidirectionnelles

A- /****Insertion avant le premier élément*******/**

```
Procédure inserer_avant_premier (VAR L : Liste , x : entier)
 VAR
        p : ^cellule
DÉBUT
            Allouer(p)
            p^.info ← x
            p^. succ ← L . premier
            p^.pred ← NIL
            Si (L.premier = NIL) Alors
                        L.dernier ← p
            Si non
                        (L.premier)^.pred ← p
            Fin Si
            L.premier ← p
FIN
```

Les listes linéaires bidirectionnelles

B- /******Insertion après le dernier élément*******/

```
Procédure inserer_ apres_dernier (VAR L : Liste , x : entier)
 VAR
        p : ^cellule
DÉBUT
            Allouer(p)
            p^.info ← x
            p^.pred ← L.dernier
            p^.succ ← NIL
             Si (L.dernier = NIL)
                        L.premier ← p
            Si non
                        (L.dernier)^.succ ← p
            Fin Si
            L.dernier ← p
FIN
```

Les listes linéaires bidirectionnelles

C- /******Insertion après élément référencé *******/

```
Procédure inserer_ apres_reference (var p :^ cellule, x : entier)
 VAR
        q : ^cellule
DÉBUT
        Allouer(q)
        q^.info ← x
        q^.succ ← p^.succ
        q^.pred ← p
        (p^.succ)^.pred ← q
        p^.succ ← q
FIN
```

Les listes linéaires bidirectionnelles

D- /******Insertion avant élément référencé *******/

```
Procédure inserer_ avant_reference (var p :^ cellule, x : entier)
 VAR
        q : ^cellule
DÉBUT
        Allouer(q)
        q^.info ← x
        q^.succ ← p
        q^.pred ← p^.pred
        (p^.pred)^.succ ← q
        p^.pred ← q
FIN
```

Les listes linéaires bidirectionnelles

5-4- Parcours d'une lise chaînée

```
Procédure affiche(L : lste)
 VAR
        p : ^cellule
DÉBUT
   Si (tester_liste_vide(L)) Alors
        Écrire ("Liste vide" )
   Si non
        Si (L.premier = L.dernier) Alors
            Écrire ("La liste à un seul élément:" ,L.premier^.info)
        Si non
            p ← L.premier   /* p←L.dernier : de droite à gauche */
            Tant que (p ≠ NIL) Faire
                Écrire ( p^.info)
                p ← p^.succ /* ← p^.pred  : de droite à gauche */

            Fin Tant que
        Fin Si
   Fin si
FIN
```

Les listes linéaires bidirectionnelles

5-5- Suppression d'un élément de la liste

On distingue cinq types de suppression:

- supprimer le premier
- supprimer le dernier
- supprimer un élément référencé
- supprimer l'élément après un élément référencé
- supprimer l'élément avant un élément référencé

Les listes linéaires bidirectionnelles

A- /******Suppression du premier élément*******/

```
Procédure supprimer_premier (VAR L : Liste)
 VAR
        p : ^cellule
DÉBUT
        Si (tester_liste_vide(L)) Alors
                Écrire ("Liste vide" )
        Si non
                Si (L.premier = L.dernier)
                        Libérer(L.premier)
                        L.premier ← NIL
                        L.dernier ← NIL
                Si non
                        p ← L.premier
                        (p^.succ)^.pred ← NIL
                        L.premier ← p^.succ
                        Libérer(p)
                Fin Si
        Fin Si
FIN
```

Les listes linéaires bidirectionnelles

B- /******Suppression du dernier élément*******/

```
Procédure supprimer_dernier (VAR L : Liste)
 VAR
        p : ^cellule
DÉBUT
        Si (tester_liste_vide(L)) Alors
                Écrire ("Liste vide" )
        Si non
                Si (L.premier = L.dernier)
                        Libérer(L.premier)
                        L.premier ← NIL
                        L.dernier ← NIL
                Si non
                        p ← L.dernier
                        (p^.pred)^.succ ← NIL
                        L.dernier ← p^.pred
                        Libérer(p)
                Fin Si
        Fin Si
FIN
```

Les listes linéaires bidirectionnelles

C- /******supprimer un élément référencé*******/

```
Procédure supprimer_element_reference (var p : ^cellule)
DÉBUT
        (p^.pred)^.succ ← p^.succ
        (p^.succ)^.pred ← p^.pred
        Libérer (p)
FIN
```

Les listes linéaires bidirectionnelles

D- /****** Suppression après élément référencé *******/

```
Procédure supprimer_apres_reference (var p : ^cellule)
VAR
        q : ^cellule
DÉBUT
        q ← p^.succ
        p^.succ ← q^.succ
        (q^.succ)^.pred ← p
        Libérer (q)
FIN
```

Les listes linéaires bidirectionnelles

E- /****** Suppression avant élément référencé *******/

```
Procédure supprimer_avant_reference (var p : ^cellule)
VAR
        q : ^cellule
DÉBUT
        q ← p^.pred
        p^.pred ← q^.pred
        (q^.pred)^.succ ← p
        Libérer (q)
FIN
```