Cours: ASD 2

Chapitre 6: Les Listes Linéaires

Réalisé par:
Sakka Rouis Taoufik

1

Ch1: Les Listes Linéaires

I. Introduction

- Une liste linéaire (LL) est la représentation informatique d'un ensemble fini, de taille variable et éventuellement nul, d'éléments de type T. Un tel ensemble est ordonnée.

- Mathématiquement, on peut présenter une LL en énumérant ses éléments dans l'ordre.

- L'accès aux éléments d'une liste se fait de manière séquentielle : chaque élément permet l'accès au suivant (contrairement au tableau dans lequel l'accès se fait de manière directe, par adressage de chaque cellule dudit tableau).

2

Ch1: Les Listes Linéaires

II. Opérations sur la SD LL

2.1 Les opérations élémentaires

- **creerliste** : permettant de créer une liste linéaire vide.
- **listevide** : permettant de voir si la liste linéaire est vide ou non ?
- **ajouter** ou opération d'**adjonction : -** en tête : avant le premier
  - en queue : après le dernier
  - quelque part au milieu
- **supprimer** un élément de la liste : - premier élément
  - dernier élément
  - quelque part au milieu
- **recherche** : permettant de voir si un élément appartient dans une liste linéaire. Le point de départ peut être fourni comme paramètre.
- **Visiter** : permettant de visiter tous les éléments de la SD LL en effectuant pour chaque élément visité une action donnée (paramètre). Cette action est connue sous le nom de **traversée**.

3

Ch1: Les Listes Linéaires

II. Opérations sur la SD LL

2.2 Les opérations élaborés

- **Inversion** permettant d'inverser une liste linéaire

ll=(a, b, c, d) ll_{inv}=(d, c, b, a)

- **supprimer_tous** : permet de supprimer tous les éléments d'une liste donnée.

- **concaténation** permet de concaténer deux listes données.

ll_1=(a, b, c, d)

ll_2=(x, y, z, w, k)

La concaténation de ll1 et ll2 dans l'ordre (ordre significatif) donne :

ll=(a, b, c, d, x, y, z, w, k)

4

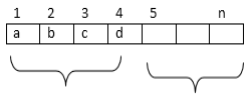
Ch1: Les Listes Linéaires

III. Représentation physique

3.1 Représentation contiguë

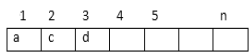
- **Illustration :** ll=(a, b, c, d)

représentation abstraite



- **suppression** : elle exige des décalages à gauche pour récupérer la position devenue disponible.

  Exemple : supprimer b



5

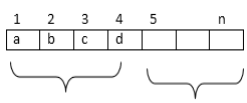
Ch1: Les Listes Linéaires

III. Représentation physique

3.1 Représentation contiguë

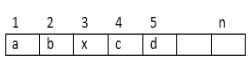
- **Illustration :** ll=(a, b, c, d)

représentation abstraite



- **visiter** : balayer tous les éléments d'un tableau dans la partie garnie
- **recherche** ➔ recherche dans un tableau, on fait appel aux algorithmes connus soit recherche séquentielle soit dichotomique
- **adjonction** : elle exige des décalages à droite.

  Exemple : ajouter x après b.



6

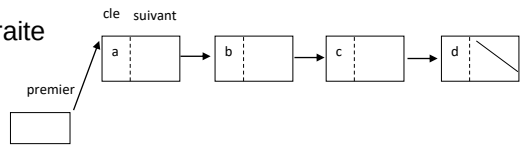
Ch1: Les Listes Linéaires

III. Représentation physique

3.2 Représentation chaînée

- **Illustration :** ll=(a, b, c, d)

représentation abstraite



- L'adjonction dans une liste linéaire (LL) concrétisée à l'aide d'une représentation chaînée n'exige pas de **décalages** contrairement à la représentation contiguë.

➔ On peut dire que la représentation contiguë de la SDLL n'est pas recommandée à cause des décalages impliqués par les deux opérations fondamentales ajouter et supprimer notamment pour les listes linéaires de taille plus au moins importante.

7

Ch1: Les Listes Linéaires

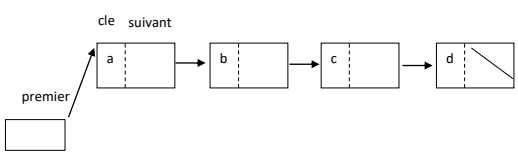
IV. Variantes de la SD liste linéaire

4.1 LL uni directionnelle avec un seul point d'entrée

- **Illustration :** ll=(a, b, c, d)

Représentation abstraite

Variante 1:



8

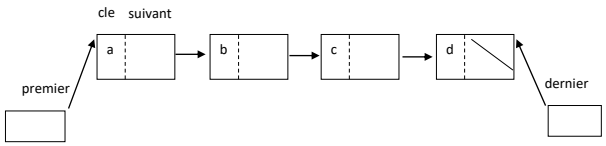
Ch1: Les Listes Linéaires

IV. Variantes de la SD liste linéaire
4.2 LL uni directionnelle avec deux points d'entrée

- **Illustration :** ll=(a, b, c, d)

Représentation abstraite

Variante 2:



- Pour les deux variantes (1) et (2), la liste linéaire **est unidirectionnelle**. À partir d'un élément donné on peut passer à son successeur. Ceci est possible grâce au champ de chaînage suivant : dans la variante (1), le premier élément est privilégié (accès direct) dans la variante (2) le premier et le dernier élément sont privilégiés (accès directe).

9

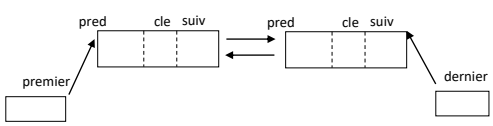
Ch1: Les Listes Linéaires

IV. Variantes de la SD liste linéaire
4.3 LL bidirectionnelle avec 2 points d'entrée

- **Illustration :** ll=(a, b, c, d)

Représentation abstraite

Variante 3:



- À partir d'un élément donné, on peut passer soit à son successeur soit à son prédécesseur.

10

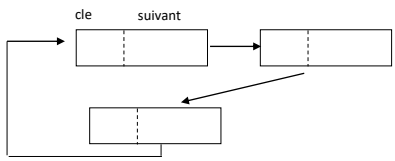
IV. Variantes de la SD liste linéaire

4.4 LL Circulaire ou Anneau

- **Illustration :** ll=(a, b, c, d)

Représentation abstraite

Variante 4:



- Les notions de premier et dernier disparaissent, c'est-à-dire ces notions n'ont pas de sens dans un anneau. Un anneau est doté uniquement d'un point d'entrée quelconque.

11

V. Matérialisation de liste chainée à 2 points d'entrée

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

```
Types
Cellule = Struct
        cle : entier
        Suiv : ^Cellule
FinStruct

Liste = Struct
        premier: ^Cellule
        dernier : ^Cellule
FinStruct
```

12

Ch1: Les Listes Linéaires

V. Matérialisation de liste chainée à 2 points d'entrée

V.1 Création d'une liste chaînée

Solution 1 : sous forme d'une procédure	Solution 2 : sous forme d'une fonction
Procédure creer_liste (Var L : Liste) Début L.premier←Nil L.dernier←Nil Fin proc	Function creer_liste (): Liste Var L1 : Liste Début L1.premier←Nil L1.dernier←Nil creer_liste←L1 fin Fn

13

Ch1: Les Listes Linéaires

V. Matérialisation de liste chainée à 2 points d'entrée

V.2 Tester la vacuité d'une liste linéaire

Solution 1	Solution 2
Fonction liste_vide (ll : Liste) : boolean Debut Si (ll.premier=Nil) alors liste_vide← vrai Sinon liste_vide← faux finsi fin Fn	Fonction liste_vide (ll :Liste) : boolean debut Si ((ll.premier=Nil) et (ll.dernier=Nil)) alors liste_vide← vrai Sinon liste_vide← faux finsi fin Fn

La solution (2) est cohérente par rapport à la réalisation de créer liste

14

V. Matérialisation de liste chainée à 2 points d'entrée
V.3 Processus d'adjonction ou d'insertion

- On distingue quatre types d'insertions :
  - insérer après un élément référencée
  - insérer avant un élément référencée
  - insérer avant premier
  - insérer après dernier

V. Matérialisation de liste chainée à 2 points d'entrées
V.3 Processus d'adjonction ou d'insertion

```
/*insertion après élément référencée*/
procedure inserer_apres (info:entier ; var p:^Cellule)
   var
          q:^Cellule
  debut
          Allouer(q)
          q^.cle←info
          q^.suiv←p^.suiv
          /*mise à jour du successeur de p*/
          p^.suiv←q
fin proc
```

V. Matérialisation de liste chainée à 2 points d'entrée
V.3 Processus d'adjonction ou d'insertion

```
/*insertion avant un élément référencé*/
procedure inserer_avant (info:entier; var p:^Cellule)
/* le problème : la liste est unidirectionnelle à partir de p,
                        on ne peut pas passer à son prédécesseur */
var
    q:^Cellule
debut
    Allouer(q)
   q^←p^
   /*mise à jour l'espace référencé par p*/
   p^.cle←info
   p^.suiv←q
Fin proc
```

17

V. Matérialisation de liste chainée à 2 points d'entrée
V.3 Processus d'adjonction ou d'insertion

```
/*insertion avant le premier élément*/

procedure inserer_avant_premier (info:entier; var ll:Liste)
var
          q:^Cellule
debut
   si (liste_vide (ll)) alors
       Allouer(q)
       q^.cle←info
       q^.suiv←Nil
       ll.premier←q
       ll.dernier←q
   sinon
                Allouer(q)
                q^.cle←info
                q^.suiv←ll.premier
                ll.premier←q
    finsi
fin proc
```

18

V. Matérialisation de liste chainée à 2 points d'entrée
V.3 Processus d'adjonction ou d'insertion

```
/*insertion après le dernier*/

procedure inserer_apres_dernier (info:entier ; var ll:Liste)
var
    q:^Cellule
debut
  si (liste_vide (ll)) alors
      Allouer(q)
      q^.cle←info
      q^.suiv←Nil

      ll.premier←q
      ll.dernier←q
  sinon
      Allouer(q)
      q^.cle←info
      q^.suiv←Nil
      (ll.dernier)^.suiv←q
      ll.dernier←q
  finsi
fin proc
```


V. Matérialisation de liste chainée à 2 points d'entrée
V.4 Parcours d'une lise chaînée

- La procédure itérative suivante permet de parcourir et afficher les éléments d'une liste chaînée.

```
Procédure AffichListe_itér (ll : Liste)
Var
   P : ^Cellule
Début
   p ←ll.premier
  TantQue (P # Nil) Faire
          Ecrire(P^.cle)
          P← P^.Suiv
    FinTQ
Fin Proc
```

V. Matérialisation de liste chainée à 2 points d'entrée
V.4 Recherche d'un élément dans une lise chaînée

```
Fonction recherche (x : Entier ; ll: Liste) : Booléen
Var
  P :^Cellule
  Trouve : Booléen
Début
  Trouve←Faux
  P ←ll.premier
  TantQue (P # Nil) ET (Trouve = Faux) Faire
    Trouve←(P^.cle = x)
    P←P^.suiv
  FinTQ
   recherche ←Trouve
Fin Fn
```


V. Matérialisation de liste chainée à 2 points d'entrée
V.5 Processes de suppression

- On distingue trois types de suppression:
  - supprimer un élément référencée,
  - supprimer le premier
  - supprimer le dernier

```
/* suppression d'un élément référencée */

procedure supprimer_elem ( var p:^Cellule)
var
   q :^Cellule
debut
   /* on suppose que p admet un successeur*/
   Assure (p^.suiv != Nil)
  /* la méthode assert de la bibliothèque assert.h en C */
  q←p^.suiv
  p^←q^
  liberer (q)
fin proc
```



```
/* suppression du premier élément */

procedure supprimer_premier (var ll :Liste)
var
     q :^Cellule
debut
     q←ll.premier
     ll.premier←q^.suiv
     liberer (q)
     si (ll.premier= Nil) alors
        ll.dernier←Nil
     finSi
fin proc
```

V. Matérialisation de liste chainée à 2 points d'entrée

V.5 Processes de suppression

```
/* suppression du dernier élément */

procedure supprimer_dernier (var ll :Liste)
var
    q :^Cellule
debut
    si (ll.premier = ll.dernier) alors
        suppressimer_premier (ll)
    sinon
        q←ll.premier
        TantQue (q^.suiv != ll.dernier)
            q←q^.suiv
        FinTQ
        q^.suiv← Nil
        liberer (ll.dernier)
        ll.dernier←q
    finsi
fin proc
```


VI. Exercices d'application

Exercice 1:
Implémenter la SD LL en adaptant une représentation chainée dotée d'un seul point d'entrée

Exercice 2 : Inversion d'une liste chaînée
Écrire une procédure qui permet d'inverser une liste chaînée Ls dans une liste Ls_Inv.

Exercice 3 : Insertion dans une liste chaînée
On suppose qu'on a en entrée une liste triée par ordre croissant d'entiers. Ecrivez l'action qui ajoute un entier dans la bonne position de façon que la liste reste encore triée après l'insertion.

VI. Exercices d'application

Exercice 4 : Tri par fusion d'une liste chaînée

Proposer un algorithme qui fait la concaténation de deux listes (en entrée) retournant une liste (en sortie). Par exemple, si l'on a les listes (3, 7, 2) et (5, 4) on obtient la liste (3, 7, 2, 5, 4).

Proposer ensuite un algorithme séparant une liste en deux. Etant donné une liste en entrée, un élément sur deux va dans la première liste et un élément sur deux va dans la deuxième liste.

Faire une action de fusion de listes supposées triées par ordre croissant en une troisième liste triée.

Faire, en utilisant au choix les trois actions précédentes une action de tri fusion qui étant donné deux listes en entrée (pas forcément triées) retourne une liste triée.