

LES LISTES CHAINEES

Introduction

Les listes chaînées sont composées d'une chaîne d'enregistrements pointeurs. Les pointeurs sont chaînés soit en simple soit en double.

Pour le chaînage simple, il y a 2 possibilités :

- La liste monodirectionnelle
- La liste circulaire

Pour le chaînage double, nous avons :

- La liste bidirectionnelle.

I. Les listes Monodirectionnelles :

1. Définition :

Une liste monodirectionnelle est composée d'un ensemble d'enregistrement pointeur chaîné en simple les uns aux autres. Chaque enregistrement pointeur de la liste monodirectionnelle dispose d'un ou de plusieurs champs informations et exactement d'un champ pointeur qui contient l'adresse de l'élément suivant. L'adresse du premier élément de la liste se trouve dans la variable **tete**. Le champ d'adresse du dernier élément est égal à **NIL**.

2. Déclaration :

```
Type NomListeMono = ↑structure
DEBUT
    Info1 : type1
    .
    .
    .
    InfoN : typeN
    NomPointeurSuivant : NomListeMono
FIN
Var tete : NomListeMono
```

Exemple 1 :

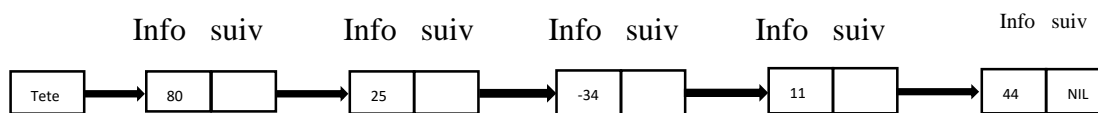
Définir la structure d'une liste monodirectionnelle de personnes puis déclarer la variable Tête à l'associer. Personne (Nom,Prenom,Age).

Exemple 2 :

Définir la structure de la liste monodirectionnelle d'employés puis déclarer la variable tête à l'associer. EMPLOYE (Mat, nom, prénom, service et date d'embauche (jour-mois-année)).

3. Manipulation de la liste Monodirectionnelle :

Soit la liste monodirectionnelle suivante :



- 1) Tete contient l'adresse du premier élément de la liste.
- 2) Tete↑.info = 80
- 3) Tete↑.suiv = L'adresse du 2^e élément
- 4) Tete↑.suiv↑.info = 25

4. Création d'une liste monodirectionnelle :

Pour créer une liste monodirectionnelle, il faut trois (3) pointeurs : **tete**, **pp** et **pc**.

- « Tete » : permet de sauvegarder l'adresse du premier élément de la liste. □
- « pc » : permet de créer tous les éléments à placer dans la liste.
- « pp » va jouer 2 rôles : un rôle de marquage du dernier élément et un rôle de chainage.

Exercice d'Application :

Écrire un sous-programme qui permet de créer une liste monodirectionnelle d'entiers. Les valeurs de cette liste seront saisies par l'utilisateur en fonction de ses besoins.

5. Parcours d'une liste monodirectionnelle

Pour parcourir une liste monodirectionnelle, il faut déclarer un pointeur de parcours à initialiser à **tete** de la liste à parcourir. Ce pointeur sera associé à une boucle. La boucle peut être « repeter » ou « tantque ».

Exercice D'application :

Écrire un sous-programme qui reçoit une liste monodirectionnelle d'entiers puis affiche les nombres positifs de la liste.

6. Les opérations applicables aux listes

Il est possible d'appliquer plusieurs types de traitements aux valeurs d'une liste. Les traitements de base sont en fonction des types de valeur des éléments de la liste mais également il est possible de faire la recherche de valeur, l'ajout, la modification, la suppression, l'intersection, la fusion etc.

Exercice d'application :

Écrire un programme qui reçoit une liste monodirectionnelle d'entiers puis transfère les nombres carrés dans une nouvelle liste monodirectionnelle.

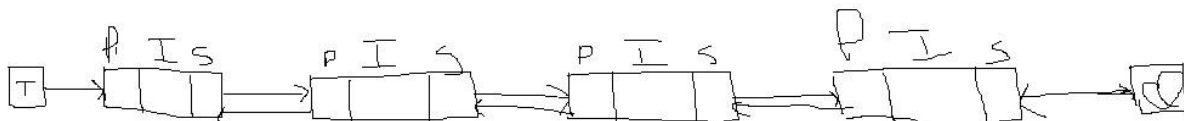
II. Les listes bidirectionnelles

1. Définition :

Une liste bidirectionnelle est une liste doublement chaînées avec deux (2) voies d'accès : Tête et Queue.

Tête : contient l'adresse du premier élément de la liste.

Queue : contient l'adresse du dernier élément de la liste.



SCHEMA D'UNE LISTE BIDIRECTIONNELLE

2. Déclaration

Syntaxe :

```
Type ListeBi = ↑structure  
DEBUT  
    Info : type  
    Suiv , prec : ListeBi  
FIN  
Var tete, queue : ListeBi
```

Exemple 1 : Déclaration d'une liste Bidirectionnelle d'entiers

```
Type LB = ↑structure  
DEBUT  
    Info : entier  
    Suiv, prec : LB  
FIN  
Var tete, queue : LB
```

Exemple 2 : Une liste bidirectionnelle de personnes

```
Type PERSONNE = structure  
DEBUT  
    Nom, prenom : chaine  
    Age : entier  
Fin  
Type LB = ↑structure  
DEBUT  
    Info : PERSONNE  
    Suiv, prec : LB  
FIN  
Var T, Q : LB
```

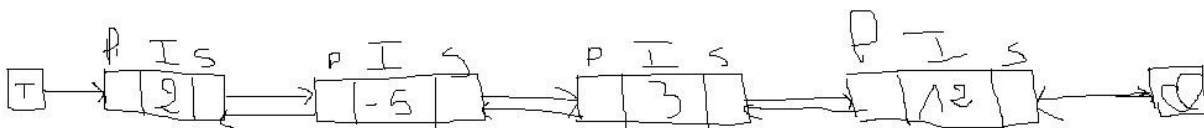
3. Manipulation d'une liste bidirectionnelle :

La liste bidirectionnelle est vide si Tête est égale à queue et égale à Nil.

Si Tête est égale à queue et que Tête est différente de Nil, alors la liste contient un seul élément.

Si Tête est différente de Queue alors, la liste a au moins deux (2) éléments.

Une liste bidirectionnelle peut être parcourue du début à la fin ou de la fin au début ou simultanément dans les 2 sens. Soit la liste d'entiers suivants :



a. Parcours du début à la fin :

Pour ce parcours, il faut utiliser le pointeur « suivant » en commençant par tête.

$T \uparrow = \text{impossible}$

$T \uparrow . \text{suiv} \uparrow . \text{suiv} = \text{l'adresse du 3}^{\text{e}} \text{ élément}$

$T = \text{l'adresse du 1}^{\text{er}} \text{ élément}$

$T \uparrow . \text{suiv} \uparrow . \text{suiv} \uparrow . \text{prec} = \text{l'adresse du 2}^{\text{e}} \text{ élément}$

$T \uparrow . \text{prec} = \text{Nil}$

$T \uparrow . \text{suiv} \uparrow . \text{suiv} \uparrow . \text{info} = 3$

$T \uparrow . \text{info} = 2$

$T \uparrow . \text{suiv} \uparrow . \text{suiv} \uparrow . \text{suiv} = \text{l'adresse du 4}^{\text{e}} \text{ élément}$

$T \uparrow . \text{suiv} = \text{l'adresse du 2}^{\text{e}} \text{ élément}$

$T \uparrow . \text{suiv} \uparrow . \text{suiv} \uparrow . \text{suiv} \uparrow . \text{suiv} = \text{Nil}$

$T \uparrow . \text{suiv} \uparrow . \text{prec} = \text{l'adresse du premier élément}$

$T \uparrow . \text{suiv} \uparrow . \text{suiv} \uparrow . \text{suiv} \uparrow . \text{suiv} \uparrow = \text{impossible}$

$T \uparrow . \text{suiv} \uparrow . \text{info} = -5$

b. Parcours de la fin au début

$Q = \text{l'adresse du dernier élément}$

$Q \uparrow . \text{prec} = \text{l'adresse du 3}^{\text{e}} \text{ élément}$

$Q \uparrow . \text{info} = 12$

$Q \uparrow . \text{suiv} = \text{Nil}$

$Q \uparrow . \text{prec} \uparrow . \text{suiv} = \text{l'adresse du 4}^{\text{e}} \text{ élément}$

$Q \uparrow . \text{prec} \uparrow . \text{info} = 3$

$Q \uparrow . \text{prec} \uparrow . \text{prec} = \text{l'adresse du 2}^{\text{e}} \text{ élément}$

$Q \uparrow . \text{prec} \uparrow . \text{prec} \uparrow . \text{suiv} = \text{l'adresse du 3}^{\text{e}} \text{ élément}$
 $Q \uparrow . \text{prec} \uparrow . \text{prec} \uparrow . \text{info} = -5$

$Q \uparrow . \text{prec} \uparrow . \text{prec} \uparrow . \text{prec} = \text{l'adresse du 1}^{\text{er}} \text{ élément}$

$Q \uparrow . \text{prec} \uparrow . \text{prec} \uparrow . \text{prec} \uparrow . \text{prec} = \text{NIL}$

Pour ce parcours, il faut commencer par queue en utilisant le champ pointeur « précédent ».

NB : Pour le parcours simultané, il faut une boucle associée à deux (2) indices.

4. Création d'une liste bidirectionnelle :

Pour créer une liste Bidirectionnelle, il faut trois (3) pointeurs : tête, queue et PC.

Tête : permet de sauvegarder l'adresse du premier élément de la liste.

Queue : permet de sauvegarder l'adresse du dernier élément de la liste.

PC : permet de créer chaque enregistrement pointeur à mettre dans la liste.

La variable « queue » sera utilisée pour le marquage et le chainage. Les pointeurs sont chaînés en double.

Exercice d'application :

Soit une liste monodirectionnelle de politiciens. Écrire un sous-programme qui transfère dans une liste bidirectionnelle tous les politiciens qui sont députés analphabètes. Un politicien est caractérisé par son id, son nom, son prénom, son nom de parti, sa fonction, est analphabète soit égale à « OUI » ou « NON ».

Exercice d'application 2:

Écrire un sous-programme qui reçoit une matrice d'entiers puis transfère les nombres divisibles par 5 dans une liste Bidirectionnelle.

5. Les traitements applicables aux listes bidirectionnelles :

Il est possible d'appliquer plusieurs types de traitements aux valeurs d'une liste Bi. Pour la plupart des traitements, la liste doit être parcourue, soit du début à la fin, soit de la fin au début ou simultanément dans les deux sens.

Exercice d'application :

Écrire un sous-programme qui reçoit une liste bidirectionnelle de politiciens. Écrire un sous-programme qui inverse cette liste sans utiliser d'autres structures.

III. Les lites circulaires

1. Définition

Une liste circulaire a la même structure qu'une liste monodirectionnelle.

La seule particularité est que le champ pointeur du dernier élément contient l'adresse du premier élément de la liste. Il existe deux (2) variantes de listes circulaires : LISTE CIRCULAIRE AVEC TETE (LCT) ou LISTE CIRCULAIRE AVEC QUEUE (LCQ).

2. Syntaxe de déclaration :

a. Avec tête

```
Type LCT = ↑structure
DEBUT
    Info : type
    Suiv : LCT
FIN
Var tete : LCT
```

b. Avec queue

```
Type LCQ = ↑structure
DEBUT
    Info : type
    Suiv : LCQ
FIN
Var queue : LCQ
```

Exemple : Liste circulaire d'entiers avec tete

Type LCT = ↑structure

DEBUT

Info : entier

Suiv : LCT

FIN

Var tete : LCT

3. Manipulation de listes circulaires

Pour manipuler une liste circulaire, il faut utiliser un champ pointeur de parcours qui sera initialisé soit à TETE ou soit à QUEUE. Ce pointeur sera associé à une boucle « REPETER » afin de parcourir tous les éléments de la liste.

La liste circulaire est vide si TETE= Nil ou QUEUE = NIL.

La liste circulaire a un élément si TETE != NIL et TETE↑.suiv = NIL.

NB : Le pointeur de parcours peut être initialisé à tete ou queue et la condition d'arrêt de la boucle « répéter » est que le pointeur de parcours reprenne la valeur initiale.

4. Création d'une liste circulaire :

Pour créer une liste circulaire, il faut trois (3) pointeurs : PC, PP, TETE.

Tête : permet de sauvegarder l'adresse du premier élément de la liste.

PP : permet de sauvegarder l'adresse du dernier élément de la liste.

PC : permet de créer chaque enregistrement pointeur à mettre dans la liste.

Exercice d'application :

Soit un tableau d'entiers de 50 cellules. Écrire un sous-programme qui transfère le contenu du tableau dans une liste circulaire avec tete.

5. Les opérations applicables aux listes circulaires :

Il est possible d'effectuer plusieurs traitements aux listes circulaires. Parmi ces traitements, nous avons les opérations scalaires, la recherche, le parcours, la modification, l'insertion...