

Série 3 : Listes doublement chaînées

Corrigée

Exercice 1

En considérant la définition d'une Liste doublement chaînée suivante

```
type
  <ident_liste>=enregistrement
    tete: ^noeud
    queue: ^noeud
    taille :entier
  fenreg

  noeud=enregistrement
    val :<type>
    suiv : ^noeud
    prec : ^noeud
  fenreg
```

Nb : la propriété taille donne le nombre des éléments contenus dans la liste

Donner l'implémentation de chacune des procédures et fonctions suivantes :

1. **Procédure insérerOptimisée(var L : Liste, x :type, position: entier)** Cette procédure permet d'insérer l'élément x à une position donnée en minimisant le nombre d'itérations permettant l'accès à la position d'insertion. C'est-à-dire si la position est plus proche de la tête que de la queue on commence le parcours à partir de la tête, sinon on commence le parcours à partir de la queue.

Procédure insérer(var L : Liste, x :type, position:entier)

Var p, predecesseur, courant : Liste, i:entier

Début

 Allouer(p)

 p^.val :=x

 p^.suiv :=NIL

 p^.prec:=NIL

 si (position<1 ou position > Taille(L)+1) alors

 écrire("position invalide")

 sinon

 si Taille(L)=0 alors

 L.tete:=p

 L.queue:=p

 sinon

 Si (position=1) alors //insertion en tete

 L.tete^.prec=p

 p^.suiv :=L.tete

 L.tete:=p

 sinon

 si (position =Taille(L)+1) alors //insertion en queue

 p^.prec:=L.queue

 L.queue^.suiv:=p

 L.queue:=p

 sinon //insertion en milieu de liste

 si (position <=L.taille/2) alors // la position est plus proche de la
 tête que de la queue

 courant:=L.tete

 i:=1

 tantque (i<position)faire

 predecesseur := courant

 courant := courant^.suiv

 i:=i+1

 fin Tq

 sinon // la position est plus proche de la queue que de la tête

```

        predecesseur:=L.queue
        i:=0
        tantque (i<=L.taille-position)faire
            courant := predecesseur
            predecesseur := predecesseur^.prec
            i:=i+1
        fin Tq
    fin si

    predecesseur^.suiv:=p
    p^.prec:=predecesseur
    p^.suiv:=courant
    courant^.prec:=p
    fin si
    fin si
    L.taille++
    fin si
    fin

```

- 2. Procédure SupprimerOptimisée(var L : Liste, position: entier)** Cette procédure permet supprimer l'élément se trouvant à une position donnée en minimisant le nombre d'itérations permettant l'accès à la position de suppression. C'est-à-dire si la position est plus proche de la tête que de la queue on commence le parcours à partir de la tête, sinon on commence le parcours à partir de la queue.

Procédure supprimer (var L : Liste, position:entier)

Var predecesseur, courant: ^noeud , i:entier

Début

si ((position < 1) ou (position > Taille(L))) alors
écrire ("position de suppression erronée")

sinon

si (L.tete=L.queue) alors

//La liste contient un seul élément

courant:=L.tete

L.tete:=NIL

L.queue:=NIL

Libérer(courant)

sinon

si (position = 1) alors

//suppression de la tête

courant := L.tete

L.tete:=L.tete^.suiv

L.tete^.prec:=NIL

Libérer(courant)

sinon

si (position=Taille(L))

//suppression de la queue

courant := L.queue

L.queue:=L.queue^.prec

L.queue^.suiv:=NIL

Libérer(courant)

sinon //cas général l'élément à supprimer est ni
la tête ni la queue

si (position <=L.taille/2) alors // la position est plus proche
de la tête que de la queue

courant:=L.tete

i:=1

Tant que (i<position)

predecesseur := courant

courant:=courant^.suiv

i:=i+1

Fin Tq

sinon // la position est plus proche de la queue que de la tête

predecesseur:=L.queue

```

        i:=0
        tantque (i<=L.taille-position)faire
            courant := predecesseur
            predecesseur := predecesseur^.prec
            i:=i+1
        fin Tq
    finsi
    // à la sortie de la boucle courant pointe sur le nœud
    // à supprimer et predecesseur pointe sur son prédécesseur
    predecesseur^.suiv:=courant^.suiv
    (courant^.suiv)^.prec:=predecesseur
    Libérer(courant)
    fin si
    fin si
    fin si
    L.taille—
finsi
fin

```

3. **Fonction ListeSupprimerRepetition(var L :Liste) :entier** Cette fonction permet de supprimer toutes les répétitions dans la liste et renvoyer le nombre de suppressions.

Fonction ListeSupprimerRepetition(var L :Liste) :entier

Var p,q, precedent,successeur:Noeud

Début

p:=L.tete //p représente le nœud pour lequel on va supprimer les répétitions

Tant que (p<>NIL) faire

q:=p^.suiv // q va parcourir la liste à la recherche de répétitions de p

Tant que(q<>NIL) faire

Si (p^.val=q^.val) alors

//suppression du nœud pointé par q

Si (q=L.queue)

L.queue:=L.queue^.prec

L.queue^.suiv:=NIL

Sinon

```

    precedent:=q^.prec
    successeur:=q^.suiv
    precedent^.suiv:=successeur
    successeur^.prec:=precedent
  fsi
  libérer(q)
fsi
q:=q^.suiv
ftq
p=p^.suiv
ftq
fin

```

4. **Procédure Append(Var L1 :Liste, L2 :Liste)** cette procédure ajoute les éléments de la liste L2 à la fin de la liste L1 (la liste L1 se retrouve alors modifiée)

Procédure Append(Var L1 :Liste, L2 :Liste)

Var nouveau,courant:Nœud

Début

Si (L1.tete=Nil) alors //L1 est vide

Si (L2.tete <> Nil) alors

Allouer(nouveau)

nouveau^.suiv:=Nil

nouveau^.prec:=Nil

nouveau^.val:=L2.tete^.val

L1.tete:=nouveau

L1.queue:=nouveau

courant:=L2.tete^.suiv

Tant que (courant<>Nil)

Allouer(nouveau)

```

nouveau^.suiv:=Nil
nouveau^.prec:=Nil
nouveau^.val:=courant^.val
L1.queue^.suiv:=nouveau
nouveau^.prec :=L1.queue
L1.queue:=nouveau
courant:=courant^.suiv
ftq
fsi
sinon // L1 n'est pas vide
  Si (L2.tete <>Nil) alors
    courant:=L2.tete

    Tant que (courant<>Nil)
      Allouer(nouveau)
      nouveau^.suiv:=Nil
      nouveau^.prec:=Nil
      nouveau^.val:=courant^.val
      L1.queue^.suiv:=nouveau
      nouveau^.prec :=L1.queue
      L1.queue:=nouveau
      courant:=courant^.suiv
    ftq
  fsi
fsi

```

Fin

Problème Liste Doublement chaînées

On veut gérer à travers une liste doublement chaînée un ensemble de livres. Pour chaque livre on garde le **titre** et la **quantité** qui indique le nombre de copies disponibles. Faite la déclaration des types **livre** et **noeud, liste** nécessaires pour notre liste puis répondre à ces questions:

Type

livre=enregistrement

titre:chaîne

quantite:entier

fenreg

liste=enregistrement

tete:^noeud

queue:^noeud

fenreg

noeud=enregistrement

val :livre

suiv : ^noeud

prec : ^noeud

fenreg

1. Écrire **Procédure ajouterLivre(var L:liste, titre:chaine, nbCopies:entier)** qui ajoute un livre dans la liste (en précisant le nombre de copies à ajouter). Si le livre existe déjà, il y'aura augmentation du nombre de copies. Si le livre n'existe pas et que sa quantité est égale à 1 l'ajout sera en tête de liste. Sinon, l'ajout sera en queue de liste.

Procédure ajouterLivre(var L:liste, titre:chaine, nbCopies:entier)

Var

nouveau:^noeud

liv:livre

courant:^noeud

trouve: boolean

Début

//rechercher le livre dans la liste

courant:=L.tete

trouve:=faux

tant que (courant<>NIL et non trouve) faire

si (courant^.val.titre=titre) alors

 trouve:=vrai

sinon

 courant :=courant^.suiv

fsi

ftq

si (trouve) alors

 courant^.val.quantite := courant^.val.quantite+nbCopies

sinon //insertion dans la liste

 liv.titre:=titre

 liv.quantite:=nbCopies

 Allouer(nouveau)

 nouveau^.val:=liv

 nouveau^.suiv:=Nil

 nouveau^.prec:=Nil

// cas ou L est vide

 Si (L.tete=Nil) alors

 L.tete:=nouveau

 L.queue:=nouveau

 Sinon

 // si le nombre de copies est 1 l'insérer en tête

 Si (nbCopies=1) alors

 nouveau^.suiv:=L.tete

 L.tete^.prec:=nouveau

 L.tete:=nouveau

 Sinon //insertion en queue

 L.queue^.suiv:=nouveau

 nouveau^.prec:=L.queue

 L.queue:=nouveau

 fsi

 fsi

fsi

Fin

2. Ecrire une fonction **taille** qui retourne la taille de la liste.

Fonction taille (L:liste):entier

 Var t:entier

 courant: ^nœud

Début

 si (L.tete=Nil) alors

 retourner 0

 sinon

 t:=0

 courant := L.tete

 tant que (courant <>Nil) faire

 t:=t+1

```

        courant:=courant^.suiv
    ftq
    retourner t
fsi
Fin

```

3. Écrire une procédure **afficher (L:liste)** qui affiche tous les Livres dans la liste.

```

procédure afficher(L:liste)
var
    courant:^noeud
début
    courant := L.tete
    tantque (courant<>Nil) faire
        écrire (courant^.val.titre, "nombre de copies",
            courant^.val.quantite)
        courant := courant ^.suiv
    ftq
fin

```

Ecrire une procédure **décrémenterLivre(L:liste, titre:chaine)**

4. qui décrémente le nombre de livres disponibles pour un titre donné. Le parcours doit commencer à partir de la queue.

```

Procédure décrémenterLivre (L:liste, titre:chaine)
var
    courant:^noeud
    trouve:boolean
Début
    trouve:=faux
    courant := L^.queue
    tantque (courant<>Nil et non trouve) faire
        si (courant^.val.titre=titre) alors
            courant^.val.quantite:= courant^.val.quantite-1
            trouve:=vrai
        sinon
            courant := courant^.prec
    ftq
fin

```

1. Écrire une fonction **supprimerVide (var L:liste):boolean** qui supprime le premier livre à partir du début dont la quantité est égale à zéro tout en

retournant la valeur vrai. Sinon, aucun livre ne sera supprimé et la fonction retourne faux.

```
fonction supprimerVide (var L:liste):boolean
var
    courant:^nœud
    predecesseur:^noeud
    trouve:boolean
Début
    trouve:=faux
    courant := L^.tete
    tantque (courant<>Nil et non trouve) faire
        si (courant^.val.quantite=0) alors
            si (courant =L^.tete) alors
                si L.tete:=L.queue alors
                    L.tete:=Nil
                    L.queue:=Nil
                sinon
                    L.tete:=L.tete^.suiv
                    L.tete^.prec:=Nil
            fsi
        Sinon
            Si (courant =L.queue) alors //supp queue
                L.queue :=L.queue^.prec
                L.queue^.suiv := Nil
            Sinon //ni tete ni queue
                predecesseur := courant^.pred
                predecesseur ^.suiv:=courant^.suiv
                courant^.suiv^.prec:= predecesseur
            fsi
        fsi
    Libérer=(courant)
    trouve:=vrai
sinon
    courant := courant^.suiv
    fsi
ftq
retourne trouve
fin
```

5. Écrire une **procédure** *EditerLivre(liste L, pos:entier)* qui affiche le livre se trouvant à la position **pos** de la liste, en optimisant la recherche de ce livre.

procédure *EditerLivre(liste L, pos:entier)*

var

taille:entier

courant:^nœud

Début

taille := taille(L)

Si (pos < 1 ou pos > taille) alors

Ecrire ("position invalide")

Sinon

Si (pos < taille/2) alors

//recherche à partir de la tête

courant := L^.tete

p:=1

tant que (p<pos) faire

courant^=courant^.suiv

p++;

ftq

Sinon

//recherche à partir de la queue

courant := L^.queue

p:=1

tant que (p<=taille-pos) faire

courant^=courant^.prec

p++;

ftq

Fsi

écrire(courant^.val.titre, " quantite=", courant^.val.quantite

fsi

fin

6. Écrire une procédure **supprimerToutVide(var Liste L)** qui supprime tous les livres dans la liste dont la quantité est égale à zéro.

//première solution avec appel à la fonction supprimer vide

```

procédure supprimerToutVide(var Liste L)
var
  suppression:boolean
début
  suppression:=supprimerVide(L)
  tant que (suppression=vrai) faire
    suppression:=supprimerVide(L)
  fin tq
fin

```

//deuxième solution sans appel à la fonction supprimer vide

```

procédure supprimerToutVide(var Liste L)
var
  courant:^nœud
  precedent:^noeud
  trouve:boolean

début
  répéter
    trouve:=faux
    courant := L^.tete
    tantque (courant<>Nil et non trouve) faire
      si (courant^.val.quantite=0) alors
        si (courant =L^.tete) alors
          si L.tete=L.queue alors
            L.tete:=Nil
            L.queue:=Nil
          sinon
            L.tete:=L.tete^.suiv
            L.tete^.prec:=Nil
          Fsi
        Si (courant =L^.queue) alors
          L.queue:=L.queue^.prec
          L.queue^.suiv:=Nil

      Sinon //ni tete ni queue
        precedent :=courant ^.prec
        precedent^.suiv:=courant^.suiv

```

```
        courant^.suiv^.prec:=precedent
    fsi
fin si
    Libérer=(courant)
    trouve:=vrai
sinon

    courant := courant^.suiv
fsi
ftq

jusqu'à (trouve = faux)
fin
```