

## Correction du TD n°3

### QUESTIONS DE COURS

1) Qu'est-ce qu'un pointeur ?

R : Un **pointeur** P est une variable statique dont les valeurs sont des adresses.

2) Quelle est la différence entre une structure de données statique et une structure de données dynamique ?

R : La différence entre une structure de données statique et une structure de données dynamique est la taille, la première est définie au début en spécifiant la taille qui est fixe, par contre pour la deuxième la taille est variable et qui est basée sur l'allocation dynamique.

### EXERCICE N°1 (RAPPEL DU COURS)

1)

```
Type  
    Liste : ^cellule  
    Cellule : Enregistrement  
        |  
        val : Entier  
        |  
        suiv : Liste  
    Fin Cellule
```

2)

```
Var  
    L : Liste
```

3)

```
Procédure CréeListe(n : Entier ; var L : Liste)  
Var  
    Tete, p : Liste  
    i : Entier  
Début  
    Allouer(Tete)  
    Ecrire("Entrer élément Tête :")  
    Lire(Tete^.val)  
    Tete^.suiv ← nil  
    L ← Tete  
  
    Pour i de 2 à n faire  
        Allouer(p)  
        Ecrire("Entrer élément n° :", i)  
        Lire(p^.val)  
        p^.suiv ← nil  
        L^.suiv ← p  
        L ← p  
    Fin Pour  
    L ← Tete //se pointer sur la tête de la liste  
Fin
```

4)

```

Procédure AffichageIter(L : Liste)
Var
    p : Liste
Début
    p ← L
    Tant que p ≠ nil Faire
        Ecrire(p^.val)
        p ← p^.suiv
    Fin Tant que
Fin

```

```

Procédure AffichageRecu(L : Liste)
Début
    Si L ≠ nil Alors
        Ecrire(p^.val)
        AffichageRecu(p^.suiv)
    Fin Si
Fin

```

5)

*Version itérative*

```

Fonction Recherche(x : Entier ; L : Liste) : Booléen
Var
    p : Liste
Début
    p ← L
    Tant que ((p ≠ nil) ET (p^.val ≠ x)) Faire
        p ← p^.suiv
    Fin Tant que
    Si p = nil Alors
        Recherche ← Faux
    Sinon
        Recherche ← Vrai
    Fin Si
Fin

```

*Version récursive*

```

Fonction Recherche(x : Entier ; L : Liste) : Booléen
Début
    Si L = nil Alors
        Recherche ← Faux
    Sinon
        Si L^.val = x Alors
            Recherche ← Vrai
        Sinon
            Recherche ← Recherche(x, L^.suiv)
        Fin Si
    Fin Si
Fin

```

6)

```

Procédure AjouterTete(x : Entier ; var L : Liste)
Var
    Tete : Liste
Début
    Allouer(Tete)
    Tete^.val ← x
    Tete^.suiv ← L
    L←Tete
Fin

```

7)

Version itérative

```

Procédure Supprimer(x : Entier ; var L : Liste)
Var
    P,Q : Liste
Début
    Si L = nil Alors
        Ecrire("Liste vide, impossible de supprimer ",x)
    Sinon
        Si L^.val = x Alors
            P ← L //mémorisation de l'élément à supprimer
            L ← L^.suiv
        Sinon
            P ← L^.suiv
            Tant que ((P ≠ nil) ET (P^.val ≠ x)) Faire
                Q ← P
                P ← P^.suiv
            Fin Tant que
            Si P ≠ nil Alors
                Q^.suiv ← P^.suiv
                Libérer(P)
            Fin Si
        Fin Si
    Fin Si
Fin

```

Version récursive

```

Procédure Supprimer(x : Entier ; var L : Liste)
Var
    P : Liste
Début
    Si L ≠ nil Alors
        Si L^.val = x Alors
            P ← L
            L ← L^.suiv
            Libérer(P)
        Sinon
            Supprimer(x, L^.Suiv)
        FinSi
    FinSi
Fin

```

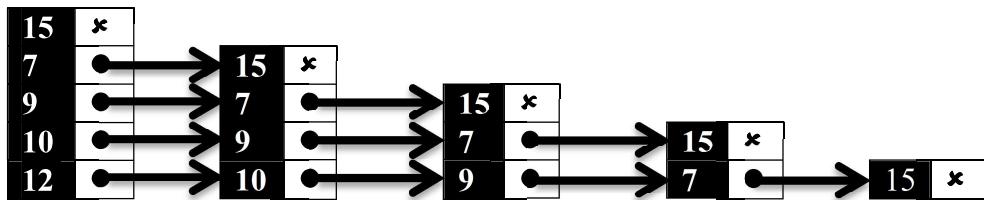
## EXERCICE N°2

```

Procédure Inverse (var L : Liste)
Var
    P, Q : Liste
Début
    P ← nil
    Tant que L ≠ nil Faire
        Q ← L^.suiv
        L^.suiv ← P
        P ← L
        L ← Q
    Fin Tant que
Fin

```

Trace d'exécution de l'exemple ci-dessous



## EXERCICE N°3

```

Procédure TriBulle(var L : Liste)
Var
    P, Q : Liste
    Temp : Entier
Début
    P ← L
    Q ← L^.suiv
    Répéter
        rep ← faux
        P ← L           // pointeur sur le premier élément
        Q ← L^.suiv     // pointeur sur le deuxième élément
        Tant que Q ≠ nil Faire
            Si P^.val > Q^.val Alors
                Temp ← P^.val
                P^.val ← Q^.val
                Q^.val ← Temp
                rep ← vrai
            Fin Si
        Fin tant que
    Jusqu'à rep = faux
Fin

```

Permutation

## EXERCICE N°4

```
Procédure Concatener(L1,L2 : Liste ; var L3 : Liste)
Début
    Si L1 = nil ET L2 = nil Alors
        L3 ← nil
    Sinon Si L1 ≠ nil ET L2 = nil Alors
        L3 ← L1
    Sinon Si L1 = nil ET L2 ≠ nil Alors
        L3 ← L2
    Sinon
        //affectation de la première liste L1
        L3 ← L1
        //se pointer sur le dernier élément de L3
        Tant que L3^.suiv ≠ nil Faire
            L3 ← L3^.suiv
        Fin Tant que
        //affectation de la liste L2 à la fin de L3
        L3^.suiv ← L2
        //se pointer sur la tête de la liste
        L3 ← L1
    Fin Si
Fin
```

## EXERCICE N°5

```
Procédure Fusion(var L1 :Liste ; L2 : Liste)
Var
Tete, P, Q : Liste
Début
    Si L1 = nil Alors //Si L1 est vide
        L1 ← L2
    Sinon
        Si L2 ≠ nil Alors
            //Fixation de la position de la tête de la liste
            Si L1^.val ≤ L2^.val Alors
                Tete ← L1
            Sinon
                Tete ← L2
            Fin Si
            Q ← Tete //mémorisation de l'élément précédent
            Tant que L1 ≠ nil ET L2 ≠ nil Faire
                Si L1^.val > L2^.val Alors
                    P ← L2
                    L2 ← L2^.suiv
                    Q^.suiv ← P
                    P^.suiv ← L1
                    Q ← P
                Fin Si
            Fin Tant que
        Fin Si
    Fin Si
Fin
```

```

    Sinon Si L1^.val > L2^.val Alors
        P ← L2
        L2←L2^.suiv
        Q ← L1
        L1←L1^.suiv
        Q^.suiv← P
        P^.suiv← L1
    Sinon
        Q ← L1 //mémorisation de l'élément précédent
        L1 ← L1^.suiv
    Fin Si
    Fin Tant que
    L1 ←Tete      //pointer L1 à la tête de la liste
Fin Si
Fin Si
Fin

```

### EXERCICE N°6

```

Procédure SupprimeDoublons(var L :Liste)
Var
Q , P : Liste
Début
    Q ← L
    Si L ≠ nilAlors
        Tant que Q^.suiv ≠ nilFaire
            P ← Q
            Q ← Q^.suiv
            Si Q^.val = P^.valAlors
                //on supprime de L l'élément pointé par Q.
                Q^.suiv ← P^.suiv
                Libérer(P)
            Fin Si
        Fin Tant que
    Fin Si
Fin

```

Liste  
triée →

```

Procédure SupprimeDoublons(var L :Liste)
Var
Q : Liste
Vpred : caractère
Début
    Q ← L
    Si L ≠ nilAlors
        Tant que Q^.suiv ≠ nilFaire
            Vpred←Q^.val
            Q ← Q^.suiv
            Si Q^.val = Vpred Alors
                Supprimer(Vpred,Q)
            Fin Si
        Fin Tant que
    Fin Si
Fin

```

Liste  
non  
triée →

Cette procédure  
se charge de  
parcourir la liste  
Q en supprimant  
toutes les valeurs  
de Vpred

## EXERCICE N°7

```

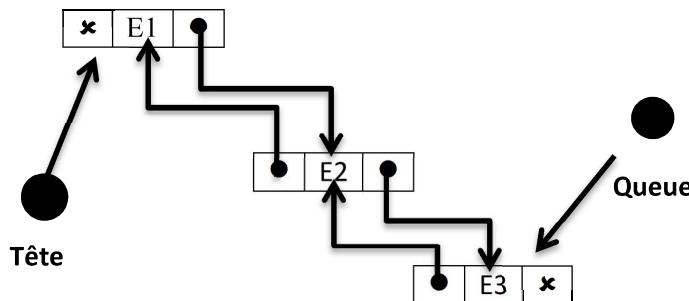
Fonction Palindrome(L : Liste) : Booléen
Var
    Stop : Booléen
    Deb, Fin, Q : Liste
Début
    Deb ← L           // Pointeur sur le premier élément
    Fin ← nil         // Pointeur sur le dernier élément
    Stop ← faux       // Signale l'interruption du traitement
    Tant que Deb^.suiv ≠ Fin ETNON(Stop) Faire
        // Récupérer le pointeur sur le dernier élément de la liste
        Q ← P
        Tant que Q^.suiv ≠ Fin Faire
            Q ← Q^.suiv
        Fin Tant que

        Si P^.val = Q^.val Alors
            Fin ← Q // se pointer sur le dernier élément de la liste L
        Sinon
            Stop ← vrai // arrêt du traitement
        Fin Si
        Si Deb ≠ Fin Alors
            Deb ← Deb ^.suiv
        Fin Si
    Fin Tant que
    Palindrome ← NON(Stop)
Fin

```

## EXERCICE N°8

- L'intérêt d'une liste doublement chainée par rapport à une liste chainée simple c'est pour accélérer la recherche d'un élément.



```

Type
    Liste : ^cellule
    Cellule : Enregistrement
        pred : Liste
        val : Entier
        suiv : Liste
    Fin Cellule
Var
    L : Liste

```

1)

```

Fonction Premier(L : Liste) : Liste
Début
    Si L ≠ nullAlors
        Premier ← L
    Sinon
        Premier ← nil
    Fin Si
Fin

```

2)

*Solution n° 1 : On connaît la queue de la liste*

```

Fonction Dernier(Queue : Liste) : Liste
Début
    Dernier ← Queue
Fin

```

*Solution n° 2 : On ne connaît pas la queue de la liste*

```

Fonction Dernier(L : Liste) : Liste
Var
    P : Liste
Début
    P ← L
    Tant que P^.suiv ≠ nullFaire
        P ← P^.suiv
    Fin Tant que
    Dernier ← P
Fin

```

3)

```

Fonction estVide(L : Liste) : Booléen
Début
    Si L = nilAlors
        estVide ← vrai
    Sinon
        estVide ← faux
    Fin Si
Fin

```

4)

```

Procédure supprimerPremier(var L : Liste)
Var
    P : Liste
Début
    Si L ≠ nilAlors
        P ← L
        L ← L^.suiv
        L^.pred ← nil
        Libérer(P)
    Fin Si
Fin

```

5)

```

Procédure ajouterAprès(P,Q : Liste ; var L : Liste)
Var
    D : Liste
Début
    D  $\leftarrow$  L
    Tant que D  $\neq$  Q ET D  $\neq$  nil Faire
        D  $\leftarrow$  D^.suiv
    Fin Tant que
    Si D = Q Alors
        D  $\leftarrow$  D^.suiv
        D^.pred  $\leftarrow$  P
        P^.suiv  $\leftarrow$  D
        Q^.suiv  $\leftarrow$  P
        P^.pred  $\leftarrow$  Q
    Sinon
        Ecrire ("Ajout impossible, élément non existant")
    Fin Si
Fin

```

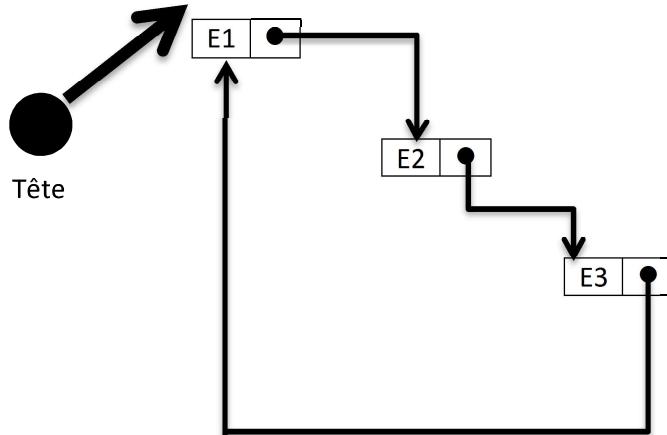
6)

```

Procédure Supprimer(x : Entier ; var L : Liste)
Var
    P,Q,D : Liste
Début
    Si L = nil Alors
        Ecrire("Liste vide, impossible de supprimer ",x)
    Sinon
        Si L^.val = x Alors
            P  $\leftarrow$  L
            L  $\leftarrow$  L^.suiv
        Sinon
            P  $\leftarrow$  L^.suiv
            Tant que ((P  $\neq$  nil) ET (P^.val  $\neq$  x)) Faire
                Q  $\leftarrow$  P
                P  $\leftarrow$  P^.suiv
            Fin Tant que
            Si P  $\neq$  nil Alors
                D  $\leftarrow$  P^.suiv
                D^.pred  $\leftarrow$  Q
                Q^.suiv  $\leftarrow$  D
                Libérer(P)
            Fin Si
        Fin Si
    Fin Si
Fin

```

## EXERCICE N°9



Une liste circulaire est une structure de données dynamique dont le dernier élément de la liste pointe sur la tête de la liste.

```
Procédure AjouterTête(e : Entier ; var L : Liste)
Var
    N,Tete,Queue: Liste
Début
    Allouer(N)
    N^.val ← e
    Si L = nil Alors
        L ← N
        N^.suiv ← L
        L^.suiv ← N
    Sinon
        Tete ← L
        Queue ← L
        Tant que Queue^.suiv ≠ Tete Faire
            Queue ← Queue^.suiv
        Fin Tant que
        Queue^.suiv ← N
        N^.suiv ← Tete
        Tete ← N
        L ← Tete
    Fin Si
Fin
```

**Remarque :** ajouter au début ou à la fin d'une liste chainée circulaire, c'est d'appliquer le même algorithme que celui d'ajouter tête de liste.