

Correction du TD N°5

Ex1 :

```

Fonction inverseListe (l1 :Liste) :Liste
var
    l2 :Liste
    P : ^Cellule
debut
    cree_liste(l2)
    P^ ← (l1.premier)^
    TantQue (P # Nil) Faire
        inserer_avant_premier (P^.cle, l2)
        P^ ← (P^.Suiv)^
    FinTQ
    inverseListe ← l2
Fin FN

```

Ex2 :

```

Procédure TriBul(Var Ls : Liste)
Var
    P,Q : ^Cellule
    x : Entier
    échange : Booléen
Début
    Répéter
        Échange ← Faux
        P←Ls.premier
        Q←P^.Suiv
        TantQue (Q # Nil) Faire
            Si (P^.cle > Q^.cle) Alors
                x← P^.cle
                P^.cle← Q^.cle
                Q^.cle← x
                échange←Vrai
            FinSi
            P← Q
            Q← Q^.Suiv
        FinTQ
        Jusqu à (échange = Faux)
Fin Proc

```

Ex3

| | | |
|--|--|--|
| Etudiant=struct nom :chaine note :réel finStruct | Cellule =Struct cle :Etudiant suiv :^Cellule finStruct | Liste=struct premier :^Cellule dernier :^Cellule finStruct |
|--|--|--|

Fonction Moyenne (L : Liste) : réel

Var : p : ^Cellule
Som, nbEtud : réel
Etd : Etudiant

Début

P \leftarrow L.premier
Som \leftarrow 0
nbEtud \leftarrow 0

Tant que p != NULL faire
 Etd \leftarrow p^.cle
 Som \leftarrow Som + Etd.note
 nbEtud ++
 p \leftarrow p^.suiv

finTQ

 si (nbEtud=0) alors Moyenne \leftarrow 0
 sinon Moyenne \leftarrow (Som/nbEtud)
 finSi

Fin FN

Ex4

procedure InsertTrié (x: entier, var L : Liste)

Var

P : ^Cellule

Début

P \leftarrow L.premier
TantQue (P # Nil) ET (P^.cle < x) Faire
 P \leftarrow P^.suiv

FinTQ

Si (p=L.premier) alors
 inserer_avant_premier (x, L)
Sinon si (p=Nil) alors
 inserer_apres_dernier (x, L)
Sinon
 inserer_avant (x, P)

finSi

Fin proc

Ex 5

Fonction concaténation (L1 : liste, L2 : liste) :Liste

Var :

P1, P2 : ^Cellule
L :Liste

Début

Créer_Liste (L)

 P1 \leftarrow L1.premier
 Tant que P1 \neq NULL faire
 inserer_apres_dernier(P1^.cle, L)
 P1 \leftarrow P1.suiv

Fin TQ

```

P2 ← L2.premier
Tant que P2 ≠NULL faire
    inserer_apres_dernier(P2^.cle, L)
    P2←P2.suiv
Fin TQ

Concaténation←L

Fin FN

procedure Séparer ( L : liste, var L1 : liste, var L2 : liste)
Var
P: ^Cellule ;
B : booléen
Début
    Créer_Liste(L1)
    Créer_Liste (L2)

    B←true
    P←L.premier
    Tant que P≠NULL faire
        Si B alors
            inserer_apres_dernier(P^.cle, L1)
        Sinon
            inserer_apres_dernier(P^.cle, L2)
        finSi
        B← !B
        P←P^.suiv
    FinTQ

Fin proc

Fonction Fusion (L1 : liste, L2 : liste): liste
Var
P1,P2 : ^Cellule
L:liste
Début
    CréerListe(L)
    P1← L1.premier
    P2←L2.premier
    Tant que (P1≠NULL et P2≠NULL)
        Si P1^.cle< P2^.cle alors
            inserer_apres_dernier (P1^.cle, L)
            P1←P1.suiv
        Sinon
            inserer_apres_dernier (P2^.cle, L)
            P2←P2.suiv
        finSi
    FinTQ

    Tant que P1≠NULL
        inserer_apres_dernier (P1^.cle, L)
        P1←P1.suiv
    FinTQ

    Tant que P2≠NULL

```

```

inserer_apres_dernier (P2^cle, L)
P2←P2.suiv
FinTQ
Fusion←L
Fin

Procedure TriFusion (Le : liste, var Ls liste)
Var
L1, L2, L1b, L2b : liste
Début
    Si (Le.premier)^.suiv != NULL alors //contient +qu'un elem
        Séparer (Le, L1, L2)
        Trifusion(L1, L2b)
        TriFusion(L2, L2b)
        Ls←Fusion(L1b, L2b)
    Sinon
        CreerListe(Ls)
        inserer_avant_premier (Le.premier)^.cle , Ls)
        //ou inserer_apres_dernier (Le.premier)^.cle , Ls) c'est le même dans ce cas
        finSi
Fin proc

```