Manipulation d'un Tableau

```
fonction RechercheMaxTab(T: Tab, taille: entier) : réel
début
    max <-- T[0]
    pour i de 1 à taille-1 faire
        si max < T[i] alors
            max <-- T[i]
        fin_si
    fin_pour
    retourner max
fin
```

```
fonction RechercheMinTab(T: Tab, taille: entier) : réel
début
    min <-- T[0]
    pour i de 1 à taille-1 faire
        si min > T[i] alors
            min <-- T[i]
        fin_si
    fin_pour
    retourner min
fin
```

```
fonction RechercheElementTab1(T: Tab, taille: entier, elt: réel) : entier
début
    /* recherche d'un élèment dans un tableau on utilisant
    la boucle REPETER...JUSQU'A*/
   i <-- 0
   trouve <-- faux
   répéter
      si T[i] = elt alors
         trouve <-- vrai
      fin_si
      i <-- i+1
   jusqu'à  (i > taille-1 Ou trouve = vrai )
   retourner i
fin
```

```
fonction RechercheElementTab2(T: Tab, taille: entier, elt: réel) : entier
début
    /* recherche d'un élèment dans un tableau on utilisant
    la boucle TANTQUE...FAIRE*/
   i <-- 0
   trouve <-- faux
   tant que  (i <= taille-1 Et trouve = faux) faire
      si T[i] = elt alors
         trouve <-- vrai
      fin_si
      i <-- i+1
   fin_tant_que
   retourner i
fin
```

```
fonction NombreOccurenceElementTab(T: Tab, taille: entier, elt: réel) : entier
début
   nb <-- 0
   pour i de 0 à taille-1 faire
      si T[i] = elt alors
         nb <-- nb+1
      fin_si
   fin_pour
   retourner nb
fin
```

```
fonction premiereOccurenceTab(T:Tab,Taille:entier,elt:réel) : entier
début
    /* ici deux cas possible, soit on trouve l'element =>
    on s'arrête la recherche et on retourne son indice i,
     soit on a arrivé à la fin du tableau sans trouvé l'élèment */
     i <-- 0
     trouve <-- faire
     tant que (trouve =faux ) Et (i <= Taille-1) faire
         si T[i] =elt alors
            trouve <-- vrai
         fin_si
         i <-- i+1
     fin_tant_que
     retourner(i)
fin
```

```
fonction SommeElementsTab1(T: Tab, taille: entier) : réel
début
    somme <--0
    pour i de 0 à taille-1 faire
        somme <-- somme +T[i]
    fin_pour
    retourner somme
fin
```

```
fonction ProduitElementsTab2(T: Tab, taille: entier) : réel
début
    prod <-- 1
    pour i de 0 à taille-1 faire
        si (T[i] <> 0) alors
            prod <-- prod * T[i]
        fin_si
    fin_pour
    retourner prod
fin
```

```
fonction ProduitElementsTab2(T: Tab, taille: entier) : réel
début
    prod <-- 1
    pour i de 0 à taille-1 faire
        si (T[i] <> 0) alors
            prod <-- prod * T[i]
        fin_si
    fin_pour
    retourner prod
fin
```

```
procédure InverseTableau(@T: Tab, taille:entier)
début
    i <-- 0
    halfT <-- taille div 2
    tant que i≤ halfT  faire
        // on va faire une permutation des valeurs
        v <-- T[i]
        T[i] <--  T[taille-i-1]
        T[taille-i-1] <-- v

        i <-- i+1
    fin_tant_que
fin
```

Manipulation des deux Tableaux(1)

```
fonction SaisirTailleTableau(borneInf:entier, borneSup: entier) : entier
début
    /*   par exemple; taille du tableau entre 20 et 50 ==> borneInf=20 et borneSup=50 ...
    les valeurs des borneInf et borneSup seront données dans le programme principale...
    dans ce cas on est pas obligé de developper un module saisie taille tableau pour chaque tableau!*/
    répéter
        écrire("Saisir un nombre positif entre ",bornInf,"et",borneSup)
        lire(tailleTab)
    jusqu'à ( tailleTab ≥ borneInf Et tailleTab <= borneSup)
    retourner tailleTab
fin
```

TDOL

Objet	Nature / Type
tailleTab	entier

```
procédure RemplissageTableau(@T:Tab,nbE:entier)
début
    pour i de 0 à nbE-1 faire
        répéter
            écrire("Saisir un chiffre inferieur ou égale à 20")
            lire(T[i])
        jusqu'à T[i] ≥ 20

    fin_pour


fin
```

TDOL

Objet	Nature / Type
i	entier

```
procédure AffichageTableau(@T:Tab,nbE:entier)
début
    /* Affichage sera de cette forme ..Element n°1 :23..
    on commence par 1 non pas par 0 ..pour cette raison on écrit dans l'affichage .
    .(i+1)..au lieu de .. (i )*/

    pour i de 0 à nbE-1 faire
        écrire("Elèment n°", i+1 ,": ", T[i])
    fin_pour
fin
```

TDOL

Objet	Nature / Type
i	entier

```
procédure SommeDeuxTableaux(@AB:Tab,A:Tab,B:Tab,nbE:entier)
début
    // les trois tableaux(A, B et AB) sont de même taille => on utilise le même compteur i
    pour i de 0 à nbE-1 faire
        AB[i]<-- A[i]+B[i]
    fin_pour
fin
```

TDOL

Objet	Nature / Type
i	entier

Tableau 1 : | 4 | 8 | 7 | 9 | 1 | 5 | 4 | 9 |

Tableau 2 : | 7 | 6 | 5 | 2 | 1 | 3 | 7 | 4 |

Tableau 3 : | 11 | 14 | 12 | 11 | 2 | 8 | 11 | 10 |

```
procédure ProduitDeuxTableaux(@AB:Tab,A:Tab,B:Tab,nbE:entier)
début
    pour i de 0 à nbE-1 faire
        AB[i]<-- A[i] * B[i]
    fin_pour
fin
```

TDOL

Objet	Nature / Type
i	entier

Tableau 1 : | 4 | 8 | 7 | 12 |

Tableau 2 : | 3 | 6 |

Le Schtroumpf sera :

3*4+3*8+3*7+3*12+6*4+6*8+6*7+6*12 = 279

```
fonction Schtroumpf(A:Tab,B:Tab,nbE:entier) : entier
début
    res=0
    pour i de 0 à nbE-1 faire
        pour j de 0 à nbE-1 faire
            res <-- res+A[i] * B[j]
        fin_pour
    fin_pour
    retourner res
fin
```

TDOL

Objet	Nature / Type
i	entier
j	entier
res	entier

```
// PROGRAMME PRINCIPALE
algorithme ManipulationDeuxTableaux
début
    tailleTab <-- SaisirTailleTableau(20,50)
    RemplissageTableau(T1,tailleTab)
    RemplissageTableau(T2,tailleTab)
    SommeDeuxTableaux(S,T1,T2,tailleTab)
    ProduitDeuxTableaux(P,T1,T2,tailleTab)
    sch <-- Schtroumpf(T1,T2,tailleTab)
fin
```

TDNType

Nouveaux Types
Tab = tableau de 50 entier

TDO

Objet	Nature / Type
tailleTab	entier
T1, T2, S, P	Tab
sch	entier

Manipulation des deux Tableaux(2)

```
fonction SaisirTailleTab() : entier
début
    répéter
        écrire("Saisir un nombre entre 10 et 15")
        lire(n)
    jusqu'à  n≤ 10 Et n≥ 15
    retourner n
fin
```

TDOL

Objet	Nature / Type
n	entier

```
procédure RemplirTableau(@T:Tab, n:entier)
début
    pour i de 0 à n-1 faire
        T[i]<-- aléa(0, 9)
    fin_pour
fin
```

TDOL

Objet	Nature / Type
i	entier

```
fonction NombreOccurence(T:Tab,n:entier, elt:entier) : entier
début
    nbocc<-- 0
    pour i de 0 à n-1 faire
        si T[i] =elt alors
            nbocc <-- nbocc+1
        fin_si
    fin_pour
    retourner nbocc
fin
```

TDOL

Objet	Nature / Type
i	entier
nbocc	entier

```
procédure RemplirTableauOccurence(@Tocc: Tabc,T:Tab, n: entier)
début
    pour i de 0 à 9 faire
        Tocc[i]<-- NombreOccurence(T,n,i)
    fin_pour
fin
```

TDOL

Objet	Nature / Type
i	entier

```
algorithme Occurence
début
    n <-- SaisirTailleTab()
    RemplirTableau(T,n)
    RemplirTableauOccurence(Tocc,T,n)
fin
    Tableau de declaration de Nouveaux types(TDNT)
```

Nouveaux Types
Tab = tableau de 15 entier
Tabc = tableau de 10 entier

Objet	Nature / Type
n	entier
Tocc	Tabc
T	Tab

Manipulation d'une Chaîne de caractères

```
fonction SaisirChaineSimple() : chaîne
début
    écrire("Saisir quelque choses: ")
    lire(ch)
    retourner (ch)
fin
```

Objet	Nature / Type
ch	chaîne

```
fonction SaisirChaineConditionLongueur(borneInf:entier,borneSup:entier) : chaîne
début
    répéter
        écrire("Saisir un texte dont le nombre de caractères entre ",borneInf,"et",borneSup)
        lire(ch)
    jusqu'à ( ch ≤ borneSup Et ch ≥ borneInf)
    retourner ch
fin
```

Objet	Nature / Type
ch	chaîne

```
fonction VerificationMiniscule(ch:chaîne) : booléen
début
    miniscule <-- vrai
    i <-- 0
    tant que miniscule = vrai Et i <= long(ch)-1 faire
        si non(ch[i] dans ['a'..'z']) alors
            miniscule <-- faux
        fin_si
        i <-- i+1
    fin_tant_que
    retourner  miniscule
fin
```

```
fonction VerificationMajuscule(ch:chaîne) : booléen
début
    majuscule <-- vrai
    i <-- 0
    tant que majuscule = vrai Et i <= long(ch)-1 faire
        si non(ch[i] dans ['A'..'Z']) alors
            majuscule <-- faux
        fin_si
        i<-- i+1
    fin_tant_que
    retourner  majuscule
fin
```

Objet	Nature / Type
i	entier
majuscule	booléen

```
fonction NombreOccurenceCaractere(ch:chaîne, c: caractère) : entier
début
    nbOcc <-- 0
    pour i de 0 à long(ch)-1 faire
        si ch[i] = c alors
            nbOcc <-- nbOcc+1
        fin_si
    fin_pour
    retourner nbOcc
fin
```

Objet	Nature / Type
i,nbOcc	entier

```
fonction Inverser(ch:chaîne) : chaîne
 début
    //inverser (exp: Bonjour => ruojnoB)
    inverseCh <-- ""
    pour i de 0 à len(ch)-1 faire
        inverseCh <-- ch[i]+ inverseCh
    fin_pour
   retourner inverseCh
 fin
```

Objet	Nature / Type
i	entier
inverseCh	chaîne

```
algorithme ManipulationChaine
début
    // Saisir une chaîne sans condition sur la longueur
    ch1 <-- SaisirChaineSimple()
    // Saisir une chaîne de longueur entre 50 et 200
    ch2 <-- SaisirChaineConditionLongueur(50,200)
    // verification si une chaine saisie est entièrement miniscule
    si VerificationMiniscule(ch2) = vrai  alors
        écrire(" La chaîne saisie est entièrement miniscule")
    fin_si
    // verification si une chaine est entièrement majuscule
    si VerificationMajuscule(ch2) = vrai alors
        écrire(" La chaîne saisie est entièrement majuscule")
    fin_si
    // calculer le nombre d'espace dans une chaîne
    écrire("Le nombre des éspaces est :",NombreOccurenceCaractere(ch2, " "))
    // verifier si une chaine est palindrome(exp: RADAR)
    si ch1 = Inverser(ch1) alors
        écrire("La chaîne est palindrome")
    fin_si
fin
```

TDO globale

Objet	Nature / Type
ch1,ch2	chaîne