Partie 1 : Les vecteurs

Exercice No 1

Soit un vecteur T (tableau à une dimension) contenant N nombres entiers (N≤100). Ecrire un algorithme qui :
1- Détermine le mini, le maxi et la moyenne des éléments d'un tableau T
2- Calcule la somme et le produit scalaire de deux vecteurs (T1 et T2).
3- Inverse le contenu d'un vecteur T.
4- Calcule le nombre d'occurrences d'une valeur V dans T
5- Détermine la différence minimale entre deux éléments quelconques du vecteur T.
6- Supprime toutes les valeurs doubles d'un vecteur T.

Exercice 2 :

Soit T un vecteur de N (N≤250) entiers supposés positifs. Ecrire un algorithme permettant, à partir de T, de construire deux vecteurs M3 et M5 contenant respectivement les multiples de 3 et les multiples de 5.

Exercice 3 :

Soient deux vecteurs d'entiers triés V1 (N entiers, N≤100) et V2 (M entiers, M≤150). Ecrire un algorithme qui construit un vecteur V3 composé des éléments de V1 qui n'existent pas dans V2.

Exercice 4 :

Soit T un tableau de N entiers (N≤100), et soient A et B deux éléments distincts appartenant à T. Ecrire un algorithme qui calcule la distance minimale entre A et B dans ce tableau.
Exemple:
Exemple
T= 3 - 18 - 90 - 7 - 10 - 7 - -40 - 3 - 100 - 3 - 90 - 3 - 20 - 10 - 50 - 7
Pour A=3 et B=7, la distance minimale est : 2

SOLUTION

1- Détermine le mini, le maxi et la moyenne des éléments d'un tableau T

```
Algorithme MinMaxMoyVecteur;
Var
  T:Tableau[1..100] de Entier;
  i, N, Min, Max, Som: Entier;
Debut
  Repeter
     Ecrire("Donner N compris entre 1 et 100");
     Lire(N);
  Jusqu'a ((N>=1) ET (N<=100));
  Pour i <--- 1 a N
  Faire
    Ecrire("Donner ", i ," eme element du Vecteur");
    Lire(T[i]);
  Fait

  /* On suppose au debut que le Min est le premier element du vecteur, de meme on suppose
que le maximum est le premier element du vecteur. */
  Min <--- T[1];
  Max <--- T[1];
  Som <--- T[1];
  Pour i <-- 2 a N
  Faire
     Som <--- Som + T[i];
     Si (T[i] > Max) alors
        Max <--- T[i];
     Sinon
        Si (T[i] < Min) alors
```

```
          Min <--- T[i];
        Fsi
      Fsi
  Fait

  Ecrire("Le Minimum du Vecteur est ", Min, " Le Maximum du Vecteur est ", Max, " et La
Moyenne est:", Som/N);
Fin
```

2- Calcule la somme et le produit scalaire de deux vecteurs (T1 et T2).

```
Algorithme  InverseVecteur;
Var
   T1, T2, T3:Tableau[1..100] de Entier;
   i, PScal, N: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=100));

   Pour i <--- 1 a N
   Faire
       Ecrire("Donner ", i, " eme valeur du vecteur T1");
       Lire(T1[i]);
   Fait

   Pour i <--- 1 a N
   Faire
       Ecrire("Donner ", i, " eme valeur du vecteur 2");
       Lire(T2[i]);
   Fait

   PScal <--- 0;
   Pour i <--- 1 a N
   Faire
        PScal <--- PScal + (T1[i] * T2[i]);
        T3[i] <--- T1[i] + T2[i];
   Fait

   Ecrire("Produit Scalaire = ", PScal);

   Ecrire("Vecteur Somme T3:");
   Pour i <--- 1 a N
   Faire
       Ecrire(T3[i]);
   Fait
Fin.
```

3- Inverse le contenu d’un vecteur T.

```
Algorithme  InverseVecteur;
Var
   T:Tableau[1..100] de Entier;
   i, J, N, Temp: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=100));

   Pour i <--- 1 a N
```

```
  Faire
     Ecrire("Donner ", i, " eme valeur du vecteur");
     Lire(T[i]);
  Fait

  i <--- 1;
  J <--- N;
  Tant Que (i < J)
  Faire
    /* Permuter T[i] avec T[J] */
    Temp <--- T[i];
    T[i] <--- T[J];
    T[J] <--- Temp;
    i <--- i + 1;
    J <--- J - 1;
  Fait

  Ecrire("Vecteur Inverse");
  Pour i <--- 1 a N
  Faire
     Ecrire(T[i]);
  Fait

Fin.
```

Solution 2: En utilisant une boucle pour. Il suffit de remarquer que dans la solution precedente on a toujours J = N - i + 1 et que i va au plus jusqu'a la moitie de N.

```
Algorithme  InverseVecteur;
Var
  T:Tableau[1..100] de Entier;
  i, N, Temp: Entier;
Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 100");
    Lire(N);
  Jusqu'a ((N>=1) et (N<=100));

  Pour i <--- 1 a N
  Faire
     Ecrire("Donner ", i, " eme valeur du veteur");
     Lire(T[i]);
  Fait

  Pour i <--- 1 a (N Div 2)
  Faire
    /* Permuter T[i] avec T[N-i+1] */
    Temp <--- T[i];
    T[i] <--- T[N - i + 1];
    T[N - i + 1] <--- Temp;
  Fait

  Ecrire("Vecteur Inverse");
  Pour i <--- 1 a N
  Faire
     Ecrire(T[i]);
  Fait

Fin.
```

4- Calcule le nombre d'occurrences d'une valeur V dans T

```
Algorithme    NbrOccValVecteur;;
Var
   T:Tableau[1..100] de Entier;
   i, N, nbOcc, val: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=100));

   Pour i <--- 1 a N
   Faire
       Ecrire("Donner ", i, " eme valeur du veteur");
       Lire(T[i]);
   Fait

   Ecrire("Donner Valeur de Val");
   Lire(val);

   nbOcc <--- 0;
   Pour i <--- 1  a  N
   Faire
      Si (T[i] = val) alors
          nbOcc <--- nbOcc + 1;
     Fsi
   Fait

   Ecrire("Nombre d'Occurrences de ", val, " dans T est:", nbOcc);
Fin.
```

5- Détermine la différence minimale entre deux éléments quelconques du vecteur T.
Si on a un ensemble de valeurs v1, v2, v3, ..., vn et que l'on cherche la plus grande valeur que peut avoir la difference (a - b) ou a est une quelconque des valeurs v1, v2, v3 , ... ou vn et de meme b est une quelconque des valeurs v1, v2, v3 , ... ou vn. Cette difference est la plus grande lorsque a prend la valeur la plus grande parmi (v1, v2, v3 , ..., vn) et b prend la valeur la plus petite parmi (v1, v2, v3 , ... vn), c'est a dire que a est le max de (v1, v2, ...vn) et b est le min de (v1, v2, ..., vn).
La meme chose la difference (a - b) a la valeur minimale lorsque a prend la valeur la plus petite parmi (v1, v2, v3 , ..., vn) et b prend la valeur la plus grande parmi (v1, v2, v3 , ... vn), c'est a dire que a est le min de (v1, v2, ...vn) et b est le max de (v1, v2, ..., vn).
Donc chercher la différence minimale entre deux éléments quelconques du vecteur T revient a chercher le minimum et le maximum de ce vecteur et faire la difference entre le minimum et le maximum.

```
Algorithme    diffMinVecteur;;
Var
   T:Tableau[1..100] de Entier;
   i, N, min, max: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 2 et 100");
      Lire(N);
   Jusqu'a ((N>=2) et (N<=100));

   Pour i <--- 1 a N
   Faire
       Ecrire("Donner ", i, " eme valeur du veteur");
       Lire(T[i]);
```

```
  Fait

  /* On suppose au debut que le Min est le premier element du vecteur, de meme on suppose
que le maximum est le premier element du vecteur. */
 min <--- T[1];
 max <--- T[1];
 Pour i <-- 2 a N
 Faire
   Si (T[i] > max) alors
     max <--- T[i];
   Sinon
     Si (T[i] < min) alors
       min <--- T[i];
     Fsi
   Fsi
 Fait

  Ecrire("La difference Minimale entre 2 elements quelconques de T est:", min - max);

Fin.
```

6- Supprime toutes les valeurs doubles d'un vecteur T.
Solution 1:
Idee: Parcourir le tableau T et des qu'on trouve un doublet decaler les elements qui suivent d'une position vers la gauche.

```
Algorithme     SuppValDoublesVecteur;
Var
  T:Tableau[1..100] de Entier;
  i, J, K, N: Entier;
Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 100");
    Lire(N);
  Jusqu'a ((N>=1) et (N<=100));

  Pour i <--- 1 a N
  Faire
     Ecrire("Donner ", i, " eme valeur du veteur");
     Lire(T[i]);
  Fait

  /* i va jusqu'a N - 1, le dernier (N eme) n'a pas de suivant donc il n'a pas de doublet qui le
suit */
  Pour   i <--- 1 a (N - 1)
  Faire
    J <--- i + 1;
    Tant Que (J <= N)
    Faire
       Si (T[i] <> T[J]) alors
         /* Si T[J] n'est pas egale a T[i] alors incrementer le J pour comparer avec le prochain
element */
         J <--- J + 1;
       Sinon
         /* T[i] = T[J] element double decaler tous elements de J+1 a N d'une position vers la
gauche */
         N <--- N - 1;
         Pour K <--- J a N
         Faire
            T[K] <--- T[K + 1];
```

```
            Fait

            /* Attention ici ne pas incrementer le J car le nouveau element a la position J
               peut etre encore un double de T[i]
             */
          Fsi
       Fait
    Fait

  Ecrire("Vecteur Sans Doublets");
  Pour i <--- 1 a N
  Faire
     Ecrire(T[i]);
  Fait

Fin.
```

Solution 2:
Idee: Parcourir le tableau T et ecraser les valeurs doubles de chaque element en mettant dans les positions des valeurs doubles les valeurs differentes de l'element courant.

```
Algorithme     SuppValDoublesVecteur;
Var
  T:Tableau[1..100] de Entier;
  i, J, K, nb, N: Entier;
Debut
  Repeter
     Ecrire("Donner N compris entre 1 et 100");
     Lire(N);
  Jusqu'a ((N>=1) et (N<=100));

  Pour i <--- 1 a N
  Faire
      Ecrire("Donner ", i, " eme valeur du veteur");
      Lire(T[i]);
  Fait

  /* i va jusqu'a N - 1,  le dernier (N eme) n'a pas de suivant donc il n'a pas de doublet qui le
suit */
  Pour   i <--- 1 a (N - 1)
  Faire
     K <--- i + 1;
     nb <--- 0;  /* nb compte le nombre de doublets de l'element courant T[i] */
     /* Parcourir le tableau a partir du prochain element de position (i + 1) */
     Pour J <--- (i + 1) a N
     Faire
        T[K] <--- T[J];
        Si (T[i] <> T[J]) alors
              /* Si T[J] n'est pas egale a T[i] alors incrementer le K car on veut que le prochain
T[J] qu'on recopie dans T n'ecrase pas la valeur courante T[J] */
              K <--- K + 1;
        Sinon
             nb <--- nb + 1;
             /* Attention ici ne pas incrementer le K car on veut ecraser les elements doubles */
         Fsi
      Fait
      N <--- N - nb;
   Fait

  Ecrire("Vecteur Sans Doublets");
```

```
  Pour i <--- 1 a N
  Faire
     Ecrire(T[i]);
  Fait
Fin.
```

Exercice 2 :
Soit T un vecteur de N (N≤250) entiers supposés positifs. Ecrire un algorithme permettant, à partir de T, de construire deux vecteurs M3 et M5 contenant respectivement les multiples de 3 et les multiples de 5.

```
Algorithme    M3M5Vecteurs;;
Var
  T, M3, M5:Tableau[1..250] de Entier;
  i, N, J, k: Entier;
Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 250");
    Lire(N);
  Jusqu'a ((N>=1) et (N<=250));

  Pour i <--- 1 a N
  Faire
     Ecrire("Donner ", i, " eme valeur du veteur");
     Lire(T[i]);
  Fait

  J <--- 0;
  k <--- 0;
  Pour i <--- 1  a  N
  Faire
    Si ((T[i] mod 3) = 0) alors
        J <--- J + 1;
       M3[J] <--- T[i];
   Fsi
   Si ((T[i] mod 5) = 0) alors
        k <--- k + 1;
       M5[k] <--- T[i];
   Fsi
  Fait

  Si (J = 0) alors
      Ecrire("Vecteur M3 est Vide");
  sinon
      Ecrire("Vecteur M3");
      Pour i <--- 1 a J Faire
          Ecrire(M3[i]);
     Fait
  Fsi
  Si (k = 0) alors
      Ecrire("Vecteur M5 est Vide");
  sinon
      Ecrire("Vecteur M5");
      Pour i <--- 1 a k Faire
          Ecrire(M5[i]);
     Fait
  Fsi

Fin.
```

Exercice 3 :
Soient deux vecteurs d'entiers triés V1 (N entiers, N≤100) et V2 (M entiers, M≤150). Ecrire un algorithme qui construit un vecteur V3 composé des éléments de V1 qui n'existent pas dans V2..

```
Algorithme      V3EltsV1NonDansV2;
Var
  V1, V3:Tableau[1..100] de Entier;
  V2:Tableau[1..150] de Entier;
  i, J, K, N, M:Entier;
Debut
  Repeter
     Ecrire("Donner N compris entre 1 et 100");
     Lire(N);
  Jusqu'a ((N>=1) ET (N<=100));

  Repeter
     Ecrire("Donner M compris entre 1 et 150");
     Lire(M);
  Jusqu'a ((M>=1) ET (M<=150));

  // On suppose dans cette boucle Pour les elements de V1 sont donnes Tries par ordre
croissant
  Pour i <--- 1 a N
  Faire
    Ecrire("Donner ", i ," eme element du Vecteur V1 Trie");
    Lire(V1[i]);
  Fait

  // On suppose dans cette boucle Pour les elements de V1 sont donnes Tries par ordre
croissant
  Pour i <--- 1 a M
  Faire
    Ecrire("Donner ", i ," eme element du Vecteur V2 Trie");
    Lire(V2[i]);
  Fait

  K <--- 0;
  Tant Que (i <= N)
  Faire
      Si (V1[i] > V2[M])  alors
           // Puisque V1 est Trie par ordre croissant alors tous les elements de V1 a
           // a partir de i sont > V2[M] et donc ne peuvent pas exister dans V2 et dans ce
           // il faut recopier tous ces elements dans V3
           Tant Que (i <= N)
            Faire
                  K <-- K + 1;
                  V3[K] <--- V1[i];
                  i <-- i + 1;
            Fait
      Sinon
           Si (V1[i] < V2[1])  alors
               // V1[i] est < a tous les elements de V2 puisque V2 est trie par ordre croissant et
               // dans ce cas V1[i] ne peut pas exister dans V2
               K <-- K +1;
               V3[K] <--- V1[i];
               i <-- i + 1;
           Sinon    // Cas ou  V2[1]<= V1[i] <= V2[M], rechercher V1[i] dans V2
                    Trouve <--- Faux;
                    J <--- 1;
```

```
                    Tant Que ((J <= M) ET (Trouve = False) ET (V1[i] >= V2[J]))
                     Faire
                          Si (V1[i] = V2[J]) alors
                                    Trouve <--- Vrai;
                          Sinon
                                    J <--- J + 1;
                         Fsi
                     Fait
                     Si (Trouve = Faux) alors
                           K <--- K + 1;
                           V3[K] <--- V1[i];
                     Fsi
                     i <-- i + 1;
              Fsi
         Fsi
    Fait

    Si (K = 0) alors
          Ecrire("Vecteur V3 des Elements de V1 ne se trouvant pas dans V2 est Vide");
    Sinon
          Ecrire("Vecteur des Elements de V1 ne se trouvant pas dans V2:");
          Pour i <--- 1 a K
          Faire
               Ecrire(V3[i]);
          Fait
     Fsi
Fin.
```

Exercice 4 :

Soit T un tableau de N entiers (N<=100), et soient A et B deux éléments distincts appartenant à T. Ecrire un algorithme qui calcule la distance minimale entre A et B dans ce tableau.
Exemple:
T= 3 - 18 - 90 - 7 - 10 - 7 - -40 - 3 - 100 - 3 - 90 - 3 - 20 - 10 - 50 - 7
Pour A=3 et B=7, la distance minimale est : 2

Remarquons que la plus grande distance possible entre A et B est lorsque A est T[1] et B est T[N] (ou bien B est T[1] et A est T[N]) et dans cas la distance est (N - 1). Donc on vas initialiser distMin a une valeur plus grande (par exemple N).

Solution 1:
Idee: Parcourir le tableau T et pour chaque element de T d'indice i (sauf le dernier) parcourir T a partir de (i + 1) en utilisant un indice J et la distance est (J - i) et si T[i] est A et T[J] est B (ou bien T[i] est B et T[J] est A) alors si cette distance est plus petite que la distance minimale courante alors elle devient la distance minimale.

```
Algorithme    DistMinVecteurs;;
Var
  T:Tableau[1..100] de Entier;
  i, N, J, distMin, dist, A, B: Entier;
  existe : Booleen;
Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 100");
    Lire(N);
  Jusqu'a ((N>=1) et (N<=100));

  Pour i <--- 1 a N
  Faire
     Ecrire("Donner ", i, " eme valeur du veteur");
     Lire(T[i]);
  Fait
```

```
// forcer l'utilisateur a donner un A appartenant a T
Repeter
      Ecrire("Donner Valeur de A appartenant a T");
      Lire(A);
      existe <--- Faux;
      i <--- 1;
      Tant Que ((i <= N) et (existe = Faux))  Faire
            Si (T[i] = A) alors
                existe <--- Vrai;
            Sinon
                i <--- i + 1;
            Fsi
       Fait
 Jusqu'a (existe = Vrai);

// forcer l'utilisateur a donner un B appartenant a T et distinct de A
Repeter
      Ecrire("Donner Valeur de B appartenant a T et distinct de ", A);
      Lire(B);
      existe <--- Faux;
      Si (B <> A) alors
           i <--- 1;
          Tant Que ((i <= N) et (existe = Faux))  Faire
                Si (T[i] = B) alors
                     existe <--- Vrai;
                Sinon
                      i <--- i + 1;
                Fsi
          Fait
      Fsi
 Jusqu'a (existe = Vrai);

distMin <--- N; /* max distMin = (N - 1)  initialiser distMin a une valeur plus grande (N)*/
Pour i <--- 1 a (N - 1)
Faire
       Pour J <--- (i+1) a N
       Faire
            Si ( ((T[i] = A) et (T[J] = B)) OU ((T[i] = B) et (T[J] = A)) ) alors
                 dist <--- J - i;
                 Si (dist < distMin) alors
                       distMin <--- dist;
                 Fsi
            Fsi
       Fait
Fait

   Ecrire("La distance Minimale entre", A, " et ", B, " est:",  distMin);
Fin.
```

Solution 2:
Idee: Parcourir le tableau T jusqu'a trouver la 1ere position de A ou B en utilisant un indice i., sauvegarder cette position dans un indice k. A partir de cette 1ere position et jusqu'au dernier element chercher A ou B en utilisant un indice J. Si on trouve un element A ou B (qui est T[J]) et si cet element est different de T[k] alors calculer la distance J-k et voir si cette distance est plus petite que la distance minimale courante. Aussi si on a trouve A ou B ne pas oublier de mettre a jour k par J (et ce pour que (T[J]<>T[k]) ou bien (T[J]=T[k])).
Remarquons que dans la solution 1 pour chaque element de T on parcourt T a partir du suivant jusqu'au dernier element de T. Dans cette 2eme solution on parcourt T une seule fois.

```
Algorithme    DistMinVecteurs;;
Var
   T:Tableau[1..100] de Entier;
   i, N, J, distMin, dist, A, B: Entier;
   existe : Booleen;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=100));

   Pour i <--- 1 a N
   Faire
       Ecrire("Donner ", i, " eme valeur du veteur");
      Lire(T[i]);
   Fait

    // forcer l'utilisateur a donner un A appartenant a T
    Repeter
          Ecrire("Donner Valeur de A appartenant a T");
          Lire(A);
          existe <--- Faux;
          i <--- 1;
          Tant Que ((i <= N) et (existe = Faux))  Faire
                Si (T[i] = A) alors
                     existe <--- Vrai;
                Sinon
                     i <--- i + 1;
                Fsi
           Fait
     Jusqu'a (existe = Vrai);

    // forcer l'utilisateur a donner un B appartenant a T et distinct de A
    Repeter
          Ecrire("Donner Valeur de B appartenant a T et distinct de ", A);
          Lire(B);
          existe <--- Faux;
          Si (B <> A) alors
               i <--- 1;
              Tant Que ((i <= N) et (existe = Faux))  Faire
                    Si (T[i] = B) alors
                         existe <--- Vrai;
                    Sinon
                          i <--- i + 1;
                    Fsi
              Fait
          Fsi
     Jusqu'a (existe = Vrai);

    distMin <--- N; /* max distMin = (N - 1)  initialiser distMin a une valeur plus grande (N)*/

     // chercher la premiere position de A ou B
    i <---  1;
    Tant Que ( (i < N) et (T[i]  <> A) et (T[i] <> B) )
     Faire
          i <--- i + 1;
     Fait
     k <--- i;

    // rechercher les prochaines occurrences de A ou B et si T[J] <> T[k] alors calculer
```

```
   // la distance
   Pour J <--- i + 1 a N
   Faire
      Si  ((T[J] = A) OU (T[J] = B))  alors
         Si (T[J] <> T[k])  alors
            dist <--- J - k;
            Si (dist < distMin)
               distMin <--- dist;
            Fsi
          Fsi
         k <--- J;   // mettre a jour k
      Fsi
    Fait

   Ecrire("La distance Minimale entre", A, " et ", B, " est:",  distMin);
Fin.
```

Partie 2 : Les matrices

Exercice 5 :
Soit une matrice A(N, M) de caractères (N≤20 et M≤30). Ecrire un algorithme qui
1- Calcule le nombre de voyelles appartenant à la matrice A.
2- Fait une rotation des colonnes de la matrice A.

Exercice 6 :
Soit une matrice carrée A(N, N) d'entiers (N≤25). Ecrire un algorithme qui
1- Calculer la trace de la matrice A. (La trace est la somme des éléments de la diagonale principale).
2- Déterminer le maximum et sa position, des valeurs des deux diagonales (principale et secondaire).

Exercice 7 :
Soit une matrice A(N, M) d'entiers (N≤20 et M≤30), écrire un algorithme qui :
- Calcule et sauvegarde la somme de chaque colonne,
- Détermine la position Jmin de la somme minimale et la position Jmax de la somme maximale.
- Permute les deux colonnes d'indices Jmin et Jmax de la matrice A si Jmin > Jmax.

Exercice 8 :
Ecrire un algorithme qui affiche le triangle de Pascal de degré N. (N étant un en entier positif).
Exemple
pour N=4

```
1
1   1
1   2    1
1   3    3    1
1   4    6    4    1
```

Exercice 5 : Soit une matrice A(N, M) de caractères (N=20 et M=30).
Ecrire un algorithme qui
1- Calcule le nombre de voyelles appartenant à la matrice A.

```
Algorithme  nbrVoyMatrice;
Var
   A:Tableau[1..20, 1..30] de Caractere;
   i, J, N, M, nbrVoy: Entier;
Debut
```

```
  Repeter
    Ecrire("Donner N compris entre 1 et 20");
    Lire(N);
  Jusqu'a ((N>=1) et (N<=20));

  Repeter
    Ecrire("Donner M compris entre 1 et 30");
    Lire(M);
  Jusqu'a ((M>=1) et (M<=30));

  Pour i <--- 1 a N
  Faire
     Pour J <--- 1 a M
     Faire
        Ecrire("Donner l'element de la matrice d'indices ", i, "et", J);
        Lire(A[i, J]);
     Fait
  Fait
  nbrVoy <--- 0;
  Pour i <--- 1 a N
  Faire
     Pour J <--- 1 a M
     Faire
        Cas A[i, J]  vaut
          'a', 'e', 'i', 'u', 'o', 'y', 'A', 'E', 'I', 'U', 'O', 'Y': nbrVoy <--- nbrVoy + 1;
        FCas
     Fait
  Fait
  Ecrire("Nombre de Voyelles de la matrice :", nbrVoy);
Fin.
```

2- Fait une rotation des colonnes de la matrice A.

```
Algorithme  rotationMatrice;
Var
  A:Tableau[1..20, 1..30] de Caractere;
  T: Tableau[1..20] de Caractere;
  i, J, N, M: Entier;
Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 20");
    Lire(N);
  Jusqu'a ((N>=1) et (N<=20));

  Repeter
    Ecrire("Donner M compris entre 1 et 30");
    Lire(M);
  Jusqu'a ((M>=1) et (M<=30));

  Pour i <--- 1 a N
  Faire
     Pour J <--- 1 a M
     Faire
        Ecrire("Donner l'element de la matrice d'indices ", i, "et", J);
        Lire(A[i, J]);
     Fait
  Fait
  /* On fait une rotation des colonnes vers la droite; donc on va commencer par sauvegarder
la derniere colonne
     dans le vecteur T */
```

```
  Pour i <--- 1 a N
  Faire
     T[i] <--- A[i, M];
  Fait

  /* Faisons maintenant un decalager a droite de toutes les colonnes en commencant par
     la colonne M - 1 jusqu'a la premiere colonne */
  J <--- M - 1;
  Tant Que (J >= 1)
  Faire
     Pour i <--- 1 a N
     Faire
        A[i, J + 1] <--- A[i, J];
     Fait
     J <--- J - 1;
  Fait
  /* Mettons maintenant le table T en 1ere colonne */
  Pour i <--- 1 a N
  Faire
     A[i, 1] <--- T[i];
  Fait
  Ecrire("Matrice A:");
  Pour i <--- 1 a N
  Faire
     Pour J <--- 1 a M
     Faire
        Ecrire(A[i, J]);
     Fait
  Fait
Fin.
```

Exercice 6 : Soit une matrice carrée A(N, N) d'entiers (N=25). Ecrire un algorithme qui
1- Calculer la trace de la matrice A. (La trace est la somme des éléments de la diagonale principale).

```
Algorithme  traceMatrice;
Var
  A:Tableau[1..25, 1..25] de Entier;
  i, J, N, trace: Entier;
Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 25");
    Lire(N);
  Jusqu'a ((N>=1) et (N<=25));

  Pour i <--- 1 a N
  Faire
     Pour J <--- 1 a N
     Faire
        Ecrire("Donner l'element de la matrice d'indices ", i, "et", J);
        Lire(A[i, J]);
     Fait
  Fait
  trace <--- 0;
  Pour i <--- 1 a N
  Faire
     trace <--- trace + A[i, i];
  Fait
```

```
   Ecrire("Trace de la matrice = ", trace);
Fin.
```

2- Déterminer le maximum et sa position, des valeurs des deux diagonales (principale et secondaire).

```
Algorithme    posMaxDiagsMatrice;
Var
   A:Tableau[1..25, 1..25] de Entier;
   i, J, N, Imax, Jmax: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 25");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=25));

   Pour i <--- 1 a N
   Faire
      Pour J <--- 1 a N
      Faire
         Ecrire("Donner l'element de la matrice d'indices ", i, "et", J);
         Lire(A[i, J]);
      Fait
   Fait
   /* On suppose au debut que le max c'est l'element d'indices 1, 1 de la matrice */
   Imax <--- 1;
   Jmax <--- 1;

   J <--- N;
   Pour i <--- 1 a N
   Faire
      Si (A[i, i] > A[Imax, Jmax]) alors
         Imax <--- i;
         Jmax <--- i;
      Fsi
      Si (A[i, J] > A[Imax, Jmax]) alors
         Imax <--- i;
         Jmax <--- J;
      Fsi
      J <-- J - 1;
    Fait
   Ecrire("Max des 2 diagonales = ", A[Imax, Jmax], " et positions ", Imax, Jmax);
Fin.
```

Exercice 7 :

Soit une matrice A(N, M) d'entiers (N≤20 et M≤30), écrire un algorithme qui :
- Calcule et sauvegarde la somme de chaque colonne,
- Détermine la position Jmin de la somme minimale et la position Jmax de la somme maximale.
- Permute les deux colonnes d'indices Jmin et Jmax de la matrice A si Jmin > Jmax.

```
Algorithme  somColonnes;
Var
   A:Tableau[1..20, 1..30] de Entier;
   SCol:Tableau[1..30] de Entier;
   i, J, N, M, Temp: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 20");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=20));
```

```
  Repeter
    Ecrire("Donner M compris entre 1 et 30");
    Lire(M);
  Jusqu'a ((M>=1) et (M<=30));

  Pour i <--- 1 a N
  Faire
     Pour J <--- 1 a M
     Faire
        Ecrire("Donner l'element de la matrice d'indices ", i, "et", J);
        Lire(A[i, J]);
     Fait
  Fait
  /* Somme de chaque colonne */
  Pour J <--- 1 a M
  Faire
     SCol[j] <--- 0;
     Pour i <--- 1 a N
     Faire
        SCol[J] <--- SCol[J] + A[i, J];
     Fait
  Fait

  JMin <--- 1;
  JMax <--- 1;
  Pour J <--- 2 a M
  Faire
     Si (SCol[j] > SCol[JMax]) alors
        JMax <--- J;
     Fsi
     Si (SCol[j] < SCol[JMin]) alors
        JMin <--- J;
     Fsi
  Fait
  Si (JMin > JMax) alors
    /* Permuter colonne JMin et JMax de la matrice */
    /* sauvegarder colonne JMin dans saveA */
    Pour   i <--- 1 a N
    Faire
       Temp <--- A[i, JMin];
       A[i, JMin] <--- A[i, JMax];
       A[i, JMax] <--- Temp;
    Fait
  Fsi

  /* Afficher la matrice */
  Pour   i <--- 1 a N
  Faire
     Pour   J <--- 1 a M
     Faire
        Ecrire(A[i, J]);
     Fait
  Fait
Fin.
```

Exercice 8 :

Ecrire un algorithme qui affiche le triangle de Pascal de degré N. (N étant un en entier positif).

Exemple pour N=4

```
1
1   1
1   2   1
1   3   3    1
1   4   6   4   1
```

On remarque que pour N on doit utiliser une matrice caree (N+1) x (N+1). Aussi tous les elements a la 1ere colonne A[*, 1] sont a 1 et les elements dans la diagonale principale A[i, i] sont a 1 aussi. Les autres elements A[i, J] ou J varie de 2 a (i - 1) sont A[i-1, J-1] + A[i - 1, J].

```
Algorithme    TriangleDePascale;
Var
   A:Tableau[1..100, 1..100] de Entier;
   i, J, N: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 99");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=99));

   Pour i <--- 1 a (N + 1)
   Faire
      A[i, 1] <--- 1;
      A[i, i] <--- 1;
      Pour J <--- 2 a (i - 1)
      Faire
         A[i, J] = A[i - 1, J-1] + A[i - 1, J];
      Fait
   Fait

   Ecrire("Triangle de Pascal:")
   Pour i <--- 1 a (N + 1)
   Faire
      Pour J <--- 1 a i
      Faire
         Ecrire(A[i, J]);
      Fait
   Fait
Fin.
```

Partie 3 : Les chaînes de caractères

Exercice 9:

Ecrire un algorithme permettant de comptabiliser le nombre de caractères majuscules dans une chaîne.

Exercice 10 :

Ecrire un algorithme qui vérifie si une chaîne est un carré ou pas.
Définition : Une chaîne de caractères est un carré si elle se compose de 2 chaînes identiques.
Exemple : "chercher" et "bonbon" sont des carrés.

Exercice 11 :

Soit S une chaine da caractères. Ecrire un algorithme permettant de nettoyer la chaine S de tous les caractères non alphabétiques mais en laissant les espaces (le caractère blanc).
Exemple :

S= "*/ Le modul(@e Al?!gor> <ithm&ique est °$ attirant."
S nettoyée= "Le module Algorithmique est attirant"

Exercice 12 :
Soit S une chaîne de caractères. Un début non vide de S est un mot formé des i premiers caractères
de S, avec i = {1,2,…, Taille(S)}. Par exemple, si S = aababa, alors les débuts de S sont les mots {a, aa, aab, aaba, aabab, aababa}.
Ecrire un algorithme qui affiche tous les débuts non vides d'un mot donné S ;

Exercice 13 :
Soit S une chaîne de caractères constituant une suite de mots séparés par un ou plusieurs blancs
(espaces).
Ecrire un algorithme permettant d'inverser les mots de S sans toucher aux blanc.
Exemple :
S = " Alger est la capitale "
donne
S = " reglA tse al elatipac "

Exercice 14 :
Ecrire un algorithme qui lit deux mots et qui détermine s'ils sont anagrammes.
Sachant qu'un mot est dit anagramme d'un autre mot s'ils utilisent (sont formés par) les même lettres.

Exemples :
CHIEN anagramme de CHINE, NICHE,
GELER n'est pas anagramme d' ALGER, …

SOLUTION

Exercice 9:
Ecrire un algorithme permettant de comptabiliser le nombre de caractères majuscules dans une
chaîne.

```
Algorithme     CarMajuscules;
Var
    S : Chaine;
    i, J, T, nbrMaj : Entier;

Debut
    Ecrire("Donner la chaine de Caracteres  S");
    Lire(S);

    nbrMaj <-- 0;
    T <-- Taille(S);
    Pour i <-- 1 a T  Faire
       Si ((T[i] >= 'A') et (T[i] <= 'Z'))  alors
          nbrMaj <-- nbrMaj + 1;
       Fsi
    Fait

    Ecrire("Nombre de caracteres majuscules = ", nbrMaj);
Fin.
```

Exercice 10 :
Ecrire un algorithme qui vérifie si une chaîne est un carré ou pas.
Définition : Une chaîne de caractères est un carré si elle se compose de 2 chaînes identiques.
Exemple : "chercher" et "bonbon" sont des carrés.

```
Algorithme     ChaineCarre;
```

```
Var
    S : Chaine;
    i, J, T : Entier;
    chCarre : Booleen;

Debut
    Ecrire(“Donner la chaine de Caracteres  S”);
    Lire(S);

    T <-- Taille(S);
    Si (T mod 2 = 1) alors
          // si taille de S est impair ca ne peut pas etre une chaine carre
       chCarre <-- Faux;
    Sinon
       chCarre <-- Vrai;
       i <-- 1;
       J <-- (T Div 2) + 1;
       Tant Que ((i <= T Div 2) et chCarre) Faire
          Si (S[i] <> S[J]) alors
             chCarre <-- Faux;
          Sinon
            i <-- i + 1;
            J <-- J + 1;
          Fsi
       Fait
    Fsi

    Si (chCarre) alors
      Ecrire(S, " est une chaine carre");
    sinon
      Ecrire(S, " N'est Pas une chaine carre");
    Fsi
Fin.
```

Exercice 11 :

Soit S une chaine da caractères. Ecrire un algorithme permettant de nettoyer la chaine S de tous les caractères non alphabétiques mais en laissant les espaces (le caractère blanc).
Exemple :

S= "*/ Le modul(@e Al?!gor> <ithm&ique est °$ attirant."
S nettoyée= "Le module Algorithmique est attirant"

Solution 1:
Utiliser une autres chaine S2 et recopier tous les caracteres alpabetiques et les blancs a partir du 1er caractere alphabetique

```
Algorithme     ChaineNettoyee;
Var
    S, S2 : Chaine;
    i, J, k, T : Entier;

Debut
    Ecrire(“Donner la chaine de Caracteres  S”);
    Lire(S);

    T <-- Taille(S);
    S2 <-- ""; // mettre chaine vide dans S2

    // chercher le 1er caractere alphabetique de S
    i <-- 1;
```

```
    Tant Que ((i <= T) et (Non(  (S[i]>='a') et (S[i]<='z'))  OU  ( (S[i]>='A') et (S[i]<='Z')) )) )
    Faire
      i <-- i + 1;
    Fait

    J <-- 0;
    Pour k <-- i a T
    Faire
          // recopier les caracteres alphabetiques et les blancs dans la chaine S2
          Si ( ((S[k]>='a') et (S[k]<='z')) OU  ((S[k]>='A') et (S[k]<='Z')) OU (S[k] = ' ') )  alors
             J <-- J + 1;
             S2[J] <-- S[k];
          Fsi
    Fait

    ChangerTaille(S2, J);
    S <-- S2;
    Ecrire("S Nettoyee : ", S);
Fin.
```

Solution 2:
Idee: Supprimer les caracteres non alphabetiques et les 1er blancs en faisant des decalages.

```
Algorithme     ChaineNettoyee;
Var
    S : Chaine;
    i, J, T : Entier;

Debut
    Ecrire(“Donner la chaine de Caracteres  S”);
    Lire(S);

    T <-- Taille(S);
    i <-- 1;

    // Tant qu'on n'a pas atteint le 1er caractere alphabetique faire des decalages pour
    // eliminer ces caracteres non alphabetiques (ramener le 1er caractere alphabetique
    // en 1ere position)
    Tant Que ((i <= T) ET ( NON ( ((S[i]>='a') et (S[i]<='z')) OU ((S[i]>='A') et (S[i]<='Z'))) ) )
    Faire
      T <--- T - 1;
      Pour  J <-- i  a  T
      Faire
            S[J] = S[J + 1];
      Fait
   Fait

    Tant Que (i <= T)
    Faire
       // si le caractere est alphabetique ou bien c'est un blanc alors
       // ne pas faire de decalage et dans ce cas incrementer le i
       Si  ( ((S[i]>='a') et (S[i]<='z')) OU  ( (S[i]>='A') et (S[i]<='Z')) OU (S[i] = ' ') ) alors
          i <-- i + 1;
       Sinon
          // faire decalages
          T <-- T - 1;
          Pour J <-- i a T Faire
             S[J] <-- S[J + 1];
          Fait
       Fsi
```

```
    Fait

    ChangerTaille(S, T);
    Ecrire("S Nettoyee : ", S);
Fin.
```

Solution 3:
Idee: supprimer les caracteres non alphabetiques et les premier blancs en ecrasant ces caracteres par les caracteres alphabetiques.

```
Algorithme     ChaineNettoyee;
Var
    S : Chaine;
    i, J, T : Entier;
Debut
    Ecrire(“Donner la chaine de Caracteres  S”);
    Lire(S);

    T <-- Taille(S);
    i <-- 1;
    // chercher l'indice du 1er caractere alphabetique de S
    i <-- 1;
    Tant Que ((i <= T) et (Non(  (S[i]>='a') et (S[i]<='z'))  OU  ( (S[i]>='A') et (S[i]<='Z')) )) )
    Faire
      i <-- i + 1;
    Fait

     // A partir de la position du 1er caractere alphabetique de S ecraser les caracteres
    // autres que les caracteres alphabetiques et les blancs
    J <-- 1;
    Pour k <-- i a T
    Faire
      S[J] <-- S[k];
      // Si le caractere n'est pas un caractere alphabetique ou bien n'est pas un Blanc
      // alors Ne PAS incrementer le J pour ecraser ce caractere
      Si (   ((S[k]>='a') et (S[k]<='z')) OU  ((S[k]>='A') et (S[k]<='Z'))  OU (S[k] = ' ') ) alors
            J <-- J + 1;
      Fsi
    Fait
    T <-- J - 1;
    ChangerTaille(S, T);
    Ecrire("S Nettoyee : ", S);
Fin.
```

Exercice 12 :
Soit S une chaîne de caractères. Un début non vide de S est un mot formé des i premiers caractères de S, avec i = {1,2,…, Taille(S)}. Par exemple, si S = aababa, alors les débuts de S sont les mots {a, aa, aab, aaba, aabab, aababa}.
Ecrire un algorithme qui affiche tous les débuts non vides d'un mot donné S ;

```
Algorithme     ChaineDebuts;
Var
    S, SDeb : Chaine;
    i, J, T : Entier;

Debut
    Ecrire(“Donner la chaine de Caracteres S”);
    Lire(S);

    T <-- Taille(S);
```

```
    Pour i <-- 1 a T Faire
       SDeb <-- ""; // initialiser SDeb a la chaine vide
       Pour J <-- 1 a i Faire
          SDeb[J] <-- S[i];
       Fait
       Ecrire("Debut de S avec ", i, " caracteres est:", SDeb);
    Fait
Fin.

Fin.
```

Exercice 13 :

Soit S une chaîne de caractères constituant une suite de mots séparés par un ou plusieurs blancs
(espaces).
Ecrire un algorithme permettant d'inverser les mots de S sans toucher aux blancs.
Exemple :
S = " Alger est la capitale "
donne
S = " reglA tse al elatipac "

```
Algorithme     ChaineMotsInverses;
Var
    S : Chaine;
    i, J, T, iDeb, iFin : Entier;
    Temp : Caractere;

Debut
    Ecrire("Donner la chaine de Caracteres S");
    Lire(S);

    T <-- Taille(S);
    i <-- 1;
    Tant Que (i <= T)
    Faire
      // sauter les blancs pour detecter le debut d'un mot
      Tant Que ((i <= T) et (S[i] = ' '))
      Faire
         i <-- i + 1;
      Fait
      Si (i <= T) alors
        iDeb <-- i;
        // detecter la fin du mot
        Tant Que ((i <= T) et (S[i] <> ' '))
        Faire
           i <-- i + 1;
        Fait
        iFin <-- i - 1;

        // le mot est compris entre iDeb et iFin
        Tant Que (iDeb < iFin) Faire
          Temp <-- S[iDeb];
          S[iDeb] <-- S[iFin];
          S[iFin] <-- Temp;
          iDeb <-- iDeb + 1;
          iFin <-- iFin - 1;
        Fait
      Fsi
    Fait
```

```
    Ecrire("Chaine S avec mots inverses est: ", S);
Fin.
```

Exercice 14 :

Ecrire un algorithme qui lit deux mots et qui détermine s'ils sont anagrammes.
Sachant qu'un mot est dit anagramme d'un autre mot s'ils utilisent (sont formés par) les même lettres.

Exemples :

CHIEN anagramme de CHINE, NICHE,
GELER n'est pas anagramme d' ALGER, …

Idee: Les 2 mots sont anagrammes si chaque caractere apparait dans les 2 mots le meme nombre fois, c'st a dire que le nombre d'occurrences de chaque caractere est le meme dans les 2 mots.

```
Algorithme    MotsAnagrammes;
Var
  Mot1,Mot2 :Chaine;
  T1,T2,i,nb1,nb2 :Entier;
  car :Caractere;
  Anagram :Booleen;

Debut
  Ecrire("Donner le premier mot");
  Lire(Mot1);
  Ecrire("Donner le deuxieme mot");
  Lire(Mot2);
  T1 <-- Taille(Mot1);
  T2 <-- Taille(Mot2);

  Anagram <-- Faux ;
  Si (T1 = T2)   alors
    Anagram <-- Vrai;
    i <-- 1 ;
    Tant Que ((i <= T1) et Anagram)  Faire
      car <-- Mot1[i];
      nb1 <-- 0;
      Pour J <-- 1 a T1 Faire
        Si (Mot1[J] = car) alors
          nb1 <-- nb1 + 1 ;
        Fsi
      Fait
      nb2 <-- 0;
      Pour J <-- 1 a T1 Faire
        Si (Mot2[J] = car) alors
          nb2 <-- nb2 + 1 ;
        Fsi
      Fait
      Si (nb1 <> nb2) alors
        anagram <-- Faux;
      Fsi
      i <-- i + 1;
    Fait
  Fsi
  Si (Anagram) alors
     Ecrire("Les deux mots sont anagrammes")
  Sinon
     Ecrire("Les deux mots ne sont pas anagrammes")
  Fsi
```

Fin.

Exercices Complémentaires

Exercice 1 :

Soit un vecteur T (tableau à une dimension) contenant N nombres entiers (N≤100). Ecrire un algorithme qui:
1- Calcule le produit des éléments non nuls de T ainsi que le nombre de valeurs strictement positives.
2- Supprime toutes les valeurs nulles d'un vecteur T.
3- Met les valeurs négatives au début ensuite les valeurs positives à la fin en utilisant un seul tableau.

1- Calcule le produit des éléments non nuls de T ainsi que le nombre de valeurs strictement positives.

```
Algorithme     ProdNonNulsNbrPosVect;
Var
   T:Tableau[1..100] de Entier;
   i, N, nbPos, nbNonNul, Prod: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=100));

   Pour i <--- 1 a N
   Faire
       Ecrire("Donner ", i, " eme valeur du vecteur");
       Lire(T[i]);
   Fait

   nbPos <--- 0;
   nbNonNul <--- 0;
   Prod <--- 1;
   i <--- 1;
   Pour i <--- 1 a N
   Faire
        Si (T[i] <> 0) alors
             nbNonNul <--- nbNonNul + 1;
             Prod <--- Prod * T[i];
             Si (T[i] > 0) alors
                  nbPos <--- nbPos + 1;
             Fsi
        Fsi
   Fait

   Si (nbNonNul = 0) alors
         Ecrire("Vecteur a toutes les Valeurs Nulles");
   Sinon
        Ecrire("Produit des Valeurs non Nulles = ", Prod);
   Fsi
   Ecrire("Nombre de Valeurs strictement positif est:", nbPos);
Fin.
```

2- Supprime toutes les valeurs nulles d'un vecteur T.
Solution 1:
Decaler toutes les valeurs apres la valeur nulle d'une position vers la gauche

```
Algorithme  SuppEltsNulsVect;
Var
   T:Tableau[1..100] de Entier;
   i, J, N: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=100));

   Pour i <--- 1 a N
   Faire
       Ecrire("Donner ", i, " eme valeur du vecteur");
       Lire(T[i]);
   Fait

   i <--- 1;
   Tant Que (i <= N)
   Faire
       Si (T[i] = 0) alors
            /* T[i] = 0 decaler tous elements de i+1 a N d'une position vers la gauche */
            Pour J <--- i + 1 a N
            Faire
                 T[J - 1] <--- T[J];
            Fait
            N <--- N - 1;
            /* Attention ici ne pas incrementer le i car le nouveau element a la position i
              peut etre nul
             */
      Sinon
           i <--- i + 1;
      Fsi
   Fait

   Si (N = 0) alors
      Ecrire("Vecteur Sans Valeur Nulle Vide");
   Sinon
      Ecrire("Vecteur Sans Valeur Nulle ");
      Pour i <--- 1 a N
      Faire
          Ecrire(T[i]);
      Fait
   Fsi

Fin.
```

Solution 2:
Recopier tous les elements V[i] dans V[J] mais si V[i] est = 0 ne pas incrementer le J de maniere
a ecraser les valeurs nulles. Cette solution est plus simple que la solution 1.

```
Algorithme  SuppEltsNulsVect;
Var
   T:Tableau[1..100] de Entier;
   i, J, N: Entier;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
```

```
  Jusqu'a ((N>=1) et (N<=100));

  Pour i <--- 1 a N
  Faire
     Ecrire("Donner ", i, " eme valeur du vecteur");
     Lire(T[i]);
  Fait

  J <--- 1;
  Pour i  <--- 1 a N
  Faire
     T[J] <--- T[i];
     /* Incrementer le J uniquement si T[i] est <> 0.
        Si T[i]=0 alors J n'est pas incremente et dans ce cas T[J] sera ecrase par le prochain
T[i]
      */
     Si (T[i] <> 0) alors
        J <--- J + 1;
     Fsi
  Fait

  Si (J = 1) alors
    Ecrire("Vecteur Sans Valeur Nulle Vide");
  Sinon
    Ecrire("Vecteur Sans Valeur Nulle ");
    Pour i <--- 1 a (J - 1)
    Faire
       Ecrire(T[i]);
    Fait
  Fsi

Fin.
```

3- Met les valeurs négatives au début ensuite les valeurs positives à la fin en utilisant un seul tableau.

Solution 1:
Idee: permuter les valeurs positives avec les valeurs negatives.
L'ordre des valeurs negatives est maintenue mais dans certains cas l'ordre des valeurs positives n'est pas maintenue.
Par exemple:
T: -5, -8, 9, 12, 15, -11, -23
produit
T: -5, -8, -11, -23, 15, 9, 12

```
Algorithme     NegPuisPos ;
Var
  T :Tableau[1..100] de entier ;
  i,J,N, Temp:entier ;
Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 100");
    Lire(N);
  Jusqu'a ((N>=1) et (N<=100));

  Pour i <--- 1 a N
  Faire
     Ecrire("Donner ", i, " eme valeur du vecteur");
     Lire(T[i]);
  Fait
```

```
  /* Chercher la 1ere valeur >=0 */
  i <--- 1;
  Tantque  ((i < N) et (T[i]<0))
  Faire
    i  <--- i + 1;
  Fait ;
  /* i est utilise comme indice des valeurs positives et J comme indice des valeurs negatives.
     Déplacer les valeurs négatives au début en permutant T[i] valeur positive avec T[J]
valeur negative.
   */
  J <--- i + 1;
  Tantque  (J <= N)
  Faire
    Si (T[J]< 0) Alors
      /*permuter T[i] avec T[J] */
      Temp <---  T[J] ;
      T[J] <--- T[i];
      T[i] <--- Temp;
      i <--- i + 1;
    Fsi
    J <--- J + 1 ;
  Fait
  Ecrire("Vecteur avec valeurs negatives au debut ");
  Pour i <--- 1 a N
  Faire
    Ecrire(T[i]);
  Fait
Fin.
```

Solution 2:

```
/*
Algorithme qui met les valeurs negatives au debut et met les valeurs positives a la fin du
vecteur en maintenant l'ordre des valeurs positives et des valeurs negatives.
Idee: Idee: decaler les valeurs positives d'une position chaque fois vers la droite.

Exemple:
Vecteur initialement:
  -3   -5    7    2   -9   -5   -8    6

Vecteur avec valeur < 0 au debut:
  -3   -5   -9   -5   -8    7    2    6
idee: s'il y'a un nombre positif puis apres il y'a un nombre negatif (soit un nombre positif suivi
par un nombre negatif ou bien plusieurs nombres positifs suivi par un nombre negatif) il
faudra decaler les nombres positifs a droite et inserer le nombre negatif a la bonne place,
soit:
   - chercher indice i position du 1er nombre positif
   - a partir de i, chercher indice J position du 1er nombre negatif
   - Tant qu'il y'a un nombre negatif
        - decaler les nombres positifs a droite et inserer le nombre negatif a la bonne place
        - chercher s'il y'a encore un nombre negatif apres
*/

Algorithme  NegatifsAvantPositifsTab;
Var
   T: Tableau[1..100] de Entier;
   N, i, J, Temp: Entier;
Debut
   Repeter
      ecrire("Entrer N Nombre de Lignes de la matrice compris entre 1 et 20 : ")
```

```
      lire(N)
   Jusqua ((N >= 1) et (N <= 100));

   Pour i <- 1 a N Faire
     Ecrire("Donner ", i, " eme valeur de T");
     Lire(T[i]);
   Fait

   i <- 1;
   /* chercher i indice de la 1ere valeur positive */
   Tantque  ((i <= N) et (T[i] < 0))  Faire
       i <- i + 1;
   Faire
   J <- i + 1;

   /* chercher J indice la 1ere valeur negative */
   Tantque  ((J <= N) et (T[J] >= 0))  Faire
       J <- J + 1;
   Faire

   Tant Que (J <= N)  Faire
      Temp <- T[J];   // sauvegarder valeur negatif dans Temp
      // decaler les valeurs positifs d'une position vers la droite
      Tant Que (J > i) Faire
           T[J] <- T[J-1];
           J <- J - 1;
      Fait
      T[i] <- Temp;  // mettre valeur negatif dans position du 1er positif
      i <- i + 1;  // nouvelle position du 1er positif
      J <- i + 1;
      /* chercher J indice de la prochaine valeur negative */
      Tant Que  ((J <= N) et (T[J] >= 0))  Faire
        J <- J + 1;
      Faire
   Fait

   Ecrire("Vecteur avec valeurs negatives au debut ");
   Pour i <--- 1 a N
   Faire
        Ecrire(T[i]);
   Fait
Fin.
```

Solution 3:

```
/*
Idee: decaler les valeurs negatives d'une position chaque fois vers la gauche.
L'ordre des valeurs negatives est maintenue et l'ordre des valeurs positives est lui aussi
maintenue.
Par exemple:
T: -5, -8, 9, 12, 15, -11, -23
produit

T: -5, -8, -11, -23, 9, 12, 15

Solution
- Chercher la position i du 1er element positif
- Mettre J a (i + 1)
1) Tant Qu'il y'a au moins une valeur negative (J < N)
  - A partir de i + 1 chercher la position J du prochain element negatif (entre i et J il y'a 1 ou
plusieurs elements positifs).
```

```
  - S'il y'a une valeur negative
     - Decaler tous les elements negatifs consecutif chaque fois d'une position vers la gauche
(et inserer l'element positif dans la position correspondanate)
    - Mettre a jour i et J et revenir a 1)
*/


Algorithme     NegPuisPos ;
Var
   T :Tableau[1..100] de entier ;
   i,J,K, K1, N, Temp:entier ;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=100));

   Pour i <--- 1 a N
   Faire
       Ecrire("Donner ", i, " eme valeur du vecteur");
       Lire(T[i]);
   Fait


   /* Chercher la 1ere valeur >= 0 */
   i <--- 1;
   Tantque  ((i < N) et (T[i]<0))
   Faire
        i <--- i + 1 ;
   Fait
   /*déplacer les valeurs négatives au début */
   J <--- i + 1;
   Tantque  (J <= N)
   Faire
      /* Chercher la prochaine valeur < 0 */
      Tantque  ((J <= N) et (T[J] >= 0))
      Faire
           J <--- J + 1 ;
      Fait
      Si (J <= N) Alors
          K <--- J - 1;
         /* Faire un decalage a gauche d'une position a la fois des valeurs < 0,
            Jusqu'a ce que K devient egale a i pour avoir les valeurs >=0 deplacees a droite dans
            le meme ordre.
          */
         Tant Que (K >= i)
         Faire
            Temp <--- T[K];  /* Temp est une valeur >= 0 */
            K1 <--- K + 1;
            Tant Que ((K1 <= N) et (T[K1] < 0))
            Faire
               T[K1 - 1] <--- T[K1];
               K1 <--- K1 + 1;
            Fait
            T[K1 - 1] <--- Temp;
            K <--- K - 1;
         Fait
         i <--- K1 - 1; /* position de la derniere val >= 0 deplacee */
         J <--- i + 1; /* recommencer la recherche de la prochaine val < 0 a partir de la position i
+ 1 */
```

```
    Fsi
  Fait
  Ecrire("Vecteur avec valeurs negatives au debut ");
  Pour i <--- 1 a N
  Faire
    Ecrire(T[i]);
  Fait
Fin.
```

Exercice 2 :
Soient deux vecteurs d'entiers triés V1 (N entiers, N≤100) et V2 (M entiers, M≤150). Ecrire un algorithme qui construit un vecteur V3 composé des éléments communs aux deux vecteurs V1 et V2.

```
Algorithme      V3V1V2Commun;
Var
  V1, V3:Tableau[1..100] de Entier;
  V2:Tableau[1..150] de Entier;
  i, J, K, N, M:Entier;
Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 100");
    Lire(N);
  Jusqu'a ((N>=1) ET (N<=100));

  Repeter
    Ecrire("Donner M compris entre 1 et 150");
    Lire(M);
  Jusqu'a ((M>=1) ET (M<=150));

  Pour i <--- 1 a N
  Faire
   Ecrire("Donner ", i ," eme element du Vecteur V1 Trie");
   Lire(V1[i]);
  Fait

  Pour i <--- 1 a M
  Faire
   Ecrire("Donner ", i ," eme element du Vecteur V2 Trie");
   Lire(V2[i]);
  Fait

  K <--- 0;

  Pour i <-- 1 a N
  Faire
    Trouve <--- Faux;
    /* V2 est un tableau trie, Chercher V1[i] dans V2 uniquement dans le cas ou
       V2[1]<=V1[i]<=V2[M]
     */
    Si ((V1[i] >= V2[1]) ET (V1[i] <= V2[M])) alors
      J <--- 1;
      Tant Que ((J <= M) ET (Trouve = False) ET (V1[i] >= V2[J]))
      Faire
         Si (V1[i] = V2[J]) alors
           Trouve <--- Vrai;
         Sinon
           J <--- J + 1;
         Fsi
      Fait
```

```
    Fsi

    Si (Trouve = Vrai) alors
          K <--- K + 1;
          V3[K] <--- V1[i];
    Fsi
  Fait

  Si (K = 0) alors
      Ecrire("Vecteur des Elements communs a  V1 et V2 est Vide");
  Sinon
      Ecrire("Vecteur des Elements communs a V1 et V2:");
      Pour i <--- 1 a K
      Faire
          Ecrire(V3[i]);
      Fait
  Fsi

Fin.
```

Exercice 3 :

Soient deux vecteurs d'entiers triés V1 (N entiers, N≤100) et V2 (M entiers, M≤150). Ecrire un algorithme qui fusionne ces deux vecteurs dans un autre vecteur V3 trié sans répétition de valeurs identiques.

```
Algorithme    V1V2TriesFusionV3 ;
Var
   V1 :Tableau[1..100] de entier ;
   V2 :Tableau[1..150] de entier ;
   /* Taille maximale de V3 est 250 (cas N=100 et M=150 et tous les elements de V1 sont
different
      de tous les elements de V2.
    */
   V3 :Tableau[1..250] de entier ;
   i,J, K, N, M:entier ;
Debut
   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(N);
   Jusqu'a ((N>=1) et (N<=100));

   Repeter
      Ecrire("Donner N compris entre 1 et 100");
      Lire(M);
   Jusqu'a ((M>=1) et (M<=150));

   Pour i <--- 1 a N
   Faire
      Ecrire("Donner ", i, " eme valeur du vecteur V1 Trie par ordre croissant");
      Lire(V1[i]);
   Fait

   Pour i <--- 1 a M
   Faire
      Ecrire("Donner ", i, " eme valeur du vecteur V2 Trie par ordre croissant");
      Lire(V2[i]);
   Fait

   /* Parcourir les 2 vecteurs V1 et V2 */
   i <--- 1;
```

```
J <--- 1;
K <--- 0;
Tantque  ((i <= N) et (J <= M))
Faire
   K <--- K + 1;
   Si (V1[i] < V2[J]) alors
      /* S'il y'a des dupliques dans V1 les sauter */
      Tantque  ((i < N) et (V1[i] = V1[i + 1]))
      Faire
            i <--- i + 1;
      Fait
      V3[K] <--- V1[i];
      i <--- i + 1;
   Sinon
      Si (V2[J] < V1[i]) alors
        /* S'il y'a des dupliques dans V2 les sauter */
        Tantque  ((J < M) et (V2[J] = V2[J + 1]))
        Faire
            J <--- J + 1;
        Fait
        V3[K] <--- V2[J];
        J <--- J + 1;
      Sinon  /* cas V1[i] = V2[J] */
         /* S'il y'a des dupliques dans V1 les sauter */
        Tantque  ((i < N) et (V1[i] = V1[i + 1]))
        Faire
            i <--- i + 1;
        Fait
        /* S'il y'a des dupliques dans V2 les sauter */
        Tantque  ((J < M) et (V2[J] = V2[J + 1]))
        Faire
            J <--- J + 1;
        Fait
        V3[K] <--- V1[i];
        i <--- i + 1;
        J <--- J + 1;
      Fsi
   Fsi
Fait

/* Mettre dans V3 les elements qui restent soit dans V1 soit dans v2 */
Tantque  (i <= N)
Faire
   /* S'il y'a des dupliques dans V1 les sauter */
   Tantque  ((i < N) et (V1[i] = V1[i + 1]))
   Faire
       i <--- i + 1;
   Fait
   K <--- K + 1;
   V3[K] <--- V1[i];
   i <--- i + 1;
Fait
Tantque  (J <= M)
Faire
   /* S'il y'a des dupliques dans V2 les sauter */
   Tantque  ((J < M) et (V2[J] = V2[J + 1]))
   Faire
       J <--- J + 1;
   Fait
   K <--- K + 1;
```

```
    V3[K] <--- V2[J];
    J <--- J + 1;
  Fait

  /* Afficher V3 */
  Ecrire("Vecteur  V3:");
  Pour i <--- 1 a K
  Faire
      Ecrire(V3[i]);
  Fait
Fin.
```

Exercice 4 :

Soit une matrice A(N, M) de caractères (N≤20 et M≤30). Ecrire un algorithme qui
1- Recherche un élément dans la matrice A.
2- Détermine la transposé de la matrice A.

Soit une matrice A(N, M) de caractères (N≤20 et M≤30). Ecrire un algorithme qui
1- Recherche un élément dans la matrice A.

```
Algorithme   rechercheValMatrice;
Var
  A: tableau [1..20, 1..30] de Entier ;
  N, M, i, J, val  : Entier;
  Trouve : Booleen;

Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 20");
    Lire(N);
  Jusqu'a ((N>=1) ET (N<=20));

  Repeter
    Ecrire("Donner M compris entre 1 et 30");
    Lire(M);
  Jusqu'a ((M>=1) ET (M<=30]));

  Pour i <--- 1 à N
  Faire
     Pour J <--- 1 a M
     Faire
        Lire(A[i, J]) ;
     Fait
  Fait

  Ecrire("Donner un entier val a Rechercher");
  Lire (val);
  Trouve <--- Faux;
  i <--- 1;

  Tant Que ((i <= N) ET (Trouve = Faux))
  Faire
      J <--- 1;
      Tant Que ((J <= M) ET (Trouve = Faux))
      Faire
          Si (A[i, J] = val) alors
              Trouve <--- Vrai;
```

```
          Sinon
             J <--- J + 1;
          Fsi
       Fait
       i <--- i + 1;
   Fait

   Si (Trouve = Vrai) alors
      Ecrire(val, "Trouve dans la Matrice");
   Sinon
      Ecrire(val, "NON Trouve dans la Matrice");
   Fsi
Fin
```

2- Détermine la transposé de la matrice A.
La transposee d'une matrice A est la matrice TA ou les lignes de A sont les colonnes de TA (et par consequent les colonnes de A sont les lignes de TA). Donc A[i, J] est TA[J, i].

```
Algorithme   transposeeMatrice;
Var
  A: tableau [1..20, 1..30] de Entier;
  TA: Tableau[1..30, 1..20] de Entier;
  N, M, i, J  : Entier;

Debut
  Repeter
    Ecrire("Donner N compris entre 1 et 20");
    Lire(N);
  Jusqu'a ((N>=1) ET (N<=20));

  Repeter
    Ecrire("Donner M compris entre 1 et 30");
    Lire(M);
  Jusqu'a ((M>=1) ET (M<=30]));

  Pour i <--- 1 à N
  Faire
      Pour J <--- 1 a M
      Faire
          Lire(A[i, J]) ;
      Fait
  Fait

  Pour i <--- 1 à N
  Faire
      Pour J <--- 1 a M
      Faire
          TA[J, i] <--- A[i, J] ;
      Fait
  Fait

  /* Afficher la matrice transposee */
  Pour i <--- 1 à M
  Faire
      Pour J <--- 1 a N
      Faire
          Ecrire(TA[i, J]) ;
      Fait
  Fait
Fin
```

Exercice 5 :

Soit une matrice carrée A(N, N) d'entiers (N≤25). Ecrire un algorithme qui vérifie si la matrice A est triangulaire inférieure. (Une matrice est 'triangulaire inférieure' si elle ne comporte que des zéros au dessus de la diagonale).

Si A est la matrice:
Les elements dans la diagonale principale sont A[i, i]
Les elements sous la diagonale principale sont A[i, J] avec i > J
Les elements au-dessus de la diagonale principale sont A[i, J] avec i < J

```
Algorithme   minMaxMatrice;
Var
  A: tableau [1..20, 1..20] de Entier ;
  N, i, J  : Entier;
  trgInf : Booleen;

Debut
  Repeter
     Ecrire("Donner N compris entre 1 et 20");
     Lire(N);
  Jusqu'a ((N>=1) ET (N<=20]));

  Pour i <--- 1 à N
  Faire
     Pour J <--- 1 a N
     Faire
        Lire(A[i, J]) ;
     Fait
  Fait

   trgInf <--- Vrai;
   i <--- 1;
   Tant Que ((i <= N) et (trgInf = Vrai))
   Faire
       J <--- i + 1;
       Tant Que  ((J <= N) et (trgInf = Vrai))
       Faire
          Si (A[i, J] <> 0) alors
                trgInf <--- Faux;
          Sinon
                J <--- J + 1;
          Fsi
        Fait
        i <--- i + 1;
     Fait

     Si (trgInf = Vrai) alors
          Ecrire("La matrice est Triangulaire Inferieur");
    Sinon
         Ecrire("La matrice N'est Pas Triangulaire Inferieur");
    Fsi

Fin
```

Exercice 6 :

Ecrire un algorithme qui détermine si une phrase donnée contient toutes les voyelles.

```
Algorithme   ToutesVoyellesDansPhrase;
Var
```

```
    ph :chaine ;
    i, T : Entier ;
    existeVoys, existeA, existeO, existeI, existeU, existeY, existeE : Booleen ;
Debut
  Ecrire("Donner une Phrase") ;
  Lire(ph) ;
  T <--- Taille(ph);
  existeA <--- Faux;
  existeO <--- Faux;
  existeI <--- Faux;
  existeU <--- Faux;
  existeY <--- Faux;
  existeE <--- Faux;
  existeVoys <--- Faux;
  i <--- 1;

  // Une voyelle existe dans la phrase si elle existe en minuscule ou bien en majuscule (ou les
2).
  Tant Que ((i <= T) et (existeVoys = Faux))  Faire
       Cas ph[i]  Vaut
          'a', 'A': existeA <--- Vrai;
          'o', 'O': existeO <--- Vrai;
          'i', 'I': existeI <--- Vrai;
          'u', 'U': existeU <--- Vrai;
          'y', 'Y': existeY <--- Vrai;
         'e', 'E': existeE <--- Vrai;
      FCas
      existeVoys <--- existeA ET existeO ET existeI ET existeU ET existeY ET existeE;
      i  <--- i + 1;
  Fait
  Si (existeVoys = Vrai) alors
       Ecrire("Toutes les Voyelles existent dans la phrase");
  Sinon
       Ecrire("Certaines Voyelles n'existent pas dans la phrase");
  Fsi
Fin.
```

Exercice 7 :

Ecrire un algorithme qui permet de supprimer les espaces supplémentaires (plus d'un espace) dans une chaîne de caractère.

```
Algorithme      ElimineEspaces;
Var
    ch, chOut : Chaine ;
    i, J, T : Entier ;
Debut
  Ecrire("Donner la chaine de caracteres ch") ;
  Lire(ch) ;
  T <--- Taille(ch);
  i <-- 1;
  J <--- 1;
  Tant Que (i <= T)  Faire
      Si (ch[i] <> ' ')  alors   // recopier les caracteres qui ne sont pas des espaces
            chOut[J] <--- ch[i];
            i <--- i + 1;
     Sinon
            chOut[J] <--- ch[i];    // recopier le 1er espace
            i <--- i + 1;
           // Sauter les blancs de ch
           Tant Que ((i <= T) et (ch[i] = ' '))  Faire
```

```
                i <--- i + 1;
            Faire
        Fsi
       J <--- J + 1;
   Fait

   ChangerTaille(chOut, J - 1);
   ch <--- chOut;
   Ecrire("La chaine ch maintenant est: ", ch);
Fin.
```

Exercice 8 :
Ecrire un algorithme qui détermine si un mot est un palindrome. Sachant qu'un palindrome se lit de gauche à droite et de droite à gauche (ex : RADAR, ELLE, ICI).

```
Algorithme    ChainePalindrome;
Var
   ch : Chaine;
   i,J : Entier ;
   pal : Booleen ;
Debut
    i <--- 1 ;
   J <--- Taille(ch) ;
   pal <--- Vrai ;
   Tantque ((i < J) et (pal))  Faire
       Si (ch[i] <> ch[J]) alors
          pal <--- Faux
       Sinon
         i <--- i + 1 ;
        J <--- J - 1 ;
      Fsi
  Fait

   Si (pal) alors
         Ecrire("La chaine: ", ch, " est un Palindrome");
   Sinon
        Ecrire("La chaine: ", ch, " N' est PAS un Palindrome");
   Fsi
Fin.
```

Exercice 9 :
Ecrire un algorithme permettant de convertir une chaine de caractères (composée des caractères 0..9) en un entier décimal.

```
Algorithme      ChaineCarADecimale;
Var
    ch : Chaine ;
    i, T, nb : Entier ;
Debut
   Ecrire("Donner une chaine de caracteres composee des caracteres 0..9") ;
   Lire(ch) ;
   nb <--- 0;
   T <--- Taille(ch);
    // On suppose ici que la chaine contient uniquement les caracteres '0' .. '9'
    // On suppose aussi que les caracteres sont codes en Ascii
    Pour i <--- 1 a T  Faire
        nb <--- (nb * 10) +  (T[i] - '0');
    Fait
    Ecrire("L'entier Decimale Correspondant a la chaine: ", ch, "est:", nb);
Fin.
```

Exercice 10.1 :

Ecrire un algorithme qui

1. Verifie l'existence d'une sous chaine dans une chaine.

```
Algorithme   existeSchaineDansChaine;
Var
    ch, sch :chaine ;
    i, J, saveI, Tch, Tsch : Entier ;
    existeSch : Booleen ;
Debut
   Ecrire("Donner la chaine de caracteres") ;
   Lire(ch) ;
   Ecrire("Donner la sous chaine de caracteres") ;
   Lire(sch) ;
   Tch <--- Taille(ch) ;
   Tsch <--- Taille(sch) ;

   existeSch <--- Faux;  /* on suppose au debut que la s/chaine n'existe pas dans la chaine */
   i <--- 1;
   Tant Que ((i <= (Tch - Tsch + 1)) ET (NON(existeSch)))
   Faire
         /* Chercher le 1er caractere de la s/chaine dans la chaine */
        Tant Que ((i <= (Tch - Tsch + 1)) ET (ch[i] <> sch[1]))
        Faire
              i <--- i + 1;
        Fait
        Si (i <= (Tch - Tsch + 1))  alors /* on a trouve le 1er caractere, comparer les autres
caracteres */
             saveI <--- i;   // sauvegarder i car on aura besoin de cette valeur si ce n'est pas une
s/chaine
             J <--- 1;
             Tant Que ((J <= Tsch) ET (ch[i] = sch[J]))
             Faire
                  i <--- i + 1;
                  J <--- J + 1;
             Fait
             Si (J > Tsch)  alors   /* on a trouve la sous chaine dans la chaine */
                 existeSch <--- Vrai;
             Sinon    /* reprendre la recherche a partir de saveI + 1 */
                 i <--- saveI + 1;
             Fsi
        Fsi
    Fait
    Si (existeSch)  alors
         Ecrire("La Sous chaine ", sch, ' Existe dans la chaine ", ch);
    Sinon
         Ecrire("La Sous chaine ", sch, ' N' Existe PAS dans la chaine ", ch);
    Fsi
Fin.
```

Exercice 10.2 :

Ecrire un algorithme qui

2- Supprime (élimine) la suite de N caractères de la chaîne CH à partir de la position P.

```
Algorithme      ElimineCarsPosP;
Var
   ch, chOut : Chaine ;
   i, J, P, N, T : Entier ;
Debut
```

```
  Ecrire("Donner la chaine de caracteres ch") ;
  Lire(ch) ;
  Ecrire("Donner le Nombre de Caracteres N a Eliminer"") ;
  Lire(N) ;
  Ecrire("Donner la Position P a partir de laquelle vous voulez eliminer les N caracteres"") ;
  Lire(P) ;
  T <--- Taille(ch);
  Si (P > T) alors
       Ecrire("Erreur Position P est superieur a la Taille de la chaine");
  Sinon
       chOut <--- ""; // initialiser chOut a la chaine Vide
       Pour i <--- 1 a (P - 1)  Faire   // recopier la partie avant P dans chOut
            chOut[i] <--- ch[i];
       Fait
       // Ajouter les Caracteres apres les N Caracteres a supprimer
       // Remarquons si (P + N > T), c.a.d on veut supprimer plus de caracteres qu'il y'a apres
P
       // on vas supprimer tous les caracteres car dans ce cas (P + N > T) et on ne rentre pas
       //  dans la boucle Pour
       J <-- i;
       Pour i <--- (P + N) a T  Faire
            chOut[J] <--- ch[i];
            J <--- J + 1;
       Fait
       ChangerTaille(chOut, J - 1);
       ch <--- chOut;

      Ecrire("La chaine CH est maintenant: ", ch);
   Fsi
Fin.
```