SOLUTION TD LES FICHIERS

Exercice 1 :

Soit le fichier NOMBRES.BIN qui contient une liste de nombres entiers.
Écrire un algorithme qui affiche les nombres du fichier, leur somme et leur moyenne.

NOMBRES.BIN :
12 – 13 - 7 – 8 – 11 - FDF
Somme: 12 + 13 + 7 + 8 + 11 = 51
Moyenne : 51 / 5 = 10.2

Solution

```
Algorithme   SomMoyFichEntiers;
Var
   F : Fichier de Entier;
   nbr, Som, cpt : Entier;
Debut
   Assigner(F, "NOMBRES.BIN");
   Relire(F);
   cpt ← 0; /* compte combien d'entiers il y'a dans le fichier */
   Som ← 0;
   Tant Que (NON(FDF(F)))
   Faire
        Lire(F, nbr);
        Ecrire(nbr);
        Som ← Som + nbr;
        cpt ← cpt + 1;
   Fait
   Fermer(F);
   Si (cpt = 0) alors
        Ecrire("Fichier NOMBRES.BIN est Vide");
   Sinon
        Ecrire("Somme des elements du fichier = ", Som, " et Moyenne
= ", Som/cpt);
   Fsi
Fin.
```

Exercice 2 :

Soit Fintervalle un fichier contenant des intervalles d'entier où chaque intervalle I est défini par deux entiers a et b inclus dans I (I=[a,b]).
1- Donner la déclaration d'un intervalle I.

2- Ecrire une action paramétrée Imax qui détermine le plus long intervalle du fichier Fintervalle.
3- Soit Fent un fichier d'entiers. Ecrire une procédure CreerF permettant de créer le fichier Fent contenant les éléments du plus long intervalle du fichier Fintervalle dans un ordre décroissant.
4- Ecrire une procédure Supprime qui supprime les bornes a et b de l'intervalle I du fichier Fent.

Solution

1-
```
Type
    Intervalle = Enregistrement
          a, b: Entier;
    Fin;
    FichIntervalle = Fichier de Intervalle;
```

2- [2, 5] 2, 3, 4, 5 [1, 4] 1, 2, 3, 4 [2, 2]
Notons que 0<a<=b.

Fichier Fintervalle:
[2 , 5] – [1 , 7] – [1 , 4] – [3 , 10] – [2 , 9] – FDF

```
Fonction  Imax(Fintervalle: FichIntervalle): Intervalle
Var
   long, d: Entier;
   iv, retIv: Intervalle;

Debut
   /*  On supposera que l'assignation du fichier Fintervalle se fait dans l'algorithme
        principale. Aussi si le fichier est vide on retournera retIv avec retIv.b = - 1.
    */
   retIv.b ← -1;
   Relire(Fintervalle);
   Si (Non(FDF(Fintervalle)))  alors
        // La plus petite valeur de (iv.b - iv.a) est 0, donc on initalise long a une valeur plus petite comme -1
         long ← -1;
        Tant Que (NON(FDF(Fintervalle)))  Faire
            Lire(Fintervalle, iv);
            d ←  iv.b – iv.a;
            Si (d > long) alors
```

```
                long ← d;
                retIv ← iv;
            Fsi
        Fait
    Fsi
    Fermer(Fintervalle);

    Retourner  retIv;
Fin;
```

3-
Fichier Fintervalle:
[2 , 5] – [1 , 7] – [1 , 4] – [3 , 10] – [2 , 8] – FDF

Le plus long intervalle est: [3 , 10] 3, 4, 5, 6, 7, 8, 9, 10
Fichier Fent:
  10 – 9 – 8 – 7 – 6 – 5 – 4 – 3 – FDF

```
Procedure   CreerF(S/ Fent:Fichier de Entier, E/ Fintervalle:
FichIntervalle)
Var
    ivMax: Intervalle;
    v: Entier;
Debut
    /*  On supposera que l'assignation des fichiers se fait dans
l'algorithme principale.
     */
     Reecrire(Fent);
     ivMax ← Imax(Fintervalle);
     Si (ivMax.b <> -1)  alors // Si fichier Fintervalle n'est pas vide
         // Dans le cas ou ivMax.a = ivMax.b on veut ecrire la borne 2
fois
         Si (ivMax.a = ivMax.b) alors
              Ecrire(Fent, ivMax.a);
              Ecrire(Fent, ivMax.a);
         Sinon
              // ecrire tous les entiers compris entre ivMax.b jusqu'a
ivMax.a
              v ← ivMax.b;
             Tant Que (v >= iv.a)  Faire
                    Ecrire(Fent, v);
                    v ← v – 1;
             Fait
```

```
        Fsi
      Fsi
      Fermer(Fent);
Fin;
```

4 -
Fintervalle : [3, 3] – [4, 5] , - [6, 6]
Fent : 4 , 5
Fichier Fent en Entree:
10 – 9 – 8 – 7 – 6 – 5 – 4 – 3 – FDF
Fichier FTmp en sortie:
9 – 8 – 7 – 6 – 5 – 4 – FDF

```
Procedure   Supprimer(E/S Fent:Fichier de Entier)
Var
   FTmp: Fichier de Entier;
   v: Entier;
Debut
   /*  On supposera que l'assignation du fichier Fent se fait dans
l'algorithme principale.
    */
    Relire(Fent);

    Si (NoN(FDF(Fent)))  alors
            Assigner(FTmp, "FichierTmp");
            Reecrire(FTmp);
           // Le fichier Fent contient au moins 2 valeurs s'il n'est pas
vide
           Lire(Fent, v);  // Lire la premiere valeur de Fent
           Lire(Fent, v);  // Lire la seconde valeur de Fent
           Tant Que (NON(FDF(Fent)))  Faire
               Ecrire(FTmp, v);
               Lire(Fent, v);    // la derniere valeur de Fent ne sera pas
ecrite dans FTmp
            Fait
          Fermer(FTmp);
          Fermer(Fent);

         /*  Recopier FTmp dans Fent */
         Relire(FTmp);
         Reecrire(Fent);
         Tant Que (NON(FDF(FTmp)))  Faire
             Lire(FTmp, v);
             Ecrire(Fent, v);
```

```
            Fait
            Fermer(FTmp);
        Fsi
        Fermer(Fent);

Fin;
```

Exercice 3 (Rattrapage 2019):

Soit FCAR un fichier de caractères contenant une suite de mots séparés par un ou plusieurs blancs (espaces).

1- Ecrire une action paramétrée permettant de créer un fichier de mots FMOT contenant tous les mots du fichier FCAR sans les espaces.

2- Soit C un caractère donné. Ecrire une action paramétrée permettant de créer un fichier de mots FPR contenant tous les mots de FMOT commençant avec le caractère C .

3- Ecrire une action paramétrée permettant de créer, à partir du fichier FMOT, un fichier FSTAT contenant, pour chaque mot de FMOT, le mot lui-même et son nombre de répétition.

Solution

1-

```
Procedure CreerFMot(E/ Fcar : Fichier de Caracteres, E/S FMot :
Fichier de Chaine)
Var
    car: Caractere;
    mot: Chaine;
Debut
     /*  L'assignation des fichiers se fait dans l'algorithme */
     Relire(FCar);
     Reecrire(FMot) ;
    Tant Que (Non(FDF(FCar)))   Faire
          Lire(F, car);
         /* sauter les blancs */
         Tant Que (Non(FDF(FCar)) ET (car = ' '))
         Faire
                Lire(FCar, car);
         Fait
        Si (car <> ' ')  alors  /* Debut d'un mot */
            i ← 0;
            mot ← "";
           Tant Que (Non(FDF(FCar)) ET (car <> ' '))
           Faire
```

```
                i ← i + 1;
                mot[i] ← car;
                Lire(FCar, car);
            Fait
           /* Cas ou le fichier se termine par un mot ou on a le dernier
caractere est suivi de
               FDF,  exemple:  abc abcdefFDF
            */
            Si (car <> ' ') alors
                 i ← i + 1;
                 mot[i] ← car;
            Fsi
            Ecrire(FMot, mot)
     Fsi
   Fait
   Fermer(FCar);
   Fermer(FMot);
Fin;
```

2-

```
Procedure CreerFPR(E/ FMot: Fichier de Chaine, E/ c: Caracctere,
                                E/S FPR: Fichier de Chaine)
Var
    mot: Chaine;
Debut
     /*  L'assignation des fichiers se fait dans l'algorithme */
     Relire(FMot);
     Reecrire(FPR) ;
    Tant Que (Non(FDF(FMot)))   Faire
          Lire(FMot, mot);
          Si (mot[1] = c) alors
               Ecrire(FPR, mot);
          Fsi
     Fait
     Fermer(FPR);
     Fermer(FMot);
Fin;
```

3-

```
TYPE
   Stat = Enregistrement
        mot: Chaine;
        nb: Entier;
   Fin;
```

```
  FichStat = Fichier de Stat;

Procedure CreerFStat(E/ FMot: Fichier de Chaine, E/S FStat:
FichStat)
Var
   FMot1: Fichier de Chaine;
   mot, mot1: Chaine;
   s: Stat;
   motExiste: Booleen;
Debut
    /*  L'assignation des fichiers FMot et Fstat  se fait dans
l'algorithme */

    /*  Recopier Fmot dans FMot1*/
   Assigner(FMot1, "FMot1");
   Relire(FMot);
   Reecrire(FMot1) ;
   Tant Que (Non(FDF(FMot)))   Faire
        Lire(F, mot);
        Ecrire(FMot1, mot);
    Fait
    Fermer(FMot1);
    Fermer(FMot);

    pos ← 0;
    Reecrire(FStat) ;
    Relire(FMot);
    Tant Que (Non(FDF(FMot)))
    Faire
      Lire(FMot, mot);
      pos ← pos + 1;

      motExiste ← Faux;
      Relire(FMot1);
      i ← 1;
     /* Ouvrir FMot1 en lecture et chercher si mot existe avant la
position pos donc on va lire
       FMot1 jusqu'a (pos - 1) au maximum et on compare avec mot si
on trouve l'element
       avant la position pos on sort
     */
     Tant Que (NoN(FDF(FMot1)) et (i < pos) et (motExiste = Faux))
     Faire
       Lire(FMot1, mot1);
```

```
        Si (mot1 = mot) alors
          /* on a trouve mot avant la position pos donc on met le
booleen a
             Vrai pour sortir du Tant Que.
           */
           motExiste ← Vrai;
        Sinon
          i ← i + 1;
        Fsi
      Fait
      Si (motExiste = Faux) /* l'element n'existe pas avant pos
                                donc calculer nbr d'occurrences */
           s.mot ← mot;
           s.nb← 0;
          Tant Que (NoN(FDF(FMot1))
          Faire
             Lire(FMot1, mot1);
             Si (mot1 = mot) alors
                  s.nb ← s.nb + 1;
             Fsi
           Fait
           Ecrire(FStat, s);
       Fsi
       Fermer(FMot1);
     Fait
     Fermer(FMot);
     Fermer(FStat);
Fin;
```

Exercice 4 :

Considérons les types d'enregistrements suivants :

```
Type
      TPoint = Enregistrement
             x,y : entier ;
      Fin;
      Tspace = Enregistrement
             Position : Tpoint ;
             Superficie, Altitude :entier ;
      Fin ;
      TVille = Enregistrement
             InfoG : TSpace;
             Nom :chaine[25] ;
             NBH :entier ; //nombre d'habitants
```

Fin;

1- Ecrire une fonction Densite qui renvoie la densité de population d'une Ville donnée.

2- Soit FVille un fichier de villes d'un pays. Ecrire un algorithme permettant de :

a- Remplir le fichier FVille.

b- Afficher la capitale du pays sachant que c'est la ville la plus dense.

c- Créer un fichier G contenant les villes se trouvant sur le même axe (Ox) que la capitale.

d- Comment peut-on trouver les villes voisines de la capitale suivant l'axe (Ox) ?

Solution

1 – Densite de population: Nombre moyen d'habitants par kilometre carre.

```
Fonction  Densite(v: Tville): Reel
Debut
    Retourner  v.NBH/v.InfoG.Superficie;
Fin;

Algorithme FichiersFetG
Type
      TPoint = Enregistrement
            x,y : entier ;
      Fin;
      Tspace = Enregistrement
            Position : Tpoint ;
            Superficie, Altitude :entier ;
      Fin ;
      TVille = Enregistrement
            InfoG : TSpace;
            Nom :chaine[25] ;
            NBH :entier ; //nombre d'habitants
       Fin;
Var
  Fville, G: Fichier de TVille;
  cont, minD, minG, d: Entier;
  v, vCap, vD, vG: Tville;

Fonction  Densite(v: Tville): Reel
Debut
    Retourner  v.NBH/v.InfoG.Superficie;
```

```
Fin;

Debut
   Assigner(FVille, "FichierVille");
   Reecrire(FVille);

   /* a. Remplir le Fichier FVille. On s'arrete lorsque l'utilisateur rentre
l'entier 0. */
   Repeter
        Avec v, v.InfoG, v.InfoG.Position Faire
             Ecrire ("Donner Position(x,y), Superficie, Altitude,
                        Nom et Nbr Habitants de ", i, " ville");
             Lire(x, y, Superficie, Altitude, Nom, NBH);
        Fait
        Ecrire(FVille, v);
        Ecrire("Rentrer 0 pour arreter le remplissage de FVille, autre
entier pour continuer le remplissage");
        Lire(cont)
    Jusqu'a (cont = 0);
    Fermer(FVille);

    /*   b. Afficher la capitale du pays sachant que c'est la ville la plus
dense. */
    Relire(FVille);
    Lire(FVille, vCap);
    Tant Que (NON(FDF(FVille)))  Faire
           Lire(FVille, v);
           Si  (Densite(v) > Densite(vCap))  Alors
                vCap ← v;
           Fsi
    Fait
    Fermer(FVille);
    Ecrire ("Capitale Nom, Nbr Habitants, Superficie, Position(x,y),
Altitude ");
    Avec vCap, vCap.InfoG, vCap.InfoG.Position Faire
          Ecire(Nom, NBH, Superficie, x, y, Altitude);
    Fait

    /*   c. Créer un fichier G contenant les villes se trouvant sur le
même axe (Ox) que la capitale.
      */

    Assigner(G, "G");
    Relire(FVille);
```

```
   Reecrire(G);
   Tant Que (NON(FDF(FVille)))  Faire
         Lire(FVille, v);
         Si ( (v.Nom <> vCap.Nom) et (vCap.InfoG.Position.y
=  v.InfoG.Position.y) ) Alors
              Ecrire(G, v);
         Fsi
   Fait
   Fermer(FVille);
   Fermer(G);

   /*   d. Comment peut-on trouver les villes voisines de la capitale
suivant l'axe (Ox) ? */
    Relire(G);
    Si (FDF(G))  Alors
         Ecrire ("  Il n'y a pas de Ville se trouvant sur le meme axe 0x
que la capitale ");
    Sinon
         /*   Il faut voir les villes dont x > x de la capitale et chercher le
min et
           aussi les ville dont x < x de la capitale et chercher le min
          */
          /*   Circonference de la terre a l'equateur est environ 40000
Km donc la distance entre 2 villes au
               maximum est egale a environ 40000 Km. Donc on vas
mettre une valeur plus grande,
               par exemple 50000 Km et dans ce cas la distance entre
2 villes ne peut aps etre > 50000 Km.
          */
          minG ← 50000;   /*  Min voisin de gauche */
          minD ← 50000;   /*   Min voisin de droite */
          Tant Que (Non(FDF(G)))  Faire
              Lire(G, v);
              Si (v.InfoG.Position.x > vCap.InfoG.Position.x) alors  //
voisin de droite
                   d ←  v.InfoG.Position.x – vCap.InfoG.Position.x;
                   Si (d < minD) alors
                        minD ← d;
                        vD ← v;
                   Fsi
              Sinon
                   d ←  vCap.InfoG.Position.x – v.InfoG.Position.x;
                   Si (d < minG) alors
                        minG ← d;
```

```
                        vG ← v;
                    Fsi
                Fsi
            Fait
            Si (minD <> 50000) Alors
                Ecrire ("Voisin de Droite:  Nom, Nbr Habitants, Superficie,
Position(x,y), Altitude");
                Avec vD, vD.InfoG, vD.InfoG.Position Faire
                        Ecrire(Nom, NBH, Superficie, x, y, Altitude);
                Fait
            Fsi
            Si (minG <> 50000) Alors
                Ecrire ("Voisin de Gauche:  Nom, Nbr Habitants,
Superficie, Position(x,y), Altitude");
                Avec vG, vG.InfoG, vG.InfoG.Position Faire
                        Ecrire(Nom, NBH, Superficie, x, y, Altitude);
                Fait
            Fsi
    Fsi
 Fin.
```

Exercice 5 :

1) Soient F1 et F2 deux fichiers d'entiers strictement positifs et sans répétition. Ecrire un algorithme qui construit (crée) un fichier G d'entiers tel que G contient pour chaque valeur de F1 la valeur et tous ses multiples appartenant à F2 (F1 et F2 sont supposés existants).

Exemple :

F1 : 3 10 20 17

F2 : 3 6 19 60 40 30

G : **3** 3 6 60 30 **10** 60 40 30 **20** 60 40 **17**

2) Ecrire un algorithme qui permet à partir du fichier résultat (G) de générer un autre fichier (H) contenant toutes les valeurs du fichier (G) (sans répétition) avec leur nombre.

Exemple :

H

: **3** 2 **6** 1 **60** 3 **30** 2 **10** 1 **40** 2 **20** 1 **17** 1

Solution

Question 1)

```
Algorithme   CreerFichierG;
Var
   F1, F2, G: Fichier d'entier;
   v1, v2:  Entier;
Debut
   Assigner(F1, "F1");
   Assigner(F2, "F2");
   Assigner(G, “G”);

   Relire(F1);
   Reecrire(G);
  Tant Que (Non(FDF(F1))
  Faire
      Lire(F1, v1);
      Ecrire(G, v1);
      Relire(F2);
     Tant Que (Non(FDF(F2))
     Faire
         Lire(F2, v2);
         Si ((v2 mod v1) = 0)  alors
            Ecrire(G, v2);
         Fsi
     Fait
     Fermer(F2);
   Fait
   Fermer(F1);
   Fermer(G);
Fin.
```

Question 2)

Solution 1

G : 3 3 6 60 30 10 60 40 30 20 60 40 17
G1 : 3 3 6 60 30 10 60 40 30 20 60 40 17

G1: 60 30 10 60 40 30 20 60 40 17

G2 : 30 10 40 30 20 40 17

nbrOcc: 3
H : 3 2 6 1 60 3

L'idee utilisee est la suivante: Recopier d'abord le fichier G dans un fichier G1 car on ne veut pas modifier G. On ouvre G1 en lecture et on vas a a).
a) On vas compter le nombre d'occurrences du 1er element de G1 et on vas recopier dans un fichier G2 tous les elements de G1 qui sont differents du 1er element de G1. Donc on initialise nbrOcc a 1 apres lecture du 1er element de G1 ensuite lire les autres elements de G1 en sequence. Si l'element qu'on lit est egale au 1er element alors incrementer le nombre d'occurrences sinon ecrire l'element dans le fichier G2 (on elimine ainsi les dupliques du 1er element dans G2).. Apres lecture de tous les elements de G1 on ecrit dans H le 1er element de G1 et son nombre d'occurrences. Ensuite on recopie G2 dans G1 et on repete la meme chose avec le prochain 1er element de G1 en allant a a).

```
Algorithme  CreerFichierH;
Var
  G, G1, G2, H: Fichier de Entier;
  nbrOcc, v1, v2: Entier;
Debut
  Assigner(G, "G");
  Assigner(G1, "G1");
  Assigner(G1, "G2");
  Assigner(H, "H");
  Reecrire(H) ;

   /*  Recopier G dans G1 */
  Relire(G);
  Reecrire(G1) ;
  Tant Que (NON(FDF(G)))  Faire
        Lire(G, v1) ;
        Ecrire(G1, v1) ;
  Fait
  Fermer(G) ;
  Fermer(G1) ;

  /*  Compter nbr d'occurrences du 1er element de G1 et recopier
dans G2 tous les
      elements de G1 qui sont differents du 1er element de G1 */
  Relire(G1);
  Tant Que (NON(FDF(G1)))  Faire
        Lire(G1, v1);
        nbrOcc  ← 1;
```

```
        Reecrire(G2);
        Tant Que (NON(FDF(G1)))  Faire
             Lire(G1, v2);
             Si (v1 = v2) alors
                nbrOcc ← nbrOcc + 1;
             Sinon
                  Ecrire(G2, v2);
             Fsi
        Fait
        Fermer(G1);
        Fermer(G2);
        Ecrire(H, v1);
        Ecrire(H, nbrOcc);
       /*  Recopier G2 dans G1 et dans ce cas on aura enleve les
dupliques de v1 */
       Relire(G2);
       Reecrire(G1);
      Tant Que (NON(FDF(G2)))  Faire
            Lire(G2, v1);
            Ecrire(G1, v1);
       Fait
       Fermer(G2);
       Fermer(G1);
       Relire(G1);  // reevrourir G1 en lecture avant de revenir au
FDF(G1)
  Faire
  Fermer(H);
  Fermer(G1) ;
Fin.
```

Solution 2

G : 3 3 6 60 30 10 60 40 30 20 60 40 17
G1 : 3 3 6 60 30 10 60 40 30 20 60 40 17

L'idee utilisee est la suivante: Lire le fichier G element par element en sequence. Chaque element qu'on lit on connait sa position: le 1er a la position 1, le 2eme la position 2 , ... ainsi de suite.
Chaque fois qu'on lit un element si cet element existe auparavant, c.a.d qu'il existe avant sa
position courante cela veut dire qu'il a ete traite et qu'il n'y a rien a faire, si cet element n'existe pas avant sa position courante cela veut dire c'est la 1ere fois qu'on rencontre cet element et qu'il faut compter son nombre d'occurrences nbrOcc et on va ecrire cet element et

nbrOcc dans H. On utilisera un fichier de nom logique G1 qui est associe au meme fichier physique que celui de G (une copie de G) pour faire la recherche de l'element courant de G avant sa position courante et pour compter le nombre d'occurrences de cet element dans le cas ou il n'existe pas avant sa position courante.

```
Algorithme  CreerFichierH;
Var
  G, G1, H: Fichier de Entier;
  nb, nb1, pos, i, nbOcc: Entier;
  nbExiste:Booleen;
Debut
  /* G et G1 sont assignes au meme fichier physique "G" */
  Assigner(G, "G");
  Assigner(G1, "G");
  Assigner(H, "H");

  Relire(G);
  Reecrire(H);
  pos ← 0;
  Tant Que (Non(FDF(G)))
  Faire
     Lire(G, nb);
     pos ← pos + 1;

     nbExiste ← Faux;
     Relire(G1);
     i ← 1;
     /* Ouvrir G1 en lecture et chercher si nb existe avant la position
pos donc on va lire
        G1 jusqu'a (pos - 1) au maximum et on compare avec nb si on
trouve l'element
        avant la position pos on sort
      */
     Tant Que (NoN(FDF(G1)) et (i < pos) et (nbExiste = Faux))
     Faire
        Lire(G1, nb1);
        Si (nb1 = nb) alors
           /* on a trouve nb avant la position pos donc on met le booleen
a
              Vrai pour sortir du Tant Que.
            */
           nbExiste ← Vrai;
        Sinon
```

```
            i ← i + 1;
          Fsi
        Fait
        Si (nbExiste = Faux) /* l'element n'existe pas avant pos
                                donc calculer son nbr d'occurrences */
            nbOcc ← 0;
           Tant Que (NoN(FDF(G1))
           Faire
              Lire(G1, nb1);
              Si (nb1 = nb) alors
                   nbOcc ← nbOcc + 1;
              Fsi
            Fait
            Ecrire(H, nb);
            Ecrire(H, nbOcc);
         Fsi
         Fermer(G1);
       Fait
       Fermer(G);
       Fermer(H);
Fin.
```

Exercice 6 (Rattrapage 2014):

Soit F un fichier d'entiers représentant des séquences de nombres séparées par un ou plusieurs zéro. Ecrire un algorithme qui réalise les traitements suivants :

1. A partir de F (fichier existant), crée un fichier G contenant pour chaque séquence, la moyenne des nombres qui la constituent.
2. Puis, Supprimer les valeurs nulles du fichier G.

Exemple :

F : 0 0 **1 4 3 7** 0 0 0 **6 -9 2 7 -6** 0 0 0 **-10 3** 0 0

G : 3,75 0,00 -3,50

G : 3,75 -3,50

Solution

```
Algorithme FichiersFetG
Var
  F: Fichier d'entier;
  G, Tmp: Fichier de Reel;
   i, v1: Entier;
```

```
  v2, Moy: Reel;
Debut
  Assigner(F, "F");
  Assigner(G, "G");

  /*  1) A partir de F (fichier existant), crée un fichier G contenant pour
chaque séquence,
        la moyenne des nombres qui la constituent.
    */
  Relire(F);
  Reecrire(G) ;
  Tant Que (Non(FDF(F)))
  Faire
      Lire(F, v1);
      /* sauter les 0 */
      Tant Que (Non(FDF(F)) ET (v1=0))
      Faire
         Lire(F, v1);
      Fait
      Si (v1 <> 0)  alors  /* Debut d'une sequence */
        Moy ← 0;
        i ← 0;
        Tant Que (Non(FDF(F)) ET (v1<>0))
        Faire
            Moy ← Moy + v1;
            i ← i + 1;
            Lire(F, v1);
        Fait
        /* Cas ou le fichier se termine par une sequence ou on le
dernier nombre est suivi de
            FDF,  exemple:  0  0  2  5 7 FDF
         */
        Si (v1 <> 0) alors
             Moy ← Moy + v1;
             i ← i + 1;
        Fsi
        Moy ← Moy / i;
        Ecrire(G, Moy)
     Fsi
   Fait
   Fermer(F);
   Fermer(G);

   /*  2) Supprimer les valeurs nulles du fichier G. */
```

```
   Relire(G);
   Reecrire(Tmp);
   Tant Que (Non(FDF(G)))
   Faire
     Lire(G, v2);
     Si (v2 <> 0)  alors
        Ecrire(Tmp, v2);
     Fsi
  Fait
  Fermer(G);
  Fermer(Tmp);

  /*  Recopier le fichier Tmp dans le fichier G. */
  Relire(Tmp);
  Reecrire(G);
  Tant Que (Non(FDF(Tmp)))
  Faire
      Lire(Tmp, v2);
      Ecrire(G, v2);
  Fait
  Fermer(G);
  Fermer(Tmp);
Fin.
```

Exercice 7 :

Soient F1 et F2 deux fichiers de chaînes de caractères. Chaque chaîne représente un mot.
Ecrire un algorithme qui construit un fichier F3, tel que F3 contient les mots de F1 qui n'existent pas dans F2.

F1 :
"Benali" - "Doukal" - "Bessa" – "Mostefa" – "Ali"

F2 :
"Douma" - "Benali" - "Bessa" - "Ali"

F3 : "Doukal" - "Mostefa"

```
Algorithme    FichiersF1F2
Var
  F1, F2, F3: Fichier de Chaine
  ch1, ch2: Chaine
```

```
  Trouve: Booleen
Debut
   Assigner(F1, "F1")
   Assigner(F2, "F2")
   Assigner(F3, "F3")
   Relire(F1)
   Reecrire(F3)
  Tant Que (Non(FDF(F1))
  Faire
      Lire(F1, ch1)
      Relire(F2)
      Trouve ← Faux
      Tant Que (Non(FDF(F2)) et (Non(Trouve))
      Faire
         Lire(F2, ch2)
         Si (ch1 = ch2) alors
            Trouve ← Vrai
         Fsi
      Fait
      Fermer(F2)
      Si (Non(Trouve)) alors
          Ecrire(F3, ch1)
      Fsi
  Fait
   Fermer(F1)
   Fermer(F3)
Fin
```

Serie Complementaire

Exercice 8 :

1) Ecrire une procédure qui supprime le dernier élément du fichier F de nombre entiers.
2) En utilisant la procédure précédente, écrire un algorithme pour vider un fichier F existant de nombre entiers et de nom physique 'ESSAI.DAT', élément par élément. Cet algorithme devra afficher, après la suppression de chaque élément, la moyenne des éléments restants de F.

Solution

```
TYPE
      FEnt = Fichier de Entier;
```

```
Procedure SupprimerDernierEltFich(E/S F: FEnt)
Var
     FTmp: Fichier d'Entier;
     v : Entier;
Debut
   // assignation de F faite dans l'algorithme
   Relire(F);
   Si (Non(FDF(F))) alors
      Assigner(FTmp, "FichierTemp");
      Reecrire(FTmp);
      Lire(F, v);
      Tant Que ( (Non(FDF(F))
      Faire
           Ecrire(FTmp, v);
           Lire(F, v);
      Fait
      Fermer(F);
      Fermer(FTmp);

      Relire(FTmp);
      Reecrire(F);
      Tant Que ((Non(FDF(F))
      Faire
          Ecrire(FTmp, v);
          Lire(F, v);
      Fait
      Fermer(FTmp);
   Fsi
   Fermer(F);
Fin;

Algorithme  FichierEssai;
Type
  FEnt = Fichier de Entier;
Var
   F: FEnt;
   v, nb: Entier;
   moy: Reel;
Procedure SupprimerDernierEltFich(E/S F: FEnt)
   ...
Fin;
   Assigner(F, "ESSAI.DAT");
   Relire(F);
   Tant Que (Non(FDF(F)))
```

```
  Faire
    Fermer(F);
    SupprimerDernierEltFich(F);
    Relire(F);
    nb ← 0;
    moy ← 0;
    Tant Que (Non(FDF(F)))
    Faire
      Lire(F, v);
      moy ← moy + v;
      nb ← nb + 1
    Fait
    Fermer(F);
    Si (nb = 0) alors
      Ecrire("Le Fichier est maintenant vide");
    Sinon
      moy ← moy/nb;
      Ecrire("La Moyenne des elements du fichier est: ", moy);
    Fsi
    Relire(F);
  Fait
  Fermer(F);
Fin.
```

Exercice 9 :

Soit le type suivant :

```
Type
    Produit = Enregistrement
          Code : Entier ;
          Désignation : Chaîne [ 80 ] ;
          Prix : Réel ;
  Fin ;
```

Soit F un fichier de produits. Ecrire une fonction qui vérifie si les éléments de F sont triés par ordre croissant de leur Code.

Solution

```
Type
  Produit  =  Enregistrement
    Code :  Entier ;
    Désignation : Chaîne [ 80 ] ;
    Prix : Réel ;
  Fin ;
```

```
FProd = Fichier de Produit;

Fonction FichierTrie(E/ F: FProd): Booleen
Var
  fTrie: Booleen;
  pr1, pr2: Produit;
Debut
  fTrie ← Vrai;
  Relire(F);
  Si (Non(FDF(F))) alors
     Lire(F, pr1);
     Tant Que (Non(FDF(F)) ET Trie)
     Faire
        Lire(F, pr2);
        Si (pr1.code > pr2.Code) alors
           fTrie ← Faux;
        Sinon
           pr1 ← pr2;
        Fsi
     Fait
  Fsi
  Fermer(F);

  Retourner   fTrie;
Fin;
```

Exercice 10 :

L'utilisation des téléphones portables permet de stocker le répertoire des contacts dans deux fichiers :
- Un fichier ' TEL.DAT ', enregistré sur la mémoire du téléphone ;
- Un fichier ' SIM.DAT ', enregistré sur la mémoire de la carte SIM.

Chaque fichier contient des références d'un contact regroupant : un nom, un prénom et un numéro de téléphone.
Les éléments des deux fichiers sont supposés déjà triés selon le numéro de téléphone.
1) Donnez la syntaxe (les instructions) d'assignation et d'ouverture des deux fichiers.
2) Ecrire une procédure qui permet de stocker les doublons dans un autre fichier. Un élément est un doublon s'il existe (figure) à la fois dans les deux fichiers.

Solution

1-

```
Assigner(FTel,"TEL.DAT") ; Relire(FTel) ;
Assigner(FSim,"SIM.DAT") ; Relire(FSim) ;
```

2-
On supposera que le numero de telephone est un entier et que le type entier contient assez de chiffres pour decrire un numero de telephone.
Les 2 fichiers sont tries par ordre croissant selon le numero de telephone:

```
Type
  Contact=Enregistrement
      Nom,Prenom :Chaine;
      Numero :Entier ;
  Fin ;
  FichContact = Fichier de Contact;
```

Idee:
On vas d'abord ecrire une procedure LireFichier (E/ F:FichContact, S/ v: Contact, S/ fdf:booleen) qui recoit en parametre un fichier F deja ouvert, puis lit un contact dans v s'il reste encore des elements dans F et met fdf a Faux. S'il ne reste plus d'elements dans F cette procedure met fdf a Vrai. Cela permet d'eviter le probleme ou apres lecture du dernier element de F d'avoir FDF a Vrai. Ensuite on vas parcourir les 2 fichiers FTel et Fsim en meme temps en lecture si element courant de Ftel a v1.num est < v2.num ou v2 est element courant de FSim alors il faudra lire le prochain element de FTel, par contre si v2.num < v1.num alors il faudra lire le prochain element de FSim. Le 3eme cas est lorsqu'on a egalite entre v1.num et v2.num et dans ce cas on vas ecrire l'element doublon dans le fichier des doublons et on vas lire les prochains elements de FTel et FSim pour refaire la meme chose.

Exemple:
FTel: 3 - 7 - 11 - 18 - 24 - 32 - 56 - FDF
FSim: 5 - 7 - 9 - 24 - 43 - 66 - 90 - 112 - FDF
Au debut on lit v1=3 et v2=5 et on compare v1 et v2 puisque v1<v2 alors on vas lire le prochain element de FTel et v1=7 et v2=5 et on compare v1 et v2 et puisque maintenant v2<v1 alors on vas lire de FSim et on a ainsi v1=7 et v2=7 et maintenant on a un doublon et on l'ecrit dans Fdouble et les prochains qu'on vas lire ca sera v1=11 et v2=9 et on continue le meme traitement...

```
Type
  Contact=Enregistrement
      Nom,Prenom :Chaine;
      Num :Entier ;
  Fin ;
  FichContact = Fichier de Contact;

Procedure LireFichier (E/ F:FichContact, S/ v: Contact, S/ fdf:
Booleen)
Debut
  Si (FDF(F)) alors
     fdf <- Vrai;
  Sinon
     fdf <- Faux;
     Lire(F, v);
  Fsi
Fin;

Procedure   creerFichDoublons(E/ ftel,fsim :FichContact ; E/S fdouble
:FichContact)
Var
  v1, v2 : Contact ;
  fdf1, fdf2 : Booleen ;
Debut
  Relire(ftel);
  Relire(fsim);
  Reecrire(fdouble);
  LireFichier (ftel, v1, fdf1);
  LireFichier (fsim, v2, fdf2);
  // parcourir les 2 fichiers en meme temps et comparer element par
element
  Tant Que ((fdf1 = Faux) et (fdf2 = Faux)) Faire
     Si (v1.num < v2.num) alors
        // v1.num < v2.num et comme fsim trie par ordre croissant
donc v1.num < tous les
        // elements qui suivent de fsim et donc ne peut pas etre un
doublon
        LireFichier (ftel, v1, fdf1);
     Sinon
        Si (v2.num < v1.num) alors
           // v2.num < v1.num et comme ftel trie par ordre croissant
donc v2.num < tous les
```

```
            // elements qui suivent de ftel et donc ne peut pas etre un
doublon
               LireFichier (fsim, v2, fdf2);
            Sinon  // cas v1.num=v2.num  on a un doublon
               Ecrire(fdouble, v1);
               LireFichier (ftel, v1, fdf1);
               LireFichier (fsim, v2, fdf2);
            Fsi
         Fsi
     Fait
     Fermer(ftel);
     Fermer(fsim);
     Fermer(fdouble);
Fin.

Fin;
```

Exercice 11:

Soit F un fichier d'entiers (supposé existant) composé de séquences de nombres, chaque séquence est une répétition du même nombre (non nul). Toutes les séquences sont séparées par un zéro. Et aucune séquence du même nombre ne se répète dans le fichier.

Nb=7

F 14 14 14 0 5 5 0 29 29 29 29 0 6 6 6

Position 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

1) Ecrire une action paramétrée (Compresser) qui crée un fichier G d'enregistrement contenant pour chaque séquence le nombre représenté ainsi que la longueur de la séquence.

G: {14. 3} – {5, 2} – {29, 4} – {6, 3}

2) En utilisant un seul parcours du fichier G et sans reparcourir le fichier F, trouver la position dans F de la plus longue séquence.

Position = 8

3) Ecrire une action paramétrée (Decompresser) qui permet de reconstruire un fichier H (de même type que F) à partir d'un fichier de même type que G.

Solution

1)

TYPE

```
  EnrG = Enregistrement
     nbr: Entier;
     long: Entier;
  Fin;
  FichG = Fichier de EnrG;

/* On suppose dans les AP suivantes que les assignations son faites
dans l'algorithme. */

Procedure Compresser(E/ F: Fichier d'entier,S/ G: FichG)
Var
  n, cpt : Entier;
  eltG : EnrG;
Debut
  Relire(F);
  Reecrire(G);
  Tant Que(Non(FDF(F)))
  Faire
     Lire(F, n);
     eltG.nbr ← n;
     cpt ← 0;
     Tant Que(Non(FDF(F)) et (n <> 0))
     Faire
        cpt ← cpt + 1;
        Lire(F, n);
     Fait
     Si (FDF(F)) alors
        cpt ← cpt + 1;
     Fsi
     eltG.long ← cpt;
     Ecrire(G, eltG);
  Fait
  Fermer(F);
  Fermer(G);
Fin;

G [Nombre, Longuer]: {14, 3} - {5, 2} - {29, 4} - {6, 3}

2) En utilisant un seul parcours du fichier G et sans reparcourir le
fichier F, trouver la position dans F de la plus longue séquence.
Position = 8

Fonction    PlusGrandeSeq(G: FichG):Entier
Var
```

```
  n, maxSeqLong, posMax, pos : Entier;
  eltG : EnrG;
Debut
  maxSeqLong ← 0;
  pos ← 1;
  Relire(G);
  Tant Que(Non(FDF(G)))
  Faire
    Lire(G, eltG);  /* lecture eltG: nbr, long */
    Si (eltG.long > maxSeqLong) alors
      maxSeqLong ← eltG.long;
      posMax ← pos;
    Fsi
    pos ← pos + eltG.long + 1;
  Fait
  Fermer(G);

  Retourner posMax;
Fin;
```

3) Ecrire une action paramétrée (Decompresser) qui permet de reconstruire un fichier H
(de même type que F) à partir d'un fichier de même type que G.

```
Procedure Decompresser(E/ G: FichG, S/ H: Fichier de Entier);
Var
  eltG : EnrG;
  i: Entier;
Debut
  Relire(G);
  Reecrire(H);
  Tant Que(Non(FDF(G)))
  Faire
    Lire(G, eltG);
    Pour i ← 1 a eltG.long
    Faire
      Ecrire(H, eltG.nbr);
    Fait
    Si (Non(FDF(G)))
        Ecrire(H, 0);
    Fsi
  Fait
  Fermer(G);
  Fermer(H);
```

Fin;

Exercice 12 :

Soit Fdata un fichier d'entiers.
1) Ecrire une procédure InsertF permettant d'insérer un entier N dans un fichier trié par ordre croissant.
2) En utilisant cette procédure, écrire un algorithme permettant de trier le fichier Fdata dans un ordre croissant

Solution

1)

```
Type
   FEnt = Fichier de Entier;
```

Solution avec utilisation de booleen

```
Procedure   InsertF(E/S F: FEnt, E/ N: Entier)
Var
   FTemp : FEnt;
   Superieur : Booleen;
   nbr : Entier;
Debut
   Assigner(FTemp, "FichTemp2");
   Relire(F);
   Reecrire(FTemp);
   Superieur ← Faux;

   /* Recopier tous les entiers de F <= N dans Ftemp */
   Tant Que (NON(FDF(F))  et NON(Superieur))
   Faire
       Lire(F, nbr);
       Si (nbr <= N) alors
           Ecrire(FTemp, nbr);
       Sinon
           Superieur ← Vrai;
       Fsi
  Fait
  /* ecrire N dans FTemp*/
  Ecrire(FTemp, N);
  Si (Superieur) alors
        Ecrire(FTemp, nbr); /* ecrire le premier entier nbr qui est >  N
*/
        /* Recopier tous les entiers qui restent dans F dans FTemp */
       Tant Que (NON(FDF(F))
        Faire
```

```
            Lire(F, nbr);
            Ecrire(FTemp, nbr);
        Fait
    Fsi
    Fermer(F);
    Fermer(FTemp);

    /* Recopier maintenant FTemp dans F */
    Relire(FTemp);
    Reecrire(F);
    Tant Que (NON(FDF(FTemp))
    Faire
        Lire(FTemp, nbr);
        Ecrire(F, nbr);
    Fait
    Fermer(F);
    Fermer(FTemp);

Fin;
```

Procedure InsertF sans booleen

```
Procedure    InsertF(E/S F: FEnt, E/ N: Entier)
Var
    FTemp : FEnt;
    nbr : Entier;
Debut
    Assigner(FTemp, "FichTemp2");
    Relire(F);
    Reecrire(FTemp);
    Si (NON(FDF(F))) alors
        Lire(F, nbr);   // lire le 1er nombre de F
      Tant Que (NON(FDF(F))  et (nbr <= N)) Faire
            Ecrire(FTemp, nbr);
            Lire(F, nbr);
      Fait
      Si (NON(FDF(F))) alors
            Ecrire(FTemp, N);
            Ecrire(FTemp, nbr);     // ecrire le 1er nbr > N
          Tant Que (NON(FDF(F))) Faire  // ecrire les nbr qui suivent
              Lire(F, nbr);
              Ecrire(FTemp, nbr);
          Fait
      Sinon
          Si(nbr <= N) alors
```

```
            Ecrire(FTemp, nbr);
            Ecrire(FTemp, N);
         Sinon
            Ecrire(FTemp, N);
            Ecrire(FTemp, nbr);
         Fsi
      Fsi
   Sinon  // cas fichier F vide
      Ecrire(FTemp, N);
   Fsi
   Fermer(F);
   Fermer(FTemp);

   /* Recopier maintenant FTemp dans F */
   Relire(FTemp);
   Reecrire(F);
   Tant Que (NON(FDF(FTemp))
   Faire
       Lire(FTemp, nbr);
       Ecrire(F, nbr);
   Fait
   Fermer(F);
   Fermer(FTemp);
Fin;
```

2)

```
Algorithme    TriFichierEntiers;
Type
   FEnt = Fichier de Entier;
Var
  F, FT : FEnt;
  elt : Entier;

Procedure    InsertF(E/S F: FEnt, E/ N: Entier)
  ...
Fin;


Debut
    Assigner(F, "Fdata");
    /* Considerons le fichier temporaire FTemporaire. C'est ce fichier
qui va contenir les entiers tries
```

```
        lorsqu'on va appeler la procedure InsertF. Pour s'assurer que
FTemporaire est vide au debut on
        va faire un Reecrire suivi d'un Fermer.
     */
    Assigner(FT, "Ftemporaire");
    Reecrire(FT);
    Fermer(FT);

    Relire(F);
    Tant Que (NON(FDF(F)))
    Faire
         Lire(F, elt);
         InsertF(FT, elt);
    Fait
    Fermer(F);

   /* Recopier maintenant FT dans F */
   Relire(FT);
   Reecrire(F);
   Tant Que (NON(FDF(FT))
   Faire
       Lire(FT, elt);
       Ecrire(F, elt);
   Fait
   Fermer(F);
   Fermer(FT);
Fin.
```