```
/*
Ce programme range des objets par ordre de ressemblance
en utilisant la matrice des distances entre eux deux a deux
*/
// Imprimer la liste du programme au format paysage !

#include <iostream>
#include <string>
#include <fstream>
#include <stdio.h> // pour avoir fread et fwrite
#include <sstream> // pour avoir les fonctions de traitement de flux de caracteres
#include <time.h>

using namespace std ;
const int NbObjMax = 20 ; // Nombre maximal d'objets autorises : NbObj <= NbObjMax.
const char Version[60] = "Version 1.02 - (c) Sellig Zed, 4 janvier 2009" ;
// Pour plus de lisibilite, les objets sont ranges dans les cellules 1 a NbObj.
// Les tableaux sont donc declares avec une dimension de NbObjMax+1.
// =====10========20========30========40========50========60========70========80========90=======100=======110
// Le programme principal assure l'acquisition des donnees et l'interface utilisateur.
// La fonction "Range" effectue l'integralite du processus de classement.
// =====10========20========30========40========50========60========70========80========90=======100=======110

//
//                          *************       **                      **
//                          **************       **                    **
//                          **            **       **                  **
//                          **             **       **                **
//                          **            **         **              **
//                          **           **           **            **
//                          *************             **          **
//                          ************              **        **
//                          **          ****          **      **
//                          **          *   *           **    **
//                          **          ****              **  **
//                          **          *                   ****
//                          **          *                     **
//
//
```

```
bool Range(int NO,float D[NbObjMax+1][NbObjMax+1],int CL[NbObjMax+1],int AG[NbObjMax+1])
   { // Range a besoin du nombre d'objets NO et de la matrice de distance D.
   // D[i][j] est la distance entre l'objet i et l'objet j, rangee en ligne i,
   // et colonne j, avec j=1, ..., NbObj et i dans ]j,NbObj].
   // Range fournit en sortie le classement CL des objets dans l'ordre de plus
   // grande ressemblance et le tableau AG donnant l'ordre de construction
   // du classement par la technique du plus proche voisin
   // Declarations internes
   bool traitement_OK = false ;      // indique si le traitement s'est bien fini
   int NP = NO*(NO-1)/2 ;            // nombre de paires d'objets
   int ELM_1[NP+1], ELM_2[NP+1] ;    // de 1 a NP, contiennent les 2 membres des paires d'objets
   float Dpaire[NP+1] ;              // distance entre les 2 elements de la paire
   bool PRIS[NP+1] ;                 // indique si une paire fut prise lors du rangement
   bool OPRIS[NO+1] ;                // indique si un objet fut pris lors du rangement
   int PairEnCours = 0, ilig, icol, Bulle, oloop, iAG = 3 ;     // Auxiliaires de comptage ou de boucle
   float Dtemp ;                     // Stockage temporaire d'une distance dans le processus
   int ORDRE[NO+1], TAG[NO+1] ;      // numero d'ordre d'un objet dans le classement, "temps" d'agregation
   int GAUCHE, DROITE, ORDREMIN, ORDREMAX ;      // Objets a gauche et a droite, leurs numeros d'ordre
   int Nouveau_GAUCHE, Nouveau_DROITE, Nouveau ; // Prochains objets a gauche et a droite, nouvelle paire
   // Impressions dans un fichier log
   FILE *Rlog ;
   Rlog= fopen("Range.log","wt") ;
   if (Rlog == NULL)
      {
      printf("\n\nImpossible d'ouvrir le fichier log !!\n\n") ;
      return 1 ;
      }
   // L'algoritme comporte trois etapes.
   // ..................................................................................... Etape 1 ..
//   printf("\nCommencement de la fonction de rangement.\n") ;
   fprintf(Rlog,"\nCommencement de la fonction de rangement.\n") ;
   // La premiere etape est le classement des paires d'objets dans les deux vecteurs ELM_1 et ELM_2
   // par ordre croissant de la distance Dpaire entre les deux objets
   // Nota: Dpaire[p]=D[ELM_1[p]][ELM_2[p]] ou =D[ELM_2[p]][ELM_1[p]] (ligne > colonne).
//   printf("-------------------------------------------------------\n") ;
//   printf("Etape 1 : rangement des paires par distance croissante :\n") ;
   fprintf(Rlog,"-------------------------------------------------------\n") ;
   fprintf(Rlog,"Etape 1 : rangement des paires par distance croissante :\n") ;
   Dpaire[0] = 0.0 ;              // Case d'indice 0 utilisee pour l'initialisation de la boucle suivante
```

```
   for (icol=1;icol<NO+1;icol++)   // On parcourt toutes les colonnes de D de 1 a NO (=tous les objets)
      {
      OPRIS[icol] = false ;                  // On profite de cette boucle pour initialiser OPRIS.
      for (ilig=icol+1;ilig<NO+1;ilig++)     // Pour chaque colonne, toutes les lignes sous la diagonale de D,
         {                                   // cette boucle range les D[i][j] triees dans Dpaire
         Bulle = PairEnCours ;               // La Bulle de tri va remonter depuis la paire vue au tour d'avant
         PairEnCours=PairEnCours+1 ;         // On incremente le compteur de la paire en cours
         Dtemp=D[ilig][icol] ;               // La distance traitee dans ce tour est mise dans Dtemp
         ELM_1[PairEnCours]=ilig ;           // Temporairement, un des 2 objets est mis en ELM_1[PairEnCours],
         ELM_2[PairEnCours]=icol ;           // ... l'autre en ELM_2[PairEnCours], ...
         Dpaire[PairEnCours]=Dtemp ;         // ... et la distance correspondante en Dpaire[PairEnCours].
         PRIS[PairEnCours] = false ;         // On profite de cette boucle pour initialiser PRIS (cf. etape 2)
         while (Dtemp<Dpaire[Bulle])         // Cette boucle va remonter la paire en cours tant que Dtemp est
            {                                // inferieure a la valeur au-dessus d'elle dans le tableau Dpaire
            ELM_1[Bulle+1]=ELM_1[Bulle] ;    // Pour cela, on commence par descendre un element de la paire...
            ELM_2[Bulle+1]=ELM_2[Bulle] ;    // ... puis l'autre ...
            Dpaire[Bulle+1]=Dpaire[Bulle] ; // et leur distance dans la case en dessous.
            ELM_1[Bulle]=ilig ;              // Puis on met le premier element en cours dans la case liberee
            ELM_2[Bulle]=icol ;              // ... puis le second ...
            Dpaire[Bulle]=Dtemp ;            // ... ainsi que leur distance.
            Bulle=Bulle-1 ;                  // On fait remonter la Bulle d'un cran pour recommencer le while.
            }                                // A la fin du while, toutes les distances sont classees.
         }
      }
   for (ilig=1;ilig<NP+1;ilig++)
      {                                      // Impression du classement obtenu pour les paires.
//      printf("ilig= %d, icol= %d, dist= %f\n",ELM_1[ilig],ELM_2[ilig],Dpaire[ilig]) ;
      fprintf(Rlog,"ilig= %d, icol= %d, dist= %f\n",ELM_1[ilig],ELM_2[ilig],Dpaire[ilig]) ;
      }
   // ..................................................................................... Etape 2 ..
   // La deuxieme etape est de ranger les objets par la technique du plus proche voisin
   // Le tableau ORDRE contient un ordre classant les objets de gauche a droite.
   // On numerote d'abord les deux plus proches voisins
//   printf("-----------------------------------------------------\n") ;
//   printf("Etape 2 : Progression de voisin en plus proche voisin.\n") ;
   fprintf(Rlog,"-----------------------------------------------------\n") ;
   fprintf(Rlog,"Etape 2 : Progression de voisin en plus proche voisin.\n") ;
   ORDRE[ELM_1[1]]=0 ;           // On choisit arbitrairement l'element 1 au rang 0 (indifferent / symetrie)
   ORDRE[ELM_2[1]]=1 ;           // ... et l'autre place en 1, c'est-a-dire a sa droite.
```

```
   AG[1]= ELM_1[1] ;             // Le premier a etre "agrege" est donc ELM_1[1] ...
   AG[2]= ELM_2[1] ;             // ... et le deuxieme ELM_2[1]
   PRIS[1]= true ;               // La premiere paire est notee comme prise ...
   OPRIS[ELM_1[1]] = true ;      // ... ainsi que son premier element ...
   OPRIS[ELM_2[1]] = true ;      // ... et que son second element.
   GAUCHE = ELM_1[1] ;           // A ce stade, l'element le plus a gauche est donc ELM_1[1] ...
   DROITE = ELM_2[1] ;           // ... et le plus a droite ELM_2[1]
   ORDREMIN = 0 ;                // A ce stade, le plus petit numero d'ordre (extremite gauche) est 0 ...
   ORDREMAX = 1 ;                // ... et le plus grand (extremite droite) est 1.
//   printf(".....................................................\n") ;
//   printf("Etape 2-1 : initialisation par la plus petite distance.\n") ;        // Impression initialisation
//   printf("GAUCHE= %d, DROITE= %d, ELM_1[1]= %d, ELM_2[1]= %d, PRIS[1]= %d\n",
//                  GAUCHE,DROITE,ELM_1[1],ELM_2[1],PRIS[1]) ;
   fprintf(Rlog,".....................................................\n") ;
   fprintf(Rlog,"Etape 2-1 : initialisation par la plus petite distance.\n") ;    // Impression initialisation
   fprintf(Rlog,"GAUCHE= %d, DROITE= %d, ELM_1[1]= %d, ELM_2[1]= %d, PRIS[1]= %d\n",
                  GAUCHE,DROITE,ELM_1[1],ELM_2[1],PRIS[1]) ;
   // A chaque etape de la boucle suivante, on cherche le plus proche voisin a droite ou a gauche,
   // en parcourant le tableau des paires de haut en bas (distances croissantes).
//   printf(".....................................................\n") ;
//   printf("Etape 2-2 : boucle for de rangement par plus proche voisin.\n") ;
//   printf("            *****\n") ;
   fprintf(Rlog,".....................................................\n") ;
   fprintf(Rlog,"Etape 2-2 : boucle for de rangement par plus proche voisin.\n") ;
   fprintf(Rlog,"            *****\n") ;
   for (oloop=3;oloop<NO+1;oloop++) // pour les nouveaux objets a entrer
      {                             // comme deux sont deja pris, la boucle commence a 3
//      printf("\nPassage dans la boucle de rangement avec oloop = %d\n",oloop) ;
//      printf("GAUCHE= %d et DROITE= %d\n",GAUCHE,DROITE) ;
      fprintf(Rlog,"\nPassage dans la boucle de rangement avec oloop = %d\n",oloop) ;
      fprintf(Rlog,"GAUCHE= %d et DROITE= %d\n",GAUCHE,DROITE) ;
      Nouveau = 1 ;                 // A chaque tour dans "for", la nouvelle paire est cherchee depuis 1
      while ( (PRIS[Nouveau]) ||    // Tant que la paire Nouveau est deja prise ou que ...
              (  (GAUCHE!=ELM_1[Nouveau]) && (GAUCHE!=ELM_2[Nouveau])       // ... elle ne comporte pas GAUCHE
              && (DROITE!=ELM_1[Nouveau]) && (DROITE!=ELM_2[Nouveau]) ) )   // ... ni DROITE.
        {      // Rappel PRIS[1] est true (initialisation) donc cette boucle est parcourue au moins une fois.
        // On passe donc dans la boucle while **si Nouveau NE CONVIENT PAS**, pour l'incrementer.
//        printf("Boucle while pour chercher la prochaine paire, Nouveau= %d non pris\n",Nouveau) ;
//        printf("  car ELM_1[%d]= %d et ELM_2[%d]= %d et PRIS[%d]= %d.\n",
```

```
//                      Nouveau,ELM_1[Nouveau],Nouveau,ELM_2[Nouveau],Nouveau,PRIS[Nouveau]) ;
         fprintf(Rlog,"Boucle while pour chercher la prochaine paire, Nouveau= %d non pris\n",Nouveau) ;
         fprintf(Rlog,"  car ELM_1[%d]= %d et ELM_2[%d]= %d et PRIS[%d]= %d.\n",
                     Nouveau,ELM_1[Nouveau],Nouveau,ELM_2[Nouveau],Nouveau,PRIS[Nouveau]) ;
         Nouveau = Nouveau + 1 ;
         }
      // En sortie de boucle while (sans y passer la derniere fois!),
      // Nouveau est le numero de la nouvelle paire a prendre en compte
      // mais 4 cas sont possibles puisque GAUCHE ou DROITE peuvent etre ELM_1[Nouveau] ou ELM_2[Nouveau].
//      printf("Sortie de while : ") ;
      fprintf(Rlog,"Sortie de while : ") ;
      if ( (GAUCHE==ELM_1[Nouveau]) || (GAUCHE==ELM_2[Nouveau]) ) // si la nouvelle paire contient GAUCHE
         {
         if (GAUCHE==ELM_1[Nouveau])                        // si GAUCHE est ELM_1[Nouveau] ...
            {
            Nouveau_GAUCHE = ELM_2[Nouveau] ;               // ... le Nouveau_GAUCHE est l'autre ELM_2[Nouveau]
            OPRIS[ELM_2[Nouveau]] = true ;                  // ... qu'on note comme pris
            AG[iAG]= Nouveau_GAUCHE ;                       // ... qu'on note comme etant le iAGeme pris ...
            iAG = iAG + 1 ;                                 // puis on incremente iAG.
            }
         else                                               // au contraire, si GAUCHE est ELM_2[Nouveau] ...
            {
            Nouveau_GAUCHE = ELM_1[Nouveau] ;               // ... le Nouveau_GAUCHE est l'autre ELM_1[Nouveau]
            OPRIS[ELM_1[Nouveau]] = true ;                  // ... qu'on note comme pris
            AG[iAG]= Nouveau_GAUCHE ;                       // ... qu'on note comme etant le iAGeme pris ...
            iAG = iAG + 1 ;                                 // puis on incremente iAG.
            }
//        printf("Cas 1 - Nouveau= %d retenu pour la gauche.\n\n", Nouveau) ;
//        printf("Elimination des paires non pertinentes: PRIS = true (rendu par 1 ci-dessous)\n", Nouveau) ;
         fprintf(Rlog,"Cas 1 - Nouveau= %d retenu pour la gauche.\n\n", Nouveau) ;
         fprintf(Rlog,"Elimination des paires non pertinentes : PRIS = true (rendu par 1 ci-dessous)\n",
                                 Nouveau) ;
         for (ilig=Nouveau;ilig<NP+1;ilig++) // Pour toutes les paires nouvellement candidates a l'elimination
            {               // si l'objet anciennement a gauche est dans une paire, on ne doit plus la prendre
            PRIS[ilig]= OPRIS[ELM_1[ilig]] && OPRIS[ELM_2[ilig]] ; // Elimine G------D
            if ((GAUCHE==ELM_1[ilig]) || (GAUCHE==ELM_2[ilig]))
               {                        // Elimine G--G------ en
               PRIS[ilig]= true ;       // marquant comme prises la nouvelle paire et toutes les autres ...
               }                        // ... contenant aussi GAUCHE.
```

```
//            printf("GAUCHE= %d, ELM_1[%d]= %d, ELM_2[%d]= %d, PRIS[%d]= %d\n",
//                           GAUCHE,ilig,ELM_1[ilig],ilig,ELM_2[ilig],ilig,PRIS[ilig]) ;
            fprintf(Rlog,"GAUCHE= %d, ELM_1[%d]= %d, ELM_2[%d]= %d, PRIS[%d]= %d\n",
                           GAUCHE,ilig,ELM_1[ilig],ilig,ELM_2[ilig],ilig,PRIS[ilig]) ;
            }
         GAUCHE = Nouveau_GAUCHE ;       // Le nouvel element devient l'extremite gauche
         ORDREMIN = ORDREMIN - 1 ;       // Le numero d'ordre minimal est decremente en consequence
         ORDRE[GAUCHE] = ORDREMIN ;      // Le nouvel element a gauche recoit ce numero d'ordre mis a jour
         }
      else                                                    // si la nouvelle paire contient DROITE
         {
         if (DROITE==ELM_1[Nouveau])                        // si DROITE est ELM_1[Nouveau] ...
            {
            Nouveau_DROITE = ELM_2[Nouveau] ;               // ... le Nouveau_DROITE est l'autre ELM_2[Nouveau]
            OPRIS[ELM_2[Nouveau]] = true ;                  // ... qu'on note comme pris
            AG[iAG]= Nouveau_DROITE ;                       // ... qu'on note comme etant le iAGeme pris ...
            iAG = iAG + 1 ;                                 // puis on incremente iAG.
            }
         else                                               // au contraire, si DROITE est ELM_2[Nouveau] ...
            {
            Nouveau_DROITE = ELM_1[Nouveau] ;               // ... le Nouveau_DROITE est l'autre ELM_1[Nouveau]
            OPRIS[ELM_1[Nouveau]] = true ;                  // ... qu'on note comme pris
            AG[iAG]= Nouveau_DROITE ;                       // ... qu'on note comme etant le iAGeme pris ...
            iAG = iAG + 1 ;                                 // puis on incremente iAG.
            }
//          printf("Cas 2 - Nouveau= %d retenu pour la droite.\n\n", Nouveau) ;
//          printf("Elimination des paires non pertinentes: PRIS = true (rendu par 1 ci-dessous)\n", Nouveau) ;
         fprintf(Rlog,"Cas 2 - Nouveau= %d retenu pour la droite.\n\n", Nouveau) ;
         fprintf(Rlog,"Elimination des paires non pertinentes : PRIS = true (rendu par 1 ci-dessous)\n",
                                 Nouveau) ;
         for (ilig=Nouveau;ilig<NP+1;ilig++)
            { // si l'objet anciennement a droite est dans une paire, on ne doit plus la prendre
            PRIS[ilig]= OPRIS[ELM_1[ilig]] && OPRIS[ELM_2[ilig]] ; // Elimine G------D
            if ((DROITE==ELM_1[ilig]) || (DROITE==ELM_2[ilig]))
               {                          // Elimine ------D--D en
               PRIS[ilig]= true ;         // marquant comme prises la nouvelle paire et toutes les autres ...
               }                          // ... contenant aussi DROITE.
//             printf("DROITE= %d, ELM_1[%d]= %d, ELM_2[%d]= %d, PRIS[%d]= %d\n",
//                           DROITE,ilig,ELM_1[ilig],ilig,ELM_2[ilig],ilig,PRIS[ilig]) ;
```

```
            fprintf(Rlog,"DROITE= %d, ELM_1[%d]= %d, ELM_2[%d]= %d, PRIS[%d]= %d\n",
                           DROITE,ilig,ELM_1[ilig],ilig,ELM_2[ilig],ilig,PRIS[ilig]) ;
            }
         DROITE = Nouveau_DROITE ;       // Le nouvel element devient l'extremite droite
         ORDREMAX = ORDREMAX + 1 ;       // Le numero d'ordre maximal est incremente en consequence
         ORDRE[DROITE] = ORDREMAX ;      // Le nouvel element a droite recoit ce numero d'ordre mis a jour
         }
//     printf("\nSortie de boucle pour oloop= %d, ORDREMIN= %d, ORDREMAX= %d\n",oloop,ORDREMIN,ORDREMAX) ;
//     printf("            *****\n") ;
     fprintf(Rlog,"\nSortie de boucle pour oloop= %d, ORDREMIN= %d, ORDREMAX= %d\n",oloop,ORDREMIN,ORDREMAX) ;
     fprintf(Rlog,"            *****\n") ;
      }
   // ..................................................................................... Etape 3 ..
   // La troisieme etape consiste a renumeroter l'ordre dans le sens croissant depuis 1 puis a recuperer le
   // classement dans CL et a stocker l'ordre de traitement de chaque objet dans le processus de rangement.
//      printf("-------------------------------------------------------------\n") ;
//      printf("Etape 3 : Stockage du classement et de l'ordre de construction.\n") ;
      fprintf(Rlog,"-------------------------------------------------------------\n") ;
      fprintf(Rlog,"Etape 3 : Stockage du classement et de l'ordre de construction.\n") ;
      for (oloop=1;oloop<NO+1;oloop++)
         {
         CL[ORDRE[oloop]+1-ORDREMIN]= oloop ;
         TAG[AG[oloop]]= oloop ;            // contient le "temps" d'inclusion de l'objet AG[oloop]
         }
//      printf("\nClassement obtenu dans la fonction : \n") ;
      fprintf(Rlog,"\nClassement obtenu dans la fonction : \n") ;
      for (oloop=1;oloop<NO+1;oloop++)
         {
//         printf("CL[%d]= %d, mis en place au temps %d\n",oloop,CL[oloop],TAG[CL[oloop]]) ;
         fprintf(Rlog,"CL[%d]= %d, mis en place au temps %d\n",oloop,CL[oloop],TAG[CL[oloop]]) ;
         }
//   printf("\nFin de la fonction de rangement !\n") ;
//   printf("*************************\n") ;
   fprintf(Rlog,"\nFin de la fonction de rangement !\n") ;
   fprintf(Rlog,"*************************\n") ;
   fclose(Rlog) ;
   traitement_OK = true ;
   return traitement_OK ;
   }
```

```
// =====10========20========30========40========50========60========70========80========90=======100=======110
int main () // ce programme principal fait l'acquisition des distances et l'interface utilisateur
    {
    // Declarations
    int i, j, ifor1, NbObj = 0, NbDist = 0, input_len = 0, taille_distance = 0 ;
    char c = 0 ;
    bool input_OK = false, Rangement = false ;
    FILE *pDistance, *Result ;
    float Distance[NbObjMax+1][NbObjMax+1] ;  // premier indice i = ligne, second j = colonne
                                              // les vraies distances sont sous la diagonale: i>j
    int Classement[NbObjMax+1], Agreg[NbObjMax+1] ;
    char nom_distance[256], tampon[256] ;
    std::string ligne;                        // variable contenant chaque ligne lue dans un fichier

    // Heure de l'execution
    time_t temps_present;
    time(&temps_present); // Determiner le temps actuel en secondes

    printf("\n\n*****************************************************\n");
    printf("Ce programme pp_voisin range des objets par ordre de\n") ;
    printf("ressemblance par un algorithme de plus proche voisin.\n") ;
    printf("\n*** %s ***\n",Version) ;

    // Lecture des distances
    printf("\n\n*************************************************\n");
    printf("Le fichier des distances est un simple fichier texte, rempli\n") ;
    printf("auparavant avec le bloc-notes. La ligne 1 contient le nombre d'objets.\n") ;
    printf("Les suivantes ont le format: ligne espace colonne espace distance,\n") ;
    printf("avec ligne>colonne.\n") ;
    input_OK = false ;
    while (!input_OK)
       {
       printf("Donner le nom du fichier des distances (avec l'extension .txt) : ") ;
       gets(nom_distance);
       input_len = strlen(nom_distance) ;
       input_OK = (input_len>4) ;
       input_OK = input_OK &&
                  ( nom_distance[input_len - 4] == '.' ) ;
```

```
        input_OK = input_OK &&
                   ( (nom_distance[input_len - 3] == 't') || (nom_distance[input_len - 3] == 'T') ) ;
        input_OK = input_OK &&
                   ( (nom_distance[input_len - 2] == 'x') || (nom_distance[input_len - 2] == 'X') ) ;
        input_OK = input_OK &&
                   ( (nom_distance[input_len - 1] == 't') || (nom_distance[input_len - 1] == 'T') ) ;
        if (!input_OK) {printf("******** ERREUR :\n") ; }
        }
     pDistance = fopen (nom_distance,"rb") ;
     if (pDistance != NULL) // S'il n'y a pas d'erreur d'ouverture
        {
        fseek (pDistance , 0 , SEEK_END);
        taille_distance = ftell (pDistance);
        printf("\nContenu du fichier distance %s :\n\n",nom_distance) ;
        rewind(pDistance) ;
        for (i=0;i<taille_distance;i++)
           {
           fread(&c,1,1,pDistance) ;  // On lit un signe du message et
           printf("%c",c) ;           // on le sort a l'ecran
           }
        fclose(pDistance) ;
        printf("\n\n*************************************************\n");
        printf("\nTaille du fichier : %d car (y compris CR et LF).",taille_distance) ;

        printf("\n\nEst-ce le bon fichier ? (o/n) : ") ;
        scanf("%c",&c) ;
        }
        else // S'il y a une erreur d'ouverture
        {
        c = 'n' ;
        }

Result= fopen("Resultat.txt","wt") ;
if (Result != NULL)
   {
   if ((c != 'n') && (c != 'N'))
        {    // Cas ou le fichier entrant est correct
        gets(tampon) ; // Vidage du reste de flux entrant - le contenu de tampon est sans importance ici
//       printf("\n\n*************************************************\n");
```

```
//       printf("*** Traduction du texte entrant en nombres exploitables ***\n");
        fprintf(Result,"Date d'execution du programme : %s", ctime(&temps_present));
        fprintf(Result,"\n\n*************************************************\n");
        fprintf(Result,"*** Distances fournies au programme ***\n");
        std::ifstream fichier(nom_distance); // le constructeur ifstream permet d'ouvrir un fichier en lecture
        if (std::getline(fichier,ligne))     // lecture du nombre d'objets
           {
           istringstream flux_lu(ligne) ;
           flux_lu >> NbObj ;
//           printf("\nNb objets= %d.\n",NbObj) ;
           fprintf(Result,"\nNb objets= %d.\n",NbObj) ;
           }
        NbDist=NbObj*(NbObj-1) ;
        for (ifor1=0;ifor1<NbDist;ifor1++)
           {
           if (std::getline(fichier,ligne)) // lecture paire - distance
              {
              istringstream flux_lu(ligne) ;
              flux_lu >> i ;
              flux_lu >> j ;
              flux_lu >> Distance[i][j] ;
//              printf("En ligne i = %d, colonne j = %d, la distance est %f.\n",i,j,Distance[i][j]) ;
              fprintf(Result,"En ligne i = %d, colonne j = %d, la distance est %f.\n",i,j,Distance[i][j]) ;
              }
           }
        printf("*************************************************\n");
        fprintf(Result,"*************************************************\n");
        Rangement=Range(NbObj,Distance,Classement,Agreg) ;
        if (Rangement)
           {
           printf("Sorties transmises par la fonction de rangement :\n") ;
           printf("   A - Classement par ordre de ressemblance :\n") ;
           fprintf(Result,"Sorties transmises par la fonction de rangement :\n") ;
           fprintf(Result,"   A - Classement par ordre de ressemblance :\n") ;
           for (ifor1=1;ifor1<NbObj+1;ifor1++)
             {
             printf("      Classement[%d]= %d\n",ifor1,Classement[ifor1]) ;
             fprintf(Result,"      Classement[%d]= %d\n",ifor1,Classement[ifor1]) ;
             }
```

```
            printf("\n   B - Ordre de construction du classement :\n") ;
            fprintf(Result,"\n   B - Ordre de construction du classement :\n") ;
            for (ifor1=1;ifor1<NbObj+1;ifor1++)
              {
              printf("      Agreg[%d]= %d\n",ifor1,Agreg[ifor1]) ;
              fprintf(Result,"      Agreg[%d]= %d\n",ifor1,Agreg[ifor1]) ;
              }
            }
          printf("\nCes sorties sont reproduites dans le fichier Resultat.txt.\n") ;
          printf("\nTaper sur Entree pour terminer.\n") ;
          scanf("%c",&c) ;
          return 0 ; // Pas d'interruption sur erreur.
          }
          else // Cas ou le fichier entrant est mauvais
          {
          gets(tampon) ; // Vidage du reste de flux entrant - le contenu de tampon est sans importance ici
          printf("\n\nRevoir le fichier entrant et relancer le programme.") ;
          printf("\nTaper sur Entree pour fermer le programme.\n") ;
          scanf("%c",&c) ;
          fprintf(Result,"\n\nRevoir le fichier entrant et relancer le programme.") ;
          return 1 ; // Interruption sur erreur.
          }
       fclose(Result) ;
       }
       else
       {
       gets(tampon) ; // Vidage du reste de flux entrant - le contenu de tampon est sans importance ici
       printf("\n\nImpossible d'ouvrir le fichier Resultat !!\n\n") ;
       printf("\nRevoir les droits sur le dossier et/ou le fichier.\n") ;
       printf("\nTaper sur Entree pour fermer le programme.\n") ;
       scanf("%c",&c) ;
       return 1 ; // Interruption sur erreur.
       }

}
```