TD8 : Tri fusion (2/2)

Exercice 1 : Tri fusion sur fichier

a. Ecrire en langage algorithmique la procédure de tri par fusion d'un fichier, en supposant que vous disposez déjà des procédures « eclatement » et « fusion » (voir les entêtes ci-dessous). La procédure de tri doit appeler les procédures « eclatement » et « fusion » jusqu'à ce que le fichier soit complètement trié.

Procédure eclatement (nomFicX : chaîne de caractères, lg : entier, nomFicA : chaîne de caractères, nomFicB : chaîne de caractères)
Précondition : le fichier appelé nomFicX contient des monotonies de longueur lg, sauf peut-être la dernière qui peut être plus courte
Postconditions : Les monotonies de nomFicX sont réparties (1 sur 2) dans des fichiers appelés nomFicA et nomFicB : la première monotonie est copiée dans le fichier A, la seconde dans le fichier B, la troisième dans le A, etc. Si ces fichiers existaient déjà, leur contenu est écrasé. Le fichier A peut recueillir une monotonie de plus que le fichier B, et cette dernière monotonie peut être de longueur inférieure à lg. Si au contraire ficA et ficB recueillent le même nombre de monotonies, la dernière monotonie écrite dans ficB peut être de longueur inférieure à lg.
Paramètres en mode donnée : nomFicX, nomFicA, nomFicB, lg

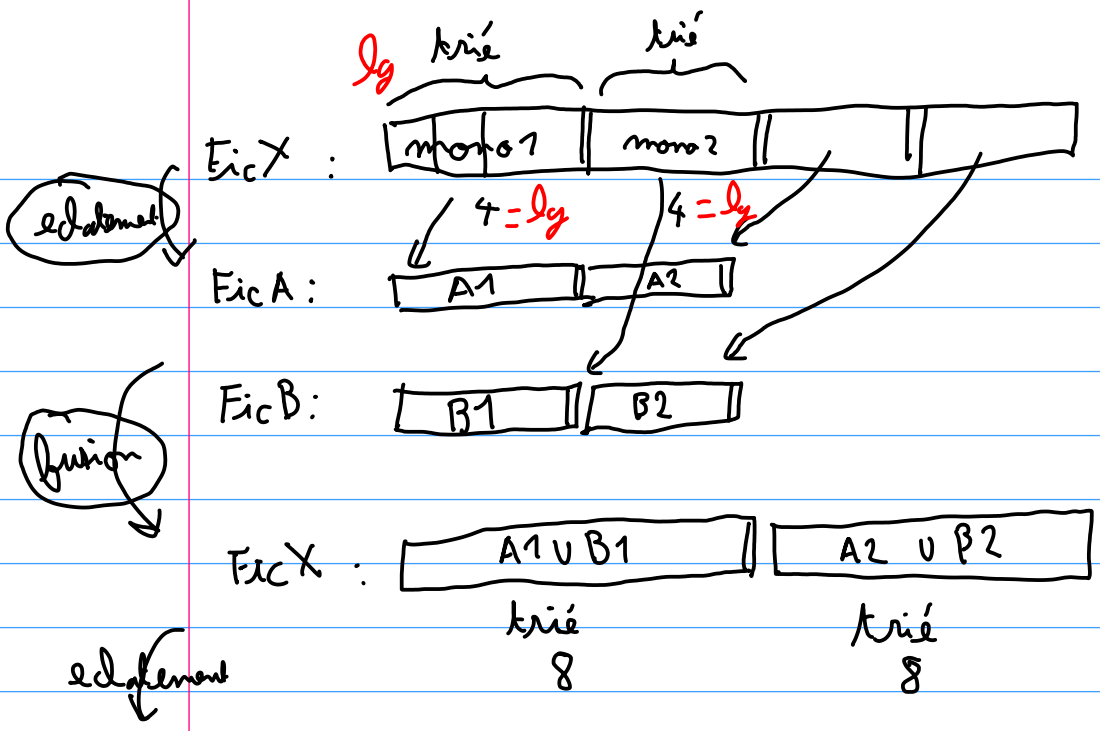
Remarque : Les chaînes de caractères contenant les noms des fichiers ne vont pas être affectées par la procédure, d'où le passage en mode donnée, mais les fichiers désignés par nomFicA et nomFicB vont être affectés, comme cela est précisé dans les post-conditions.

Procédure fusion (nomFicA : chaîne de caractères, nomFicB : chaîne de caractères, lg : entier, nomFicX : chaîne de caractères, nbMonoDansX : entier)
Préconditions : les fichiers appelés nomFicA et nomFicB contiennent des monotonies de longueur lg. FicA peut contenir une monotonie de plus (éventuellement incomplète) que FicB. Si au contraire FicA et FicB contiennent le même nombre de monotonies, alors la dernière monotonie de FicB peut être incomplète.
Postconditions : le fichier appelé nomFicX contient nbMonoDansX monotonies de longueur 2*lg, la dernière pouvant être plus courte. Ces monotonies résultent de la fusion 2 à 2 des monotonies de FicA et de FicB. Si FicX existait déjà, son ancien contenu est écrasé.
Paramètres en mode donnée : nomFicX, nomFicA, nomFicB, lg
Paramètre en mode donnée-résultat : nbMonoDansX

b. Donnez le code de la procédure d'éclatement en langage C++.

Exercice 2 : Tri fusion interne

Ecrire une version itérative (c'est-à-dire non récursive) de l'algorithme du tri fusion d'un tableau de réels. Cette version du tri fusion n'utilisera pas de fichiers, on pourra utiliser de la mémoire sur le tas pour stocker des (sous-) tableaux.



if (!fic.eof()) .... teste si la lecture précédente dans fic s'est bien passée

TD8 : Tri fusion (2/2)

Exercice 1 : Tri fusion sur fichier

a. Ecrire en langage algorithmique la procédure de tri par fusion d'un fichier, en supposant que vous disposez déjà des procédures « eclatement » et « fusion » (voir les entêtes ci-dessous). La procédure de tri doit appeler les procédures « eclatement » et « fusion » jusqu'à ce que le fichier soit complètement trié.

Procédure eclatement (nomFicX : chaîne de caractères, lg : entier, nomFicA : chaîne de caractères, nomFicB : chaîne de caractères)
Précondition : le fichier appelé nomFicX contient des monotonies de longueur lg, sauf peut-être la dernière qui peut être plus courte
Postconditions : Les monotonies de nomFicX sont réparties (1 sur 2) dans des fichiers appelés nomFicA et nomFicB : la première monotonie est copiée dans le fichier A, la seconde dans le fichier B, la troisième dans le A, etc. Si ces fichiers existaient déjà, leur contenu est écrasé. Le fichier A peut recueillir une monotonie de plus que le fichier B, et cette dernière monotonie peut être de longueur inférieure à lg. Si au contraire ficA et ficB recueillent le même nombre de monotonies, la dernière monotonie écrite dans ficB peut être de longueur inférieure à lg.
Paramètres en mode donnée : nomFicX, nomFicA, nomFicB, lg

Remarque : Les chaînes de caractères contenant les noms des fichiers ne vont pas être affectées par la procédure, d'où le passage en mode donnée, mais les fichiers désignés par nomFicA et nomFicB vont être affectés, comme cela est précisé dans les post-conditions.

Procédure fusion (nomFicA : chaîne de caractères, nomFicB : chaîne de caractères, lg : entier, nomFicX : chaîne de caractères, nbMonoDansX : entier)
Préconditions : les fichiers appelés nomFicA et nomFicB contiennent des monotonies de longueur lg. FicA peut contenir une monotonie de plus (éventuellement incomplète) que FicB. Si au contraire FicA et FicB contiennent le même nombre de monotonies, alors la dernière monotonie de FicB peut être incomplète.
Postconditions : le fichier appelé nomFicX contient nbMonoDansX monotonies de longueur 2*lg, la dernière pouvant être plus courte. Ces monotonies résultent de la fusion 2 à 2 des monotonies de FicA et de FicB. Si FicX existait déjà, son ancien contenu est écrasé.
Paramètres en mode donnée : nomFicX, nomFicA, nomFicB, lg
Paramètre en mode donnée-résultat : nbMonoDansX

```
Procédure tri_par_fusion (nomFic : chaîne de caractères)
Précondition : le fichier appelé nomFic contient des éléments
Postcondition : le fichier appelé nomFic contient les mêmes éléments, mais triés.
Paramètres en mode donnée : nomFic
Variables locales :
  longueur, nbMonotonies : entier
  nomA, nomB : chaînes de caractères
Début
  longueur ← 1
  nomA ← « fichierAnnexeA.txt »
  nomB ← « fichierAnnexeB.txt »
  Répéter
    eclatement(nomFic, longueur, nomA, nomB)
    fusion(nomA, nomB, longueur, nomFic, nbMonotonies)
    longueur ← longueur * 2
  Jusqu'à ce que (nbMonotonies = 1)
Fin tri_par_fusion
```

b. Donnez le code de la procédure d'éclatement en langage C++.

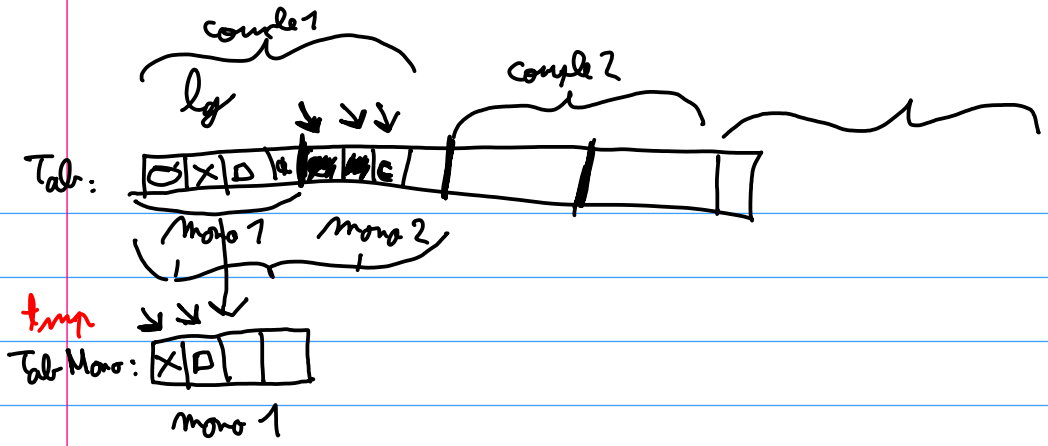
```
void eclatement (const string & nomFicX, int lg,
```

```
                    const string & nomFicA, const string & nomFicB) {
  bool mettreDansFicA = true;
  int i = 0;
  double element;

  ifstream ficX(nomFicX.c_str());
  if (!ficX.is_open()) {
     cout << "Erreur dans l'ouverture en lecture du fichier : " << nomFicX << endl;
     exit(EXIT_FAILURE);
  }

  ofstream ficA(nomFicA.c_str());
  if (!ficA.is_open()) {
     cout << "Erreur dans l'ouverture en ecriture du fichier : " << nomFicA << endl;
     exit(EXIT_FAILURE);
  }

  ofstream ficB(nomFicB.c_str());
  if (!ficB.is_open()) {
     cout << "Erreur dans l'ouverture en ecriture du fichier : " << nomFicB << endl;
     exit(EXIT_FAILURE);
  }

  // On boucle tant qu'il y a quelque chose à lire
  while (!ficX.eof()) {
     [illegible]

     if (mettreDansFicA) ficA << element << " ";
     else ficB << element << " ";

     i++;
     if (i == lg) {
        mettreDansFicA = ! mettreDansFicA;
        i = 0;
     }
  }

  ficX.close();
  ficA.close();
  ficB.close();
}
```

Exercice 2 : Tri fusion interne

Ecrire une version itérative (c'est-à-dire non récursive) de l'algorithme du tri fusion d'un tableau de réels. Cette version du tri fusion n'utilisera pas de fichiers, on pourra utiliser de la mémoire sur le tas pour stocker des (sous-) tableaux.

Procédure tri_par_fusion (tab: tableau [1...n] de réels)
Précondition : tab[1], tab[2], … tab[n] initialisés
Postcondition : tab[1] ≤ tab[2] ≤ … ≤ tab[n]
Paramètres en mode donnée : aucun
Paramètre en mode donnée-résultat : tab
Variables locales :
  i, j, k : entiers
  lgMono, nbMono1traites, nbMono2traites : entiers
  debutMono1, debutMono2 : entiers
  tmp: pointeur sur réel
Début
  lgMono ← 1


couple 1
lg
couple 2
Tab:
Mono 1
Mono 2
tmp
Tab Mono:
mono 1

```
Tant que (lgMono < n)
   tmp ← réserve tableau [1...lgMono] de réels
  debutMono1 ← 1
  debutMono2 ←  debutMono1 + lgMono

  Tant que (debutMono2 ≤ n) /* tant qu'on a une paire de monotonies à fusionner */
    {On recopie la premiere monotonie dans tmp. On est sûr qu'elle est complète, vue la condition de
    bouclage de ce Tant que}
    k ← debutMono1   {indice dans tab}
    i ←  1                {indice dans tmp}
    Tant que (k < debutMono1 + lgMono)
       tmp[i] ←  tab[k]
       k ←   k + 1
       i ←   i + 1
    Fin Tant que

    {On fusionne les deux monotonies (attention, la 2ème peut être incomplete)}
    i ←  1                {indice dans tmp, monotonie 1}
    j ←  debutMono2   {indice dans tab, monotonie 2}
    k ←  debutMono1   {indice dans tab, monotonie fusionnée}
    nbMono1traites  ←  0
    nbMono2traites  ← 0
    Tant que ((nbMono1traites < lgMono) et (nbMono2traites < lgMono) et (j ≤ n))
       {tant qu'aucune des deux monotonies n'est epuisée}
       Si (tmp[i] < tab[j]) Alors
         tab[k] ← tmp[i]
         nbMono1traites ← nbMono1traites + 1
         i ← i +1
       Sinon
         tab[k] ←  tab[j]
         nbMono2traites ← nbMono2traites + 1
         j ← j + 1
       Fin Si
      k ← k + 1
    Fin Tant que
    Si ((nbMono2traites = lgMono) ou (j > n)) Alors
       {On a épuisé la seconde monotonie, il reste à recopier la première}
       Tant que (nbMono1traites < lgMono)
          tab[k] ← tmp[i]
          nbMono1traites ← nbMono1traites + 1
          i ← i + 1
          k ← k +1
       Fin Tant que
    Sinon
      {Dans le cas où l'on est sorti du « Tant que » parce que la monotonie 1 est épuisée, il n'y a rien à faire,
      les éléments de la monotonie 2 sont déjà en place}
    Fin Si

    {On passe à la paire de monotonies suivante}
    debutMono1 ← debutMono1 + (2*lgMono);
    debutMono2 ← debutMono2 + (2*lgMono);
  Fin Tant que
```

```
    Libère tmp
    lgMono ← lgMono * 2 {pour la prochaine passe sur le tableau, les monotonies seront 2x plus longues}
  Fin Tant que

Fin tri_par_fusion
```