TD 8 : fichiers et images numériques

Exercice 1 (Fichiers)**.**
On considère la fonction `mystere` suivante, avec un exemple d'appel :

```
void mystere(string nomFichier) {
    ifstream fichier;
    fichier.open(nomFichier);
    string t;
    int n;
    while ( fichier >> t and fichier >> n ) {
        cout << t << " " << 2 * n << endl;
    }
    fichier.close();
}
```

```
    mystere("truc.txt");
```

Le fichier `truc.txt` contient les premières lignes suivantes :

```
Henry 4
Messi 2
Ronaldo 3
Xavi 2
Neymar 1
```

(1) Expliquer ce que fait cette fonction. Préciser le format que doit avoir le fichier. Choisir des noms plus informatifs pour cette fonction et ses variables, et écrire sa documentation.

Correction :

La fonction lit le fichier dont le nom est pris en paramètre. Ce fichier doit contenir une suite de chaînes de caractères (par exemple des noms) suivie chacune d'un entier (par exemple une note). Elle affiche chaque chaîne suivie de l'entier multiplié par 2.

```
/** affiche les notes contenues dans un fichier après les avoir
 *  multipliées par deux.
 *  Format: chaque ligne doit être de la forme "<nom> <note>"
 *  @param nomFichier le nom du fichier
 **/
void afficheNotes(string nomFichier) {
    ifstream fluxFichier;
    fluxFichier.open(nomFichier);
    string nom;
    int noteSur10;
    while ( fluxFichier >> nom and fluxFichier >> noteSur10) {
        cout << nom << " " << 2 * noteSur10 << endl;
    }
    fluxFichier.close();
}
```

(2) Changer la fonction pour que chaque ligne de la sortie soit de la forme :

```
Nom: Alfred, note sur 10: 7, note sur 20: 14
```

Correction : Il suffit de remplacer l'instruction de la boucle while par :

```
cout << "Nom: " << nom << ", note sur 10 : " <<   noteSur10
                         << ", note sur 20 : " << 2 * noteSur10 << endl;
```

Exercice 2 (Fichiers)**.**

(1) Implanter les deux fonctions suivantes.

```
/** calcule la moyenne des notes contenues dans un fichier
 *  Format: chaque ligne est de la forme "<nom> <note>"
 *  @param nomFichier le nom du fichier
 *  @return la moyenne des notes
 **/
float moyenne(string nomFichier);
```

```
/** lit les notes contenues dans un fichier et en fait un tableau
 *  Format: chaque ligne est de la forme "<nom> <note>"
 *  @param nomFichier le nom du fichier
 *  @return un tableau contenant les notes
 **/
vector<int> lit_notes(string nomFichier);
```

Correction :

```
float moyenne(string nomFichier) {
    ifstream varFichier;
    varFichier.open(nomFichier);
    string temp;
    int note;
    int somme = 0;
    int nb = 0;
    while ( varFichier >> temp and varFichier >> note ) {
        somme = somme + note;
        nb = nb + 1;
    }
    varFichier.close();
    return somme / nb;
}

vector<int> lit_notes(string nomFichier) {
    ifstream fichier;
    fichier.open(nomFichier);
    vector<int> notes;
    string temp;
    int note;
    while ( fichier >> temp and fichier >> note ) {
        notes.push_back(note);
    }
    fichier.close();
    return notes;
}
```

(2) Lorsque cela est possible, écrire un test.

Correction : Après avoir écrit un fichier de test, on peut faire des tests automatiques :

```
CHECK( moyenne("truc.txt") == 3 );
CHECK( lit_notes("truc.txt") == vector<int>( {4,2,3,2,1,5,5,4,4,5} ) );
```

(3) ♣ Comment pourrait-on tester la fonction du premier exercice ?

Correction : Cette fonction est plus difficile à tester automatiquement car elle fait des affichages. Il faudrait rediriger le flux de sortie standard et l'analyser. Autre possibilité : on pourrait aussi construire une chaîne de caractères (dans la fonction) et la comparer à la sortie souhaitée.

Exercice 3 (Images numériques)**.**
Pour manipuler informatiquement une image analogique (continue), on doit la numériser, c'est à dire l'encoder sous forme d'une image numérique (discrète). Il en existe deux grandes catégories :

— Les images vectorielles : une image y est encodée par une combinaison de primitives géométriques (lignes, disques, ...) auxquelles sont appliquées des transformations.
En savoir plus : `https://fr.wikipedia.org/wiki/Image_vectorielle`

— Les images matricielles, ou « carte de points » (de l'anglais bitmap) : une image y est constituée d'une matrice – ou tableau ou grille – où chaque case – appellée pixel – possède une couleur qui lui est propre. Il s'agit donc d'une juxtaposition de carrés de couleur formant, dans leur ensemble, une image.

FIGURE 1. Un exemple d'image matricielle

En savoir plus : `https://fr.wikipedia.org/wiki/Image_matricielle`

Le fichier suivant contient une image noir et blanc au format PBM (*Portable Bit Map*). Deviner comment fonctionne ce format de fichier et dessiner l'image correspondante.

```
P1
# CREATOR: GIMP PNM Filter Version 1.1
10 10
0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0
1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1
1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0
0 0 0 1 1 1 1 0 0 0
```

Indications : La première ligne précise le format du fichier (texte, image noir et blanc) ; la deuxième est un commentaire ; tout le reste a un rôle !

Voir `http://fr.wikipedia.org/wiki/Portable_bitmap` pour plus d'informations sur ce format de fichier.

Correction : La ligne `10 10` indique qu'il s'agit d'une image avec 10 lignes de 10 pixels. La suite donne les pixels de l'image (0 pour blanc, 1 pour noir), en la parcourant de haut en bas et de gauche à droite (comme la lecture d'un texte en français). Cela donne : ☺

Exercice 4 (Images numériques couleur)**.**
Le fichier suivant contient une image en couleur au format PPM (*Portable Pix Map*). Deviner comment fonctionne ce format de fichier et dessiner l'image correspondante.

```
P3
# CREATOR: GIMP PNM Filter Version 1.1
3 2
255
0 255 0 255 255 255
```

```
255 0 0 0 255 0 255
255 255 255 0 0
```

Indication : Quelles sont les trois couleurs primaires usuelles (pour les écrans) ?

Correction : La ligne `P3` précise le format du fichier (texte, image couleur RGB). La ligne `3 2` indique qu'il s'agit d'une image avec 2 lignes de 3 pixels. La ligne `255` donne la valeur de l'intensité maximale. Ensuite chaque pixel est codé par trois valeurs (RGB = rouge, vert, bleu) comprises entre 0 et l'intensité maximale. Cela donne : 

Exercice ♣ 5.
Implanter les fonctions suivantes :

```
/** compte le nombre de mots d'un fichier
 *  @param nomFichier le nom du fichier
 *  @return le nombre de mots contenus dans le fichier
 **/
int word_count(string nomFichier);
```

```
/** compte le nombre de lignes d'un fichier
 *  @param nomFichier le nom du fichier
 *  @return le nombre de lignes contenues dans le fichier
 **/
int line_count(string nomFichier);
```

```
/** affiche (sur la sortie standard) le contenu d'un fichier
 *  @param nomFichier le nom du fichier
 **/
void cat(string nomFichier);
```

```
/** copie le contenu d'un fichier dans un autre
 *  @param source le nom du fichier source
 *  @param destination le nom du fichier destination
 **/
void copy(string source, string destination);
```

Indication : la bibliothèque standard fournit la fonction suivante :

```
/** lit une ligne d'un flux et la stocke dans la chaîne de caractères s
 * @param f un flux entrant
 * @param s une chaîne de caractères
 * @return le flux entrant
 **/
istream getline(istream &f, string &s);
```

Correction :

```
int word_count(string nomFichier) {
    ifstream fichier;
    fichier.open(nomFichier);
    int nbr = 0;
    string temp;
    while ( fichier >> temp ) {
        nbr = nbr + 1;
    }
    fichier.close();
    return nbr;
}

int line_count(string nomFichier) {
    ifstream fichier;
    fichier.open(nomFichier);
    string ligne;
    int compteur = 0;
    while ( getline(fichier, ligne) ) {
```

```
        compteur = compteur + 1;
    }
    fichier.close();
    return compteur;
}

void cat(string nomFichier) {
    ifstream fichier;
    fichier.open(nomFichier);
    string ligne;
    while ( getline(fichier,ligne) ) {
        cout << ligne << endl;
    }
    fichier.close();
}

void copy(string source, string destination) {
    ifstream fichierS;
    fichierS.open(source);
    ofstream fichierD;
    fichierD.open(destination);
    string ligne;
    while ( getline(fichierS, ligne) ) {
        fichierD << ligne << endl;
    }
    fichierS.close();
    fichierD.close();
}
```

Exercice ♣ 6.
Deviner ce que fait la fonction `mystereEpais` suivante et écrire un test :

```
int mystereEpais(string zut) {
    ifstream bla;
    bla.open(zut);
    int foo = 0;
    char y;
    while ( bla >> y ) {
        foo++;
    }
    bla.close();
    return foo;
}
```

Correction : La fonction renvoie le nombre de caractères du fichier en entrée (sans compter les espaces et retours à la ligne : ils sont considérés comme de simples délimiteurs).
Exemple de test :

```
    CHECK( mystereEpais("truc.txt") == 65 );
```

Exercice ♣ 7.
On dispose de trois fichiers texte nommés `etud1.txt`, `etud2.txt`, et `etud3.txt`, dans lesquels chaque ligne contient : le nom d'un-e étudiant-e, un espace, son groupe, un espace, une note.

(1) Fusionner ces trois fichiers en un seul fichier pour que le fichier final contienne cinq colonnes (séparées par des espaces) contenant : nom groupe note1 note2 note3.
On vérifiera que les noms des étudiants sont dans le même ordre dans les trois fichiers, et qu'il n'y a pas de nom manquant dans certains fichiers.

(2) Créer un fichier par groupe, dans lequel on écrira le nom de chaque étudiant-e, suivi d'un espace, suivi de sa note moyenne (obtenue en faisant la moyenne des trois notes figurant dans le fichier précédent).
Chaque fichier sera enregistré sous un nom au format suivant :
`groupeB5.txt`, `groupeA1.txt`, `groupeC3.txt`.
Ce programme doit pouvoir fonctionner quel que soit le nombre de groupes et quels que soient les noms de ces groupes.

Correction :

```
#include <fstream>
#include <iostream>
#include <vector>
using namespace std;

vector<string> fic_notes = {"etud1.txt", "etud2.txt", "etud3.txt"};
int nb_notes = fic_notes.size();
int k;

string nom, gpe, nom_fusion ;
double note, somme ;

//typedef vector<double> TabReels ;
vector<double> notes(nb_notes);

// la taille de chaque fichier de notes
//typedef vector<int> TabInt;
vector<int> taille_fichier(nb_notes);

int main(){

for ( int i=0; i<nb_notes; i++ ){
    ifstream f;
    f.open(fic_notes[i]);
    taille_fichier[i]=0;
    // on lit chaque ligne : nom gpe note
    while (f >> nom >> gpe >>note) {
        taille_fichier[i]++;
    }
    f.close();
}
//si au moins un fichier n'est pas de même taille que les autres, on s'arrête
bool meme_taille = true;
k=0;
while (k < nb_notes-1 and meme_taille){
```

```
    meme_taille = (taille_fichier[k]==taille_fichier[k+1]);
    k++;
}

if (not(meme_taille)){
    cout << "les fichiers de notes ne sont pas de même taille" << endl;
}
else {
    // on recopie nom, groupe, note1 dans le fichier fusionné
    ifstream note1_lire;
    note1_lire.open(fic_notes[0]);
    ofstream fusion_ecrire;
    fusion_ecrire.open("etud_fusion.txt");
    while (note1_lire >> nom >> gpe >>note) {
        fusion_ecrire << nom << ' '<< gpe <<' '<<note << endl;
    }
    note1_lire.close();
    fusion_ecrire.close();

    // on recopie les notes suivantes dans le fichier fusionné
    // en utilisant un fichier temporaire à chaque étape pour faire la recopie.
    // on vérifie au passage que les noms sont dans le même ordre.
    // NB : on aurait pu vérifier aussi les noms des groupes
    bool erreur_nom = false ;
    k = 1;
    while (not(erreur_nom) and k < nb_notes){
        ifstream fusion_lire;
        fusion_lire.open("etud_fusion.txt");
        ofstream temp_ecrire;
        temp_ecrire.open("temp.txt");
        ifstream notek_lire;
        notek_lire.open(fic_notes[k]);

        while (not(erreur_nom) and notek_lire>>nom>>gpe and fusion_lire>>nom_fusion
                erreur_nom = (nom  != nom_fusion);
                if (erreur_nom) {
                cout << "pas les mêmes noms" << endl ;
            }
            else {
                temp_ecrire << nom << ' ' << gpe ;
                for (int i=0;i<k; i++){
                            if (fusion_lire >>notes[i]){
                                temp_ecrire << ' ' << notes[i];
                            }
                        }
                        if (notek_lire >> notes[k]){
                            temp_ecrire << ' ' << notes[k] << endl;
                        }

            }
        }
        temp_ecrire.close();
        fusion_lire.close();
        notek_lire.close();
```

```
    // on recopie le fichier temporaire dans le fichier fusionné
    ifstream temp_lire;
    temp_lire.open("temp.txt");
    ofstream fusion_ecrire;
    fusion_ecrire.open("etud_fusion.txt");
    while(temp_lire >> nom >> gpe ){
        fusion_ecrire << nom << ' ' << gpe ;
        for (int i=0;i<=k; i++){
                        if (temp_lire >>notes[i]){
                            fusion_ecrire << ' ' << notes[i];
                    }
            }
            fusion_ecrire << endl ;

    }
    temp_lire.close();
    fusion_ecrire.close();
    k++;
}

if (not(erreur_nom)) {
    // on détermine la liste des groupes
    //vector<string> groupes(taille_fichier[0]);
    //au maximum, il y aura un gpe par étudiant
    vector<string> groupes;
    ifstream f ;
    f.open("etud_fusion.txt");
    bool present ;
    int j ;
    while (f >> nom >> gpe){
            present = false ;
            j= 0;
            while (j<groupes.size() and not(present)){
            present = (gpe == groupes[j]);
            j++ ;
        }
        if (not(present)){
            groupes.push_back(gpe);
        }
        for (int i=0;i<nb_notes; i++){
                f >>notes[i];
        }
    }
    f.close();

    // on écrit un fichier pour chaque groupe
    for (int p=0 ; p< groupes.size(); p++){
        ofstream fgpe;
        fgpe.open("groupe"+groupes[p]+".txt");
        ifstream fusion;
        fusion.open("etud_fusion.txt");
        while (fusion >> nom >> gpe){
            for (int i=0;i<nb_notes; i++){
```

```
                    fusion >>notes[i];
                }

                if (gpe==groupes[p]){ // si c'est le même groupe
                    // on calcule la somme des nb_notes notes
                    somme = 0;
                    for (int n=0; n<nb_notes ; n++){
                        somme = somme + notes[n];
                    }
                    fgpe << nom << ' ' << somme/nb_notes << endl ;
                }
            }

            fgpe.close();
            fusion.close();
        }
    }
}
return 0;
}
```