

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.
- (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
```

Exercice 2 (Fichiers).

- (1) Implanter les deux fonctions suivantes.

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

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

- (2) Lorsque cela est possible, écrire un test.
- (3) ♣ Comment pourrait-on tester la fonction du premier exercice ?

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 – appelée 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.

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



FIGURE 1 – Un exemple d'image matricielle

- (1) **Images en noir et blanc.** Le fichier suivant contient une image en 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 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 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.

- (2) **Images en 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) ?

Exercice ♣ 4.

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 une chaîne de caractères
 * @param f un flux entrant
 * @param s une chaîne de caractères
 * @return le flux entrant
 */
istream getline(istream &f, string &s);
```

Exercice ♣ 5.

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;
}
```

Exercice ♣ 6.

On dispose de trois fichiers texte nommés `note1.txt`, `note2.txt`, et `note3.txt`. Dans chacun d'eux, chaque ligne contient trois informations, séparées par des espaces : le nom d'un-e étudiant-e, son groupe, une note. Les trois fichiers pourraient par exemple correspondre aux notes dans trois matières pour une même promotion.

- (1) Fusionner ces trois fichiers en un seul fichier `notes.txt` dont chaque ligne aura cinq informations séparées par des espaces : `<nom> <groupe> <note1> <note2> <note3>`.
On vérifiera au passage que les trois fichiers de départ contiennent exactement les mêmes noms d'étudiant-es et dans le même ordre.
- (2) Créer un fichier par groupe, qui indiquera, pour chaque étudiant-e de ce groupe, sa note moyenne (moyenne des trois notes), au format `<nom> <note_moyenne>`.
Les fichiers seront nommés `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.