TP de Système n° 2 : Lecture/Écriture dans un fichier

Lecture et écriture dans un fichier

Exercice 1 : Remplissage d'un tableau de structures à partir d'un fichier

1. Définir le type `pixel` contenant trois champs correspondant aux intensités de rouge, vert et bleu d'un pixel. On considère que chaque intensité est codée sur un octet.
2. Écrire la fonction `lecture_pixel` qui affecte à un pixel de type `pixel` ses trois valeurs de type `unsigned char` lues dans un fichier à partir de la position courante de son curseur.
3. Écrire la fonction `main` qui remplit un tableau de `pixel` à partir d'un fichier contenant sur la première ligne, la `longueur` du tableau et sur la deuxième ligne, une chaîne de caractères de longueur 3 `longueur`. Le nom du fichier sera passé en argument du `main`.

Pour cet exercice, vous utiliserez les fonctions suivantes de la bibliothèque standard :

```
size_t fread(void *ptr, // adresse recevant la copie
size_t size,           // taille d'un élément en octets
size_t nmemb,          // nombre d'éléments
FILE *stream);         // adresse du flux

int fscanf(FILE *stream,// adresse du flux
const char *format,    // format à suivre
...);                  // adresse(s) recevant la(les) copie(s)

FILE *fopen(const char *path,
const char *mode);     // "r" pour lire et "w" pour écrire

int fclose(FILE *fp);
```

Exercice 2 : Écriture dans un fichier à partir d'un tableau de structures

Écrire la fonction `ecriture_pixel` qui écrit un pixel de type `pixel` dans un fichier. Vous utiliserez la fonction :

```
size_t fwrite(const void *ptr, size_t size,
size_t nmemb, FILE *stream);
```

Exercice 3 : Le type `image`

Une image est représentée sous la forme d'une structure contenant le nombre de lignes, le nombre de colonnes et un tableau « bidimensionnel - plat » de pixels.

1. Définir le type `image`.
2. Écrire la fonction `allouer_image` en fonction d'un nombre de lignes et d'un nombre de colonnes.
3. Écrire la fonction `copier_image` qui retourne une copie d'une image.

Manipulation des images

Le but est d'écrire un programme qui permet de manipuler des images au format `ppm P6` . Les noms des fichiers contenant de telles images ont un suffixe `ppm` ; ces fichiers peuvent être visualisés à l'aide de la commande `display`.

Voici un exemple de fichier `ppm` , pour comprendre comment ce format fonctionne :

```
P6                  ! format du fichier
2 4                 ! largeur et hauteur (nombre de pixels)
15                  ! profondeur maximale des couleurs
cccccc              || spécification des couleurs :
cccccc              || 3 caractères par pixel, correspondant
cccccc              || à une combinaison de rouge, vert et bleu
cccccc              || (sans retour à la ligne)
```

Dans ce format, chaque caractère définissant une couleur est codé sur un octet si la profondeur des couleurs est inférieure à 256, sur deux octets sinon. En outre, les caractères ne sont séparés par aucun caractère d'espacement.

Dans la suite du TP, on fait l'hypothèse que la profondeur maximale de couleur de tous les fichiers manipulés est inférieure à 256 : chaque couleur est donc codée sur un octet.

Un fichier d'en-tête vous est fourni : `image.h`. Vous y trouverez des définitions de macros et de types, ainsi que les spécifications des fonctions à écrire. Vous trouverez également des fichiers `ppm P6` pour tester votre programme.

Déplacement dans le fichier `ppm` Chaque pixel correspond à trois octets. Comme il n'y a pas de caractère d'espacement entre les couleurs, la description des pixels ressemble à un tableau « bidimensionnel - plat ». Le pixel de coordonnées `(i,j)` se situe donc à l'octet `offset+(i*largeur+j)*RGB`, où `offset` est la position du premier pixel dans le fichier `ppm` et `RGB` le nombre de couleurs par pixel (`RGB=3`).

Vous trouverez dans les pages du manuel la description d'outils utiles pour faire ce TP, notament les fonctions `fread`, `fscanf`, `fwrite`.

Exercice 4 : Conversion entre `ppm` et `image`

1. Programmer la fonction `ppm_to_image`.
2. Ajouter la fonction inverse `image_to_ppm`.
3. Écrire un `main` pour tester ces fonctions, en particulier s'assurer que l'identité est conservée (*i.e.* qu'on retrouve bien le même fichier de départ lorqu'on passe par la conversion en *image*). Faire en sorte que le programme prenne le nom du fichier `ppm` en argument sur la ligne de commande.

Exercice 5 :

Écrire les fonctions de :
- miroir horizontal (par rapport à l'axe des abscisses) d'une image,
- rotation de 90° (on testera aussi avec des images non carrées),
- symétrie centrale par blocs de 32 32 pixels. L'image est découpée en blocs, et les blocs symétriques par rapport au centre de l'image sont échangés. Découvrir l'image `image_mystere.ppm` (la fonction est son propre inverse).