
SÉANCE 6




Objectif

Le but de cette sixième séance est de permettre à ceux qui ont du retard de le rattraper et de poursuivre l'utilisation du mécanisme de passage des paramètres, et aux autres d'aller un peu plus loin.

Nous vous rappelons que du travail personnel, en dehors des séances, est indispensable.

Consignes pour cette séance :

- Pour ceux qui n'ont pas terminé la séance 5 : terminez-la et n'oubliez pas d'appeler l'enseignant-e en cas de difficulté ; passez ensuite à l'exercice 1 ci-dessous.
- Pour ceux qui ont terminé la séance 5 : traitez l'exercice 1 ci-dessous et passez à la rubrique .



Exercices

Exercice 1 (Conversion de durées)

L'objectif de cet exercice consiste à écrire deux fonctions permettant de convertir une durée exprimée en secondes en la même durée exprimée en heures, minutes et secondes. Par exemple, 3666 secondes représentent 1 heure 1 minute et 6 secondes.

Pour tester vos deux fonctions, écrivez une fonction principale qui demande à l'utilisateur de taper une durée en seconde, la lit, appelle chacune des deux fonctions et affiche les résultats. Aucune entrée au clavier ni sortie à l'écran ne sera donc réalisée par les fonctions de conversion.

Les informations en entrée et en sortie des deux fonctions doivent passer par leurs paramètres selon le schéma suivant :

1. la première fonction doit prendre en entrée la durée en secondes et délivrer en sortie le nombre d'heures, le nombre de minutes et le nombre de secondes (4 paramètres en tout) ;
2. la seconde fonction doit prendre en entrée la durée en secondes et délivrer en sortie le résultat sous la forme d'un tableau à 3 cases, la première contenant le nombre d'heures, la deuxième le nombre de minutes et la troisième le nombre de secondes (2 paramètres en tout).



Pour aller plus loin

Exercice 2 (Manipulation d'images de niveaux de gris)

L'objectif de cet exercice est de poursuivre la manipulation d'images de niveaux de gris commencée lors de la séance 4 où vous avez réalisé l'application d'une transformation géométrique.

Il s'agit ici d'implémenter un autre type de transformation d'image : les transformations ponctuelles. Le résultat d'une telle transformation est une autre image dont le niveau de gris de chaque pixel est fonction du niveau de gris du pixel situé au même endroit dans l'image initiale. Les niveaux de gris étant des entiers compris entre 0 et 255, cette transformation peut être complètement décrite par une « table de transcodage » que vous implémenterez par un tableau de 256 `unsigned char`. L'indice de chaque case de cette table indique l'ancien niveau de gris et son contenu indique le nouveau niveau de gris. Par exemple, si la case d'indice 112 contient la valeur 184, cela signifie que tous les pixels dont le niveau de gris dans l'image initiale était égal à 112 vont avoir le niveau de gris 184 dans l'image finale.

Pour obtenir l'image résultat, il suffit donc de parcourir l'image initiale et, pour chaque pixel, mettre dans l'image finale le niveau de gris contenu dans la case de la table de transcodage dont l'indice est égal au niveau de gris de l'image initiale.

L'effet visuel obtenu dépendra des valeurs contenues dans la table de transcodage. Par exemple, si elle est représentée par le tableau `Table` et si `Table[0]==255`, `Table[1]==254`, `Table[2]==253`, ..., `Table[254]==1`, `Table[255]==0`, c'est-à-dire si `Table[g]==255-g`, alors l'image finale sera le « négatif » (au sens de la photographie) de l'image initiale.

On suppose que les images auront au plus 512 lignes et 512 colonnes. Définissez deux constantes symboliques `LMAX` et `CMAX` avec ces valeurs. Si vous utilisez votre programme sur des images plus grandes, il suffira de modifier les valeurs de ces deux constantes symboliques.

Pour tester chacune des fonctions demandées ci-dessous, écrivez une fonction principale que vous ferez évoluer au fur et à mesure. Pour lancer l'exécution de votre programme, reportez-vous au sujet de la séance 4.

Remarque : dans les en-têtes des fonctions ci-dessous, le nombre de lignes des tableaux à deux dimensions et le nombre d'éléments des tableaux à une dimension sont précisées. Ce n'est pas obligatoire car c'est inutile pour le compilateur (cf. les explications données en cours). Mais cela permet de vous rappeler la taille des tableaux que vous allez passer en paramètres lors de l'appel de ces fonctions dans le cas particulier de cet exercice.

1. En vous aidant de ce que vous avez écrit lors de la séance 4, écrivez la fonction d'en-tête :

```
void LireImage(unsigned char Im[LMAX][CMAX], int *pNbLignes, int *pNbColonnes)
```

 qui lit le contenu d'un fichier au format PGM-ASCII tapé au clavier par l'utilisateur, stocke les niveaux de gris des pixels dans le tableau `Im` et délivre en sortie le nombre de lignes et le nombre de colonnes de l'image.
2. En vous aidant de ce que vous avez écrit lors de la séance 4, écrivez la fonction d'en-tête :

```
void AfficherImage(unsigned char Im[LMAX][CMAX], int NbLignes, int NbColonnes)
```

 qui affiche à l'écran le contenu du fichier au format PGM-ASCII correspondant à l'image dont les niveaux de gris sont dans le tableau `Im`. Mettez systématiquement 255 comme valeur du plus grand niveau de gris de l'image.
3. Écrivez la fonction d'en-tête :

```
void AppliquerTable(unsigned char Table[256],
                    unsigned char ImSource[LMAX][CMAX],
                    int NbLignes,
                    int NbColonnes,
                    unsigned char ImDestination[LMAX][CMAX])
```

 qui, à partir d'une image de niveaux de gris `ImSource` de taille `NbLignes` × `NbColonnes`, crée l'image de niveaux de gris `ImDestination` de la même taille, en appliquant la transformation ponctuelle décrite par la table de transcodage `Table`.
4. Écrivez la fonction d'en-tête :

```
void RemplirTableInversion(unsigned char Table[256])
```

 qui crée la table de transcodage correspondant à la transformation d'inversion des niveaux de gris (obtention du négatif) :

$$F(g) = 255 - g \quad (1)$$

où g désigne l'ancien niveau de gris et $F(g)$ le nouveau niveau de gris.

inversion.pdf

5. Testez votre programme sur l'image `chien.pgm` utilisée lors de la séance 4.

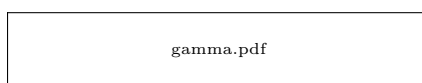
6. Écrivez la fonction d'en-tête :

```
void RemplirTableGamma(double Gamma,unsigned char Table[256])
```

qui crée la table de transcodage correspondant à la transformation appelé « correction gamma » :

$$F(g) = \left\lceil 255 \left(\frac{g}{255} \right)^\gamma \right\rceil \quad (2)$$

où $[x]$ désigne l'entier le plus proche de x . Si $\gamma = 1$ alors la transformation sera sans effet. Si $\gamma < 1$ alors l'image finale sera plus claire que l'image initiale (généralement, on choisit $\gamma \in [0.4, 0.5]$). Si $\gamma > 1$ alors l'image finale sera plus sombre que l'image initiale (généralement, on choisit $\gamma \in [2, 2.5]$).



La fonction double `round(double x)` retourne l'entier le plus proche de x .

La fonction double `pow(double x, double y)` retourne x^y .

Ces deux fonctions font partie de la bibliothèque standard mathématique. Elles nécessitent donc l'inclusion de `math.h` et l'ajout de `-lm` à la commande de compilation.

7. Testez votre programme sur l'image `paysage.pgm` avec plusieurs valeurs pour γ , par exemple 0.4 et 2.5.



Pour aller encore plus loin

✎ Exercice 3 (Manipulation d'images en couleur)

Pour aller encore plus loin, vous pouvez manipuler des images en couleur. Une des manières de représenter la couleur de chaque pixel avec trois composantes consiste à donner son niveau de rouge, son niveau de vert et son niveau de bleu (espace RVB ou RGB), chacun de ces niveaux étant un entier compris entre 0 et 255.

Il existe de nombreux formats de fichiers contenant des images en couleur. Le format « PPM-ASCII » est l'extension à la couleur du format « PGM-ASCII ». Il s'agit de fichiers de texte contenant les mêmes informations (cf. le sujet de la séance 4), avec les exceptions suivantes :

- la signature du format PPM-ASCII est constituée des deux caractères `P3` ;
- à la place du plus grand niveau de gris de l'image, se trouve la valeur du plus grand niveau parmi les trois composantes de tous les pixels ;
- à la place du niveau de gris de chaque pixel se trouvent le niveau de rouge suivi du niveau de vert suivi du niveau de bleu séparés par un ou plusieurs séparateurs.

Par exemple, le fichier `imagette.ppm` :

```
P3
4 5
255
0 0 0 0 0 0 0 0 0 255 0 255
0 0 0 0 255 127 0 0 0 0 0 0
0 0 0 0 0 0 0 255 127 0 0 0
255 0 255 0 0 0 0 0 0 0 0 0
255 0 0 0 255 0 0 0 255 255 255 255
```

contient une image couleur composée de 5 lignes et 4 colonnes dont le pixel situé à la fin de la première ligne a un niveau de rouge égal à 255, un niveau de vert égal à 0 et un niveau de bleu égal à 255 (couleur magenta).

Une manière de représenter les couleurs des pixels d'une image consiste à utiliser trois tableaux à deux dimensions, chacun contenant les niveaux d'une composante.

Dans un premier temps, vous pouvez créer des versions étendues à la couleur des fonctions `LireImage` et `AfficherImage` :

```
void LireImageCouleur(unsigned char Rouge[LMAX][CMAX],
                     unsigned char Vert[LMAX][CMAX],
                     unsigned char Bleu[LMAX][CMAX],
                     int *pNbLignes, int *pNbColonnes)
```

```
void AfficherImageCouleur(unsigned char Rouge[LMAX][CMAX],
                         unsigned char Vert[LMAX][CMAX],
                         unsigned char Bleu[LMAX][CMAX],
                         int NbLignes, int NbColonnes)
```

Ensuite, vous pouvez manipuler les composantes, par exemple en mettant à 0 tous les niveaux de l'une des composantes. Vous pouvez échanger deux composantes. Vous pouvez appliquer une transformation ponctuelle (la correction gamma ou l'inversion) à une, à deux ou aux trois composantes d'une image. Vous pouvez laisser libre cours à votre imagination.

Vous pouvez tester votre programme sur l'image `chenille.ppm`.