U.E. S2IN324 - CCTP - Devoir

Indications : *Ce devoir est à réaliser en binôme (ou individuellement) et à rendre exclusivement par dépôt sur le site Moodle du cours en plusieurs parties :*
La partie 1.1 Représentation - Chargement - Sauvegarde (exercice 1, 2, 3, 4) est à déposer ***avant le 21 novembre 23h55***
La partie 1.2 Manipulation d'une image en mémoire (exercice 5, 6, 7, 8) est à déposer ***avant le 5 décembre 23h55***
Le devoir complet avec la partie 1.3 Sténographie et le programme principale (exercices 9 à 16) est à déposer ***avant le 19 décembre 23h55*** .
Vous indiquerez en commentaire au début de chacun des fichiers de code : numéros étudiant, noms et prénoms des protagoniste.

Travail à réaliser

Il s'agit de fournir un module `manipimage` (fichier en-tête et fichier source) permettant de manipuler des images au travers des quelques opérations demandées dans l'énoncé. Vous devrez aussi coder un programme principal qui proposera d'utiliser les différentes opérations du module.

Votre devoir sera donc constituer de 3 fichiers : `manipimage.h`, `manipimage.c` et `main.c`.

Vous déposerez sur Moodle une archive de l'ensemble des fichiers de votre devoir. Si vous le faites à deux, vous ne devez rendre qu'un seul devoir déposé sur un seul de vos compte Moodle.

Introduction au traitement d'image

Ce devoir propose d'aborder le traitement d'image à travers quelque opérations à effectuer sur une image en utilisant une représentation en mémoire sous forme de tableau à deux dimensions dont les éléments représentent les pixels d'une image.

La première partie concerne le codage d'un module `manipimage` qui doit contenir :

— La définition des types `tPixel` et `tImage` utilisés pour représenter une image en mémoire.
— Les opérations de base permettant de charger une image en mémoire dans une structure `tImage` depuis un fichier au format `ppm`, de sauver une image dans un fichier au bon format et de réaliser la copie d'une `tImage`.
— Quelques opérations simples de traitement d'image : floutage, détourage, transformation en niveaux de gris.
— Des opérations d'introduction à la stéganographie qui est l'art de dissimuler des données dans d'autres données. En l'occurrence il s'agira de dissimuler une image dans une autre et de dissimuler un texte dans une image.

Dans la deuxième partie vous coderez un programme principal utilisant le module `manipimage` et permettant de tester toutes les opérations du module.

Les types d'images numériques

Il existe trois types d'images numériques : les images noir et blanc, les images en niveau de gris et les images couleurs.

Une image numérique et un tableau rectangulaire dont les éléments sont des pixels. Selon le type d'image un pixel est codé :

— pour les images en noir et blanc, sur 1 bit : 0 pour noir et 1 pour blanc
— pour les image en niveau de gris, sur 8 bits, soit une entier compris entre 0 (noir) et 255 (blanc)
— pour les image en couleur, par un triplet d'entiers codant les trois couleurs rouge, vert et bleu. Le niveau de chaque couleur est codée par un entier généralement compris entre 0 (absence de la couleur) et 255 (niveau maximum).

Le format `pnm`

Le format de fichier d'image `pnm` permet de représenter les trois types d'images. Il se décline en trois nominations selon la nature de ces images :

— les fichiers `.pbm` de type P1 pour les images en noir et blanc ;
— les fichiers `.pgm` de type P2 (format ascii dans fichier texte) ou P5 (format raw dans fichier binaire) pour les images en niveau de gris ;

— les fichiers `.ppm` de type P3 (ascii) ou P6 (raw) pour les images en couleur.
Dans ce devoir, vous manipulerez essentiellement des images `.ppm` de type P3.

Structure d'un fichier `.ppm` ascii

— la première ligne du fichier contient le type de l'image : P3 ou P6
— suivent éventuellement quelques lignes de commentaires qui débutent par le caractère `#`
— la première ligne après les commentaires contient la largeur et la hauteur (dans cet ordre) de l'image en pixels
— la ligne suivante contient la valeur maximale que peut prendre le niveaux de rouge, vert et bleu (généralement 255)
— les lignes suivantes représentent la liste des pixels décrivant l'image, ligne par ligne, depuis le premier pixel situé en haut à gauche jusqu'au dernier pixel situé en bas à droite.

Par exemple, voici le début du fichier `.ppm` d'une image en couleur codée en ascii, visualisé avec un éditeur de texte. Cet extrait contient le type de l'image, une ligne de commentaire indiquant le logiciel ayant servi à créer l'image, une ligne contenant la largeur et la hauteur en nombre de pixels. La ligne suivante contient la valeur maximale des niveaux : 255. Les 6 entiers qui suivent représentent les niveaux de (rouge, vert, bleu) des deux premiers pixels de l'image : (105, 145, 204) et (113, 150, 227)

```
P3
# CREATOR: GIMP PNM Filter Version 1.1
1600 1200
255
105
145
204
113
150
227
...
```

1 Le module `manipimage`

1.1 Représentation - Chargement - Sauvegarde

On donne ci-dessous (ainsi que dans le fichier `pourDevoirC.txt` qui accompagne cet énoncé) la définition des types que vous utiliserez pour représenter une image couleur en mémoire. Ces définitions sont à recopier au début dans le fichier en-tête du module `manipimage`..

```
// ************************************************
// Définition des types

// Le type tPixel pour représenter les niveaux de (rouge, vert, bleu)
typedef struct pixel {
    int r;
    int v;
    int b;
} tPixel;

// Le type tImage pour représenter une image dans un tableau de pixels
typedef struct image {
    int hauteur;        // Hauteur en pixels
    int largeur;        // Largeur en pixels
    char type[3];       // Type de l'image pnm P1 P2 P3 P4 ou P6
    int maxval;         // Valeur maximale de l'intensite d'une couleur
    tPixel** img;       // Le tableau des pixels
} tImage;
```

Le champ `tPixel** img` est un tableau rectangulaire de pixels, c'est à dire un tableau de tableaux de pixels. Le premier indice fait référence aux lignes de l'images et le second indice aux colonnes. Il y a `hauteur` lignes de `largeur` pixels. Ainsi `img[i][j]` contient le pixel situé en i^{eme} ligne et j^{eme} colonne.

Exercice 1 — Allocation de mémoire pour stocker une image

Ecrivez une fonction **tImage initImage(int haut, int larg, char typ[3], int vmax)** qui retourne une structure `tImage` dans laquelle les champs `hauteur` et `largeur` sont initialisés avec les valeurs données en paramètre et qui réalise l'allocation du tableau `img` pour `haut` lignes de `larg` pixels. Les champs `type` et `maxval` sont respectivement initialisés avec `typ` et `vmax`.

Exercice 2 — Copie d'une image

Ecrivez une fonction **tImage copieImage(tImage im)** qui réalise la copie de l'image donnée en paramètre dans une nouvelle image initialisée et allouée avec les mêmes caractéristiques que l'image donnée en paramètre et dans laquelle tous les pixels sont copiés. La fonction retourne la copie de l'image.

Exercice 3 — Chargement en mémoire d'une image depuis un fichier `.ppm`

Le chargement d'une image en mémoire consiste à initialiser et remplir une structure de type `tImage` à partir des informations contenu dans un fichier `.ppm`.
Ecrivez une fonction **tImage chargePpm(char* fichier)** qui charge dans une structure `tImage` l'image contenue dans le fichier dont le nom est donné en paramètre, sachant que ce fichier est au format .ppm ascii décrit plus haut.
Ce fichier est un fichier de type texte où chaque donnée est séparée de la suivante par un espace ou un passage à la ligne, vous pouvez utiliser la fonction `fscanf` pour récupérer les données. Attention aux lignes de commentaires qu'il faut ignorer.

Exercice 4 — Sauvegarde dans un fichier `.ppm`

Ecrivez une fonction **void sauvePpm(char* nom, tImage im)** qui créer un fichier `.ppm` dont le nom est donné en paramètre et dans lequel vous sauvez le contenu de la structure `tImage` donnée en paramètre en respectant le format d'un fichier `.ppm`.
Vous ajouterez dans ce fichier une ligne de commentaire contenant votre nom et prénom pour renseigner le créateur du fichier.
Vous pouvez utilisez `fprintf` pour écrire dans le fichier.

1.2 Manipulation d'une image en mémoire

Une fois chargée en mémoire les opérations qu'on peut effectuer sur une image consistent essentiellement à faire des calculs sur le tableau des pixels.

Niveaux de gris

Avec le format d'une image couleur où chaque pixel est représenté par les trois niveau r, v et b, la couleur grise est obtenue en affectant la même valeur au trois niveau de couleur.

Pour transformer une image couleur en image en niveau de gris il faut affecter à chaque pixel une niveau de gris, qu'on appelle la luminance, calculé à partir des niveaux de couleur r, v et b.

L'oeil humain n'étant pas sensible de la même manière à ces trois couleurs, pour obtenir un bon résultat on ne peut pas se contenter de faire la moyenne simple de ces trois valeurs.

La C.I.E (Commission Internationale de l'Éclairage) propose de caractériser l'information de luminance (la valeur de gris) d'un pixel par la formule :

Gris = 0.2125 Rouge + 0.7154 Vert + 0.0721 Bleu

Exercice 5

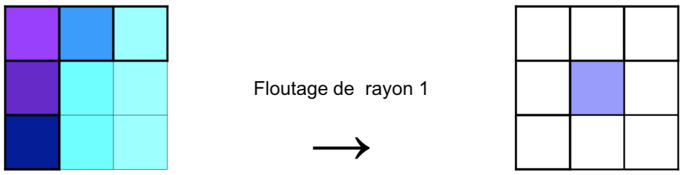
Ecrivez une fonction **tImage niveauGris(tImage im)** qui retourne une image en niveau de gris créée à partir de l'image donnée en paramètre. Utilisez la formule donner pour calculer la luminance de chacun des pixels.

Floutage

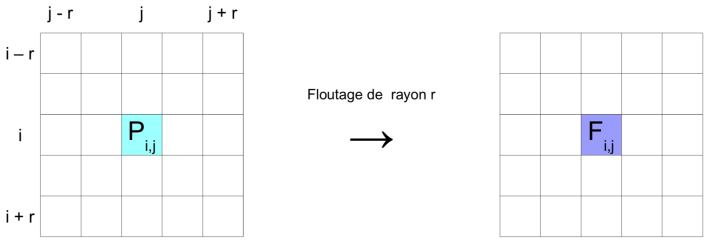
L'algorithme du flou gaussien consiste à remplacer chaque pixel d'une image par la valeur moyenne des pixels qui l'entourent. Comme cette opération modifie les valeurs des pixels et effectue les calculs à partir des valeurs des pixels environnants il est nécessaire de ne pas modifier l'image d'origine et de créer une nouvelle image.

Le degré de floutage dépend du nombre de pixels considérés autour du pixel à flouter. On exprimera cela par le rayon de floutage qui définit le carré des pixels situés autour du pixels à flouter à partir desquels on calcule la valeur moyenne.

Par exemple, dans la figure suivante le pixel bleu clair du milieu sera flouté en le remplaçant par la moyenne des 9 pixels situés dans un rayon de 1 pixel autour de lui pour obtenir le pixel de couleur parme.



De manière plus générale, étant donné un floutage de rayon r pixels on remplace un pixel $P_{i,j}$ par la moyenne des pixels situés dans un carré de côté $2r+1$ autour de $P_{i,j}$



Le pixels $p_{i,j}$ est remplacé par le pixel $F_{i,j} = \frac{1}{(2r+1)^2}\sum_{l=i-r}^{i+r}(\sum_{c=j-r}^{j+r} P_{l,c})$ (la moyenne des valeurs des $(2r+1)^2$ pixels dont $p_{i,j}$ est le centre). Bien sûr ce calcul est à effectuer pour chacune des composantes rouge, vert et bleu.

Exercice 6

Ecrivez une fonction **tPixel floumoy(tImage im, int i, int j, int r)** qui retourne le pixel obtenu en calculant la moyenne des pixels situés dans le carré de rayon r autour du pixel de coordonnées (i, j) de l'image **im**.
Faites attention de bien gérer les bords de l'image ! Il n'y a pas $(2r+1)^2$ pixels autour d'un pixel $P_{i,j}$ si celui-ci se trouve à moins de r pixels d'un bord.

Exercice 7

Ecrivez une fonction **tImage flou(tImage im, int n)** qui retourne comme résultat une nouvelle image créée en floutant l'image donnée en paramètre avec un rayon de floutage r. L'image résultat est obtenue en invoquant la fonction **floumoy** sur chacun des pixels de l'image donnée.

Détourage

Exercice 8

Obtenir les contours des éléments d'une image peut être réalisé à partir du floutage gaussien en remplaçant chaque pixel $P_{i,j}$ de l'image de départ par le pixel $D_{i,j}$ obtenu avec la formule :

$$D_{i,j} = Blanc - |P_{i,j} - F_{i,j}|$$

où *Blanc* désigne le pixel blanc dont les composantes r, v et b valent toutes 255 et dans laquelle le pixel flou $F_{i,j}$ est calculé avec un rayon $r = 2$.

Ecrivez une fonction **tImage contours(tImage im)** qui retourne comme résultat une nouvelle image contenant les contours de l'image donnée.

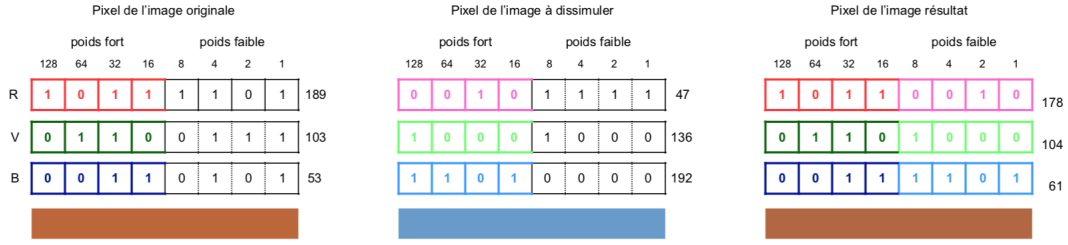
1.3 Stéganographie

Pour cacher une image ou un texte dans une image vous allez utiliser la méthode des bits de poids faible. Il s'agit de modifier les pixels de l'image originale en stockant l'information à cacher dans les bits de poids faibles des pixels.

Chaque pixel d'une image et codé par un triplet d'octets exprimant l'intensité de 0 à 255 des trois couleurs rouge, vert et bleu. Modifier légèrement les intensités de rouge de vert et de bleu d'un pixel ne modifie que peu le rendu de la couleur du pixel. C'est sur cela que repose la méthode des bits de poids faible.

Dissimuler une image dans une autre

Cacher. Le schéma suivant décrit la méthode utilisée pour dissimuler une image dans une autre. Les pixels de l'image originale sont modifiés en remplaçant leurs 4 bits de poids faible par les 4 bits de poids fort de l'image à dissimuler. Cette opération ne modifie au plus que de 15 la valeur des intensités de chaque couleur et les pixels de l'image résultat ne diffèrent que peu des pixels originaux.



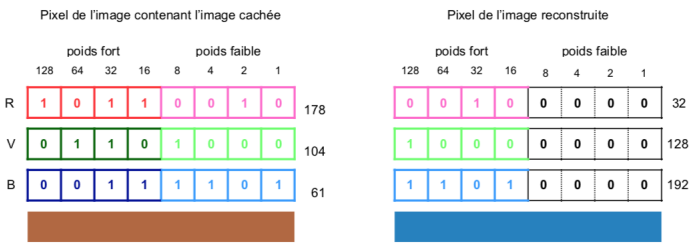
Exercice 9

Ecrivez une fonction **int fusionOctets(int octet1, int octet2)** qui prend en paramètres deux octets (i.e. 2 entiers compris entre 0 et 255) et retourne comme résultat l'octet obtenu en remplaçant les 4 bits de poids faible de *octet1 par les 4 bits de poids fort de* octet2. Étant donné un octet, les 4 bits de poids faible sont obtenus en calculant le reste de la division entière par 16. Et les quatre bits de poids fort en soustrayant à la valeur de l'octet la valeur des 4 bits de poids faible. Ainsi pour l'octet $10011110 1_2 = 189$, les 4 bits de poids faible valent $189\%16 = 13$ soit 1101_2 et les quatre bits de poids fort valent $189 - 13 = 176$ soit 1011_2.

Exercice 10

Ecrivez une fonction **tImage cacheImage(tImage originale, tImage adissimuler)** qui prend en paramètres deux images et qui retourne comme résultat une nouvelle image obtenue en dissimulant **adissimuler** dans **originale**.
Cette fonction doit vérifier que l'opération est faisable, c'est à dire que les deux images sont de même dimension et de même type. Si ça n'est pas le cas la fonction affiche une message d'erreur et retourne une copie de l'image originale.
Vous utiliserez la fonction **int fusionOctets** pour calculer les intensités de rouge, vert et bleu des pixels de l'image résultat obtenus à partir des pixels correspondants des images **originale** et **adissimuler**.

Révéler. Le schéma suivant indique comment retrouver une image qui a été dissimulée dans une autre. Il s'agit de reconstruire les pixels de l'image cachée à partir des bits de poids faible de l'image où elle est dissimulée. L'image ainsi reconstruite est dégradée par rapport à l'image de départ car l'opération de dissimulation à fait perdre les bits de poids faible qui sont remplacés par des zéros lorsqu'on reconstruit l'image.



Exercice 11

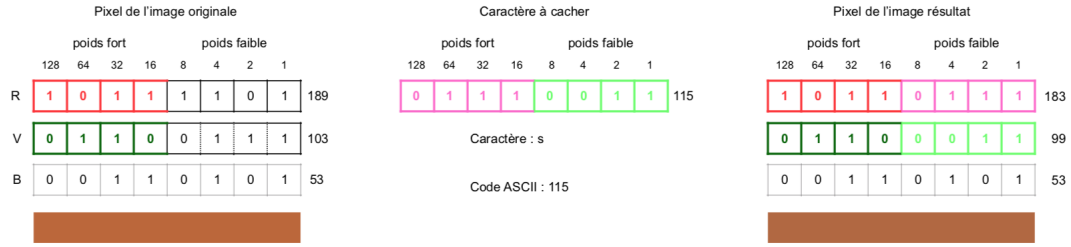
Ecrivez une fonction **tImage reveleImage(tImage im)** qui retourne une image reconstruite à partir des bits de poids faible des pixels de l'image donnée en paramètre.

Dissimuler du texte dans une image

Pour dissimuler du texte dans une image on utilise un principe analogue à celui de cacher une image dans une autre. Un caractère étant représenté en codage ASCII par un octet il s'agit de cacher cet octet dans les bits de poids faible d'un pixels de l'image.

Cependant, si le fait de perdre les bits de poids faible d'une image qu'on cache ne change que peu les couleurs et n'empêche pas de la reconstruire il ne faut surtout pas perdre d'information lorsqu'on cache un texte sinon on ne peut pas retrouver le caractère de départ. Il faut donc cacher dans l'image à la fois les bits de poids fort et les bits de poids faible de chaque caractère du texte.

Le schéma suivant indique comment cacher un caractère dans un pixel. Seul les composantes rouge et vert du pixel sont utilisées : les 4 bits de poids fort du caractère sont dissimulés dans les bits de poids faible de la composante rouge, les 4 bits de poids faible du caractère sont dissimulés dans les bits de poids faible de la composante vert et la composante bleu n'est pas modifiée.



Exercice 12

Ecrivez une fonction **tPixel cacheCarac(tPixel pix, char c)** qui retourne comme résultat le pixel obtenu en cachant le caractère **c** dans le pixel **pix** en suivant le principe du schéma ci-dessus.

Exercice 13

Ecrivez une fonction **char extraitCaract(tPixel pix)** qui réalise l'opération inverse de **cacheCarac**, c'et à dire qui retourne le caractère caché dans les bits de poids faible des composante rouge et vert du pixel **pix**.

Exercice 14

Ecrivez une fonction **tImage cacheTexte(tImage im, char* lefichier)** qui prend en paramètres une image **im**, le nom d'un fichier de caractères contenant un texte à dissimuler et qui retourne comme résultat une image dans laquelle on a caché les caractères du texte.
Attention, cette fonction prend en paramètre le nom du fichier qui contient le texte à dissimuler et non pas le texte lui même. Il faut donc ouvrir ce fichier en lecture et le lire caractère par caractère.
Lorsque la fin du fichier est atteinte on indique la fin du texte dans l'image en cachant le caractère de code ASCII 0.
Cette fonction doit vérifier que le fichier dont le nom est donné en paramètre ne provoque pas d'erreur à l'ouverture. Si l'ouverture ne réussit pas l'opération est impossible à réaliser et la fonction retourne comme résultat une copie de l'image après avoir afficher un message d'erreur.

Exercice 15

Ecrivez une fonction **void reveleTexte(tImage im, char* fichExtrait)** qui extrait d'une image le texte caché qu'elle contient. Il s'agit d'extraire les caractères cachés dans chacun des pixels de l'image et de les écrire dans un fichier texte dont le nom est donné en paramètre.
La fin du texte à extraire est repérée lorsque le caractère extrait d'un pixel est le caractère de code ASCII 0. Les pixels suivants ne contiennent pas de texte !

2 Programme principal

Exercice 16

Ecrivez la fonction `main.c` de votre programme de manière à tester toutes les opérations programmées.
Le programme principal doit au minimum afficher le menu suivant puis réaliser l'opération choisie :

```
Quelle opération voulez-vous effectuer ?
1. Transformer une image en niveau de gris
2. Flouter une image
3. Détourer une image
4. Dissimuler une image dans une autre
5. Révéler une image cachée une autre
6. Dissimuler du texte dans une image
7. Révéler un texte caché dans une image
0. Quitter
Choix ==>
```

Selon le choix effectué il devra demander les noms des fichiers contenant les images et/ou le texte à manipuler ainsi que le nom du fichier dans lequel sauver l'image résultat de l'opération ou le nom du fichier dans lequel écrire le texte révélé.