U.E. S2IN324 - CCTP

Indications : Ce devoir est à réaliser en binôme (ou individuellement) et à rendre exclusivement par dépôt sur le site Moodle du cours en trois étapes (chaque partie est évaluée sur 20) :

- **avant le lundi 18 novembre 23h55 - Partie 1.** Déposer une archive contenant les fichiers `libimage.h` **complet** (avec les en-têtes de toutes les fonctions, y compris celles des parties 2 et 3) et le fichier `libimage.c` contenant uniquement le code des fonctions de la partie 1, exercices 1 à 8.
  Un corrigé de cette première étape vous sera donné le 19/11 afin de pouvoir continuer le devoir sur de bonnes bases si nécessaire).
- **avant le lundi 2 décembre 23h55 - Partie 2.** Déposer uniquement le fichier `libimage.c` complété avec le code des fonctions de la partie 2, exercices 9 à 13.
- **avant le vendredi 20 décembre 23h55 -Partie 3.** : Déposer une archive contenant tous les fichiers du devoir complet : `libimage.h`, `libimage.c` complété avec le code des fonctions des exercices 14 à 17, et le fichier `main.c` contenant le programme principal (exercice 18).

Travail à réaliser
Il s'agit de fournir un module `libimage` (fichier en-tête et fichier source) permettant de manipuler des images au travers des quelques opérations demandées dans l'énoncé et un programme principal permettant de tester ce module.
A terme le module doit fournir les types de données définis ci-dessous ainsi que toutes les fonctions des exercices 1 à 17.
Le programme principal (exercice 18) doit permettre d'utiliser les différentes opérations du module.
Votre devoir terminé sera donc constitué de 3 fichiers : `libimage.h`, `libimage.c` et `main.c`.

Ce devoir est à faire de préférence en binôme (pas à plus de 2) mais peut être fait seul.
Si vous faites le devoir à deux, vous ne devez le déposer que sur un seul de vos comptes Moodle (le même pour les 3 dépôts).
Une fois un binôme constitué il reste le même pour tout le devoir.
Vous indiquerez en commentaire au début de chacun des fichiers de code : les numéros étudiant, noms et prénoms des protagonistes.

Le soin, la clarté du code et les commentaires seront pris en compte pour une partie non négligeable dans la notation.

Le code que vous rendez doit nécessairement compiler sans erreur. Il se peut qu'il plante à l'exécution mais il doit pouvoir être compilé et fournir une exécutable. **Un code qui ne compile pas ne pourra obtenir la moyenne.**

Tout devoir qui apparaîtra de manière trop flagrante comme étant copié en tout ou en partie d'un autre devoir sera sanctionné pour chacun des binômes sans chercher qui à copié sur qui ou qui à donné son code à qui.

Introduction au traitement d'image

Ce devoir propose d'aborder le traitement d'image à travers quelques opérations à effectuer sur une image au travers d'une représentation en mémoire sous forme de tableau. Le but est d'écrire un module (`libimage`) contenant ces différentes opérations de traitement d'images.

Dans la première partie vous allez voir comment lire et stocker une image en couleur ou en niveau de gris en mémoire et la sauver dans un fichier. Dans la deuxième partie vous coderez des opérations permettant de modifier une image : conversion d'une image couleur en niveaux de gris, négatif d'une image couleur ou en niveau de gris, floutage et détourage d'une image couleur. Dans la troisième partie vous coderez des techniques de stéganographie pour dissimuler une image couleur ou du texte dans une image couleur ainsi que les opérations inverses pour extraire une image ou un texte caché dans une image. Vous fournirez aussi un programme principal pour tester toutes les fonctionnalités 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 peut être vue comme 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 canaux de 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).

Formats de fichier graphique pour les échanges.

Le portable pixmap file format (PPM), le portable graymap file format (PGM) et le portable bitmap file format (PBM) sont des formats de fichier graphique permettant de représenter les trois types d'images numériques et utilisés pour les échanges. Ces fichiers sont composés sur la même base :

- le *nombre magique* du format qui indique le type de format et sa représentation binaire ou ASCII :
  - P1 (ASCII) ou P4 (binaire) pour les images en noir est blanc, extension de fichier `.pbm` ;
  - P2 (ASCII) ou P5 (binaire) pour les images en niveaux de gris, extension `.pgm` ;
  - P3 (ASCII) ou P6 (binaire) pour les images couleurs, extension `.ppm` ;
- un caractère d'espacement (espace, tabulation ou passage à la ligne)
- suivent éventuellement quelques lignes de commentaires qui débutent par le caractère `#`
- la largeur de l'image en nombre de pixels, écrit explicitement sous forme d'un nombre en caractères ASCII
- un caractère d'espacement,
- la hauteur de l'image,
- un caractère d'espacement,
- **uniquement pour les fichiers en niveau de gris et en couleur** : la valeur maximale utilisée pour coder les niveaux de gris ou les niveaux de rouge, vert et bleu. Cette valeur maximale, la plupart du temps 255, doit être inférieur à 65535. Cette valeur est suivie d'un espacement.
- suivent ensuite les valeurs associées à chaque pixel, décrivant l'image ligne par ligne et pixel par pixel, depuis le premier en haut à gauche de la première ligne de l'image jusqu'au dernier en bas à droite de la dernière ligne. Si c'est une image en couleur, à un pixel correspondent 3 entiers successifs dans le fichier et si c'est une image en niveau de gris chaque entier du fichier correspond à un pixel.

Dans ce devoir, vous ne manipulerez que des images de type P2 et P3.

Exemple de fichier d'image couleur de type P3

Voici le début du fichier `.ppm` d'une image en couleur de type P3 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 six entiers qui suivent représentent les niveaux de (rouge, vert, bleu) des deux premiers pixels de l'image : (105, 145, 204) et (113, 150, 227). Chaque pixel de l'image est donc codé par 3 valeurs donnant les intensités de rouge, de vert et de bleu.

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

Exemple de fichier d'image en niveau de gris de type P2

Voici le début du fichier `.pgm` d'une image en niveau de gris de type P2 codée en ascii. 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 du niveau de gris : 255. Les quatre entiers qui suivent représentent les niveaux de gris des quatre premiers pixels de l'image : 128, 137, 160 et 212. Chaque pixel de l'image est donc codé par un seul entiers donnant le niveau de gris du pixel.

```
P2
# CREATOR: GIMP PGM Filter Version 1.1
850 1300
255
128
137
160
212
...
```

1 Partie 1 - Stockage d'une image en mémoire. Lecture depuis un fichier et écriture dans un fichier de type ppm ou pgm

On donne ci-dessous (ainsi que dans le fichier `pourCCTP.txt` qui accompagne cet énoncé) la définition des types que vous utiliserez pour représenter des images en couleur ou en niveaux de gris en mémoire.

Le champ `PixelRGB** tpix` du type `ImP3` est un tableau rectangulaire de pixels, c'est à dire un tableau de tableaux de PixelRGB. 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 `tpix[i][j]` contient le pixel situé en i^{eme} ligne et j^{eme} colonne (en comptant à partir de 0).
Le champ `unsigned char** tpix` du type `ImP2` est un tableau rectangulaire **d'octets non signés**.

Ces définitions sont à recopier dans le fichier en-tête `libimage.h`.

```
// ***********************************************
// Le type PixelRGB pour représenter un pixel par ses 3 canaux : rouge, vert, bleu
// Chaque quantité de couleur est comprise entre 0 et 255
// d'où le type unsigned char (un entier non signé sur un octet)

typedef struct pixelrgb {
    unsigned char r;
    unsigned char v;
    unsigned char b;
} PixelRGB;

// Le type ImP3 pour stocker une image de type P3 dans un tableau rectangulaire de pixels
typedef struct image3 {
    int hauteur;        // Hauteur en pixels
    int largeur;        // Largeur en pixels
    int maxval;         // Valeur maximal d'un pixel
    PixelRGB** tpix;       // Le tableau des pixels
} ImP3;

// Le type ImP2 pour stocker une image de type P2 dans un tableau rectangulaire d'octets

typedef struct image2 {
    int hauteur;        // Hauteur en pixels
    int largeur;        // Largeur en pixels
    int maxval;         // Valeur maximal d'un pixel
    unsigned char** tpix;       // Le tableau des pixels gris = octets
} ImP2;
// ***********************************************
```

1.1 Représentation en mémoire des images de type P2 et P3 : initialisation et copie d'une image en mémoire

Exercice 1

Ajoutez au module une fonction **ImP3 initImP3(int haut, int larg, int vmax)** qui retourne une structure `ImP3` dans laquelle les champs `hauteur` et `largeur` sont initialisés avec les valeurs données en paramètre et qui réalise l'allocation du tableau `tpix` pour `haut` lignes de `larg` pixels en couleur. Le champs `maxval` sera initialisé à `vmax`.

Exercice 2

Ajoutez au module une fonction **ImP3 copieImP3(ImP3 im)** qui réalise la copie de l'image donnée en paramètre dans une nouvelle image (une structure ImP3) 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

Ajoutez au module une fonction **ImP2 initImP2(int haut, int larg, int vmax)** qui retourne une structure `ImP2` dans laquelle les champs `hauteur` et `largeur` sont initialisés avec les valeurs données en paramètre et qui réalise l'allocation du tableau `tpix` pour `haut` lignes de `larg` pixels en niveau de gris. Le champs `maxval` sera initialisé à `vmax`.

Exercice 4

Ajoutez au module une fonction **ImP2 copieImP2(ImP2 im)** qui réalise la copie de l'image donnée en paramètre dans une nouvelle image (une structure ImP2) 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.

1.2 Chargement en mémoire d'une image à partir d'un fichier - Sauvegarde d'une image dans un fichier

Le chargement d'une image en mémoire consiste à initialiser et remplir une structure de type `ImP3` (resp. `ImP2`) à partir des informations contenu dans un fichier `.ppm` (reps. `.pgm`) en respectant le format des fichiers `.ppm` et `pgm` tels qu'ils sont décrits plus haut.

Exercice 5 — Chargement d'une image de type P3

Ajouter au module une fonction **ImP3 chargeImP3(char* fichier)** qui charge dans une structure `ImP3` l'image couleur contenue dans le fichier dont le nom est donné en paramètre. Votre fonction devra prendre en compte deux cas d'erreur :

- s'il y a une erreur à l'ouverture en lecture du fichier demandé
- si le type de l'image contenu dans le fichier n'est pas `P3`

En cas d'erreur, la fonction **ImP3 chargeImP3(char* fichier)** doit retourner une image vide, c'est à dire une image de hauteur et largeur 0 et dont le tableau de pixels est par conséquent le pointeur NULL.

Exercice 6 — Chargement d'une image de type P2

Ajouter au module une fonction **ImP2 chargeImP2(char* fichier)** qui charge dans une structure `ImP2` l'image en niveau de gris contenue dans le fichier dont le nom est donné en paramètre. Votre fonction devra prendre en compte deux cas d'erreur :

- s'il y a une erreur à l'ouverture en lecture du fichier demandé
- si le type de l'image contenu dans le fichier n'est pas `P2`

En cas d'erreur, la fonction **ImP2 chargeImP2(char* fichier)** doit retourner une image vide, c'est à dire une image de hauteur et largeur 0 et dont le tableau de pixels est par conséquent le pointeur NULL.

> **Indications :** *Lors de la lecture du fichier, vous devez « passer » les lignes de commentaires, c'est dire à toutes les lignes qui commencent par un* `#`*.*
>
> *Pour cela vous pouvez, par exemple, écrire une boucle qui lit le premier caractère d'une ligne et, tant que celui-ci est un* `#`*, passe tous les caractères jusqu'au début de la ligne suivante. A la sortie de cette boucle vous aurez lu le premier caractère de la ligne qui contient les dimensions de l'image. Vous pouvez replacer ce*

caractère dans le flot de lecture grâce à la fonction `ungetc(int c, FILE *stream)` *puis lire les dimensions avec la fonction* `fscanf`

Vous pouvez aussi écrire une boucle qui lit une ligne entière dans un chaîne suffisamment grande tant que le premier caractère est un `#`*. En sortant de cette boucle, la ligne lue n'a pas de* `#` *comme premier caractère mais correspond à la ligne contenant les dimensions de l'image que vous pouvez extraire à l'aide de la fonction* `sscanf` *(lecture depuis une chaîne).*

Pour la lecture des pixels, chaque donnée étant séparée de la suivante par un espace ou un passage à la ligne vous pouvez utiliser la fonction `fscanf` *pour récupérer les valeurs des canaux.*

Exercice 7

Ajoutez une fonction **void sauveImP3(char* nom, ImP3 im)** qui créer un fichier image `.ppm` dont le nom est donné en paramètre et dans lequel vous sauvez le contenu de la structure `im` donnée en paramètre en respectant le format d'un fichier image couleur `.ppm` de type P3 .
Vous ajouterez dans ce fichier une ligne de commentaire contenant votre nom et prénom pour renseigner le créateur du fichier.
Utilisez `fprintf` pour écrire dans le fichier.

Exercice 8

Ajoutez une fonction **void sauveImP2(char* nom, ImP2 im)** qui créer un fichier image `.pgm` dont le nom est donné en paramètre et dans lequel vous sauvez le contenu de la structure `im` donnée en paramètre en respectant le format d'un fichier image en niveau de gris `.pgm` de type P2 .
Vous ajouterez dans ce fichier une ligne de commentaire contenant votre nom et prénom pour renseigner le créateur du fichier.
Utilisez `fprintf` pour écrire dans le fichier.

2 Partie 2 : Niveau de gris, négatif, floutage, détourage

Une fois chargée en mémoire les modifications qu'on peut effectuer sur une image consistent essentiellement à appliquer des calculs sur les pixels du tableau.

2.1 Image en niveaux de gris

À partir d'une image en couleur où chaque pixel est représenté par les trois canaux r, v et b, les pixels de l'image correspondante en niveaux de gris sont obtenus en leur affectant une nuance de gris calculée à partir des valeurs des 3 canaux.
L'oeil humain n'étant pas sensible de la même manière aux trois couleurs rouge, vert et bleu, 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 nuance de gris) d'un pixel couleur au format rvb par la formule :

Gris = 0.2125 Rouge + 0.7154 Vert + 0.0721 Bleu

Exercice 9

Ajoutez au module une fonction **ImP2 P3toP2(const ImP3 im3)** qui prend en entrée une image couleur de type P3 et renvoie l'image en niveaux de gris de type P2 correspondante, c'est à dire dans laquelle les niveaux de rouge, vert et bleu de chaque pixel sont remplacés par la valeur de luminance du pixel.

2.2 Négatif d'une image

Etant donné un pixel (r, v, b) d'une image de type P3, le pixel négatif correspondant a comme valeur de canaux $(maxval - r, maxval - v, maxval - b)$. Si c'est un pixel g d'un image en niveau de gris de type P2, le pixel négatif a comme niveau de gris $maxval - g$.

Exercice 10

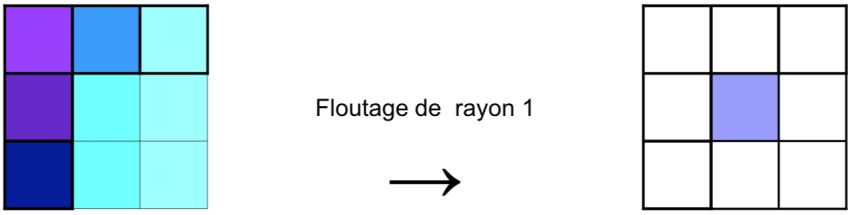
Ajoutez au module une fonction **ImP2 negatifP2(const ImP2 im)** qui renvoie l'image en négatif d'une image en niveau de gris donnée en paramètre.

Exercice 11

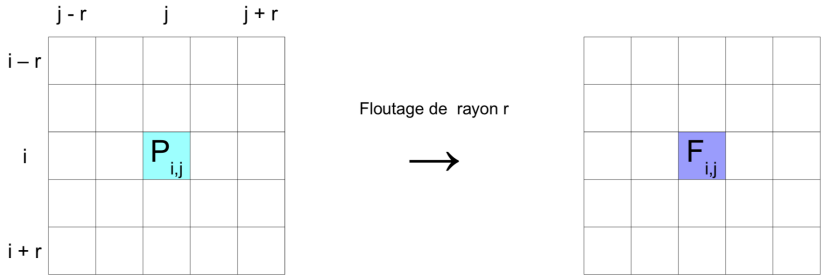
Ajoutez au module une fonction **ImP3 negatifP3(const ImP3 im)** qui renvoie l'image en négatif d'une image couleur rvb de type P3 donnée en paramètre.

2.3 Floutage et détourage

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 (image de gauche) sera flouté en le remplaçant par la moyenne des 9 pixels situés dans un rayon de 1 pixel autour de lui pour obtenir un pixel de couleur parme dans l'image floutée (image de droite) .



De manière plus générale, pour 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 $F_{i,j} = \frac{1}{(2n+1)^2}\sum_{l=i-r}^{i+r}(\sum_{c=j-r}^{j+r} P_{l,c})$. Bien sûr ce calcul est à effectuer pour chacune des composantes rouge, vert et bleu.

Exercice 12

Ajouter au module une fonction **ImP3 flou(const ImP3 im, int r)** qui renvoie comme résultat une image couleur créée en floutant l'image donnée en paramètre avec un rayon de floutage r.

Exercice 13

Le détourage d'une image peut être obtenu 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 $r = 2$.

Ajoutez une fonction **ImP3 detoure(const ImP3 im)** qui renvoie comme résultat une image créée en détourant l'image donnée en paramètre.

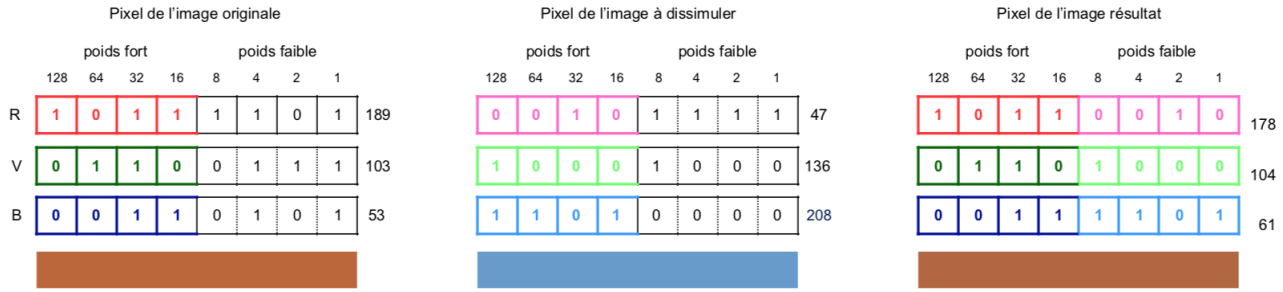
3 Partie 3 : Stéganographie et programme principal

Pour cacher une image ou un texte dans une image on peut 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.

3.1 Dissimuler une image dans une autre

3.1.1 Dissimuler

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.



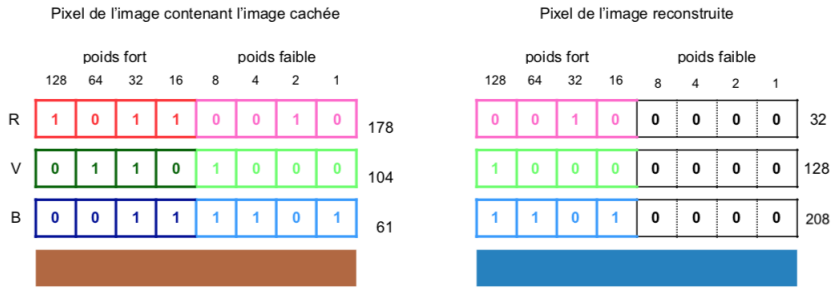
Étant donné un octet, les 4 bits de poids faible sont obtenus en calculant le reste de la division entière par 16. Et les quatre bits de poids fort en soustrayant à la valeur de l'octet la valeur des 4 bits de poids faible. Ainsi pour l'octet 10011101_2 = 189, les 4 bits de poids faible valent $189\%16 = 13$ soit 1101_2 et les quatre bits de poids fort valent $189 - 13 = 176$ soit 1011_2.

Exercice 14

Ajoutez au module une fonction **ImP3 cacheImage(const ImP3 im1, const ImP3 im2))** qui prend en paramètres deux images couleur et qui retourne comme résultat une image obtenue en dissimulant l'image **im2** dans l'image **im1**. Cette fonction doit vérifier que l'opération est faisable, c'est à dire que les deux images sont de même de même type et que les dimensions de l'image à dissimuler (im2) sont inférieures à celles de l'image dans laquelle on la dissimule (im1). Si ça n'est pas le cas la fonction affiche une message d'erreur et retourne une copie de l'image originale.

3.1.2 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.



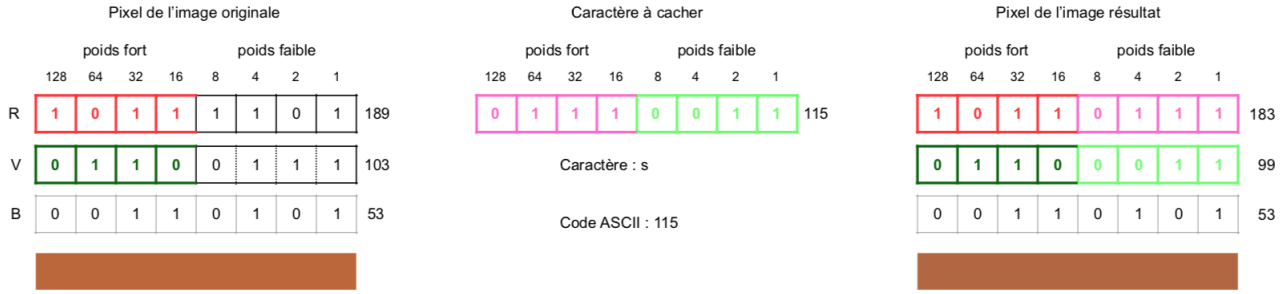
Exercice 15

Ajoutez au module une fonction **ImP3 reveleImage(const ImP3 im)** qui renvoie une image reconstruite à partir des bits de poids faible des pixels de l'image donnée en paramètre.

3.2 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 canaux rouge et vert du pixel sont utilisées : les 4 bits de poids fort du caractère sont dissimulés dans les bits de poids faible du canal rouge, les 4 bits de poids faible du caractère sont dissimulés dans les bits de poids faible du canal vert ; le canal bleu n'est pas modifié.



Exercice 16

Ajouter au module une fonction **ImP3 cacheTexte(const ImP3 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 et dissimuler les caractères dans une image.

Lorsque la fin du fichier texte est atteinte on indique la fin du texte dans l'image en cachant le caractère `'\0'` de code ASCII 0. **On supposera sans avoir à le vérifier que l'image contient suffisamment de pixels pour dissimuler le texte.**

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 17

Ecrivez une fonction **void reveleTexte(const ImP3 im, char* nonfichier)** 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 donné en parmaètre et de les écrire dans le 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 !

3.3 Programme principal

Exercice 18

Ecrivez un programme principal `main.c` qui utilise le module **libimage** et qui propose, à l'aide d'un menu, de tester toutes les opérations du module que vous avez programmées.
Faites en sorte d'avoir un programme facile à utiliser et qui laisse la possibilité à l'utilisateur d'indiquer les noms des fichiers à utiliser : aussi bien les fichiers (images ou texte) à charger et à manipuler que les fichiers dans lesquels sauvegarder les résultats des opérations choisies.