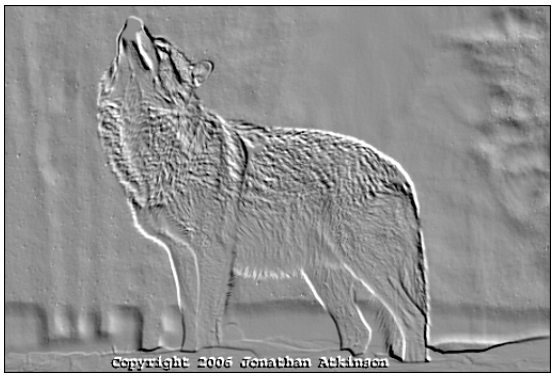
TP n° 3 : Tableaux et structures

NOM : ..... Prénom : .....

Attention, ce TP est noté :

- Les documents autorisés sont les diapositives de cours éventuellement annotées, les notes de TD et les sujets éventuellement annotés des TP 1 et 2.
- L'utilisation des logiciels de chat et d'e-mail est interdite. Les plagiats seront sanctionnés par un zéro.
- La note sera divisée par deux si le programme ne compile pas. Moralité : il faut absolument essayer de compiler au fur et à mesure ! Tout mettre en commentaires au dernier moment n'est pas une solution : les portions de code en commentaires ne seront pas évaluées.
- Vous avez cependant droit à deux « jokers » au cours des trois heures : vous pouvez appeler deux fois l'encadrant pour une erreur de compilation. Vous devrez par contre trouver vous-même les calculs à effectuer.
- Le rendu du code se fait sur Spiral, dans l'espace « Echange de docs » correspondant à votre groupe. Attention de bien uploader les fichiers .c et non .c~. Cet espace Spiral sera automatiquement fermé à 19h, réservez impérativement les 10 dernières minutes à cela. Les codes rendus en retard par e-mail ne seront pas évalués.

L'objectif de ce TP est de programmer différents traitements sur des images au format BMP 24bits : vous allez programmer le passage d'une image en niveaux de gris, puis un filtre d'embossage. Les procédures de lecture et d'écriture de fichier vous sont fournies, de sorte à ce que vous vous concentriez sur les calculs des valeurs RGB des pixels.



En haut à gauche : image originale, en couleurs

En haut à droite : image en niveaux de gris

Ci-contre : image en niveaux de gris après traitement par un filtre d'embossage, donnant une impression de relief.

Rappel : Bitmap, connu aussi sous son abréviation BMP, est un format d'image matricielle ouvert développé par Microsoft et IBM. C'est l'un des formats d'images les plus simples à développer et à utiliser pour programmer. Il est lisible par quasiment tous les visualiseurs et éditeurs d'images. Le fichier se découpe en 2 zones :

- L'en-tête du fichier : on y trouve notamment la largeur et la hauteur de l'image, en nombre de pixels.
- Les données relatives à l'image : les pixels de l'image sont codés ligne par ligne, en partant de la ligne inférieure de l'image. Si l'image est codée en 24 bits (ce qui sera le cas pour tout ce TP), chaque pixel est codé par trois octets (trois unsigned char) codant successivement les niveaux de bleu, vert et rouge. Chacun de ces trois niveaux est compris entre 0 et 255. Exemples de pixels :
  (0, 0, 0) → noir
  (255, 255, 255) → blanc
  (255, 0, 0) → rouge
  (0, 255, 0) → vert
  (0, 0, 255) → bleu
  (255, 192, 203) → une nuance possible de rose

Commencez par créer un répertoire TP3 à l'intérieur de votre répertoire LIF5, en utilisant les lignes de commandes vues au TP1 (cd, mkdir, ls…), et placez-y les fichiers disponibles sur Spiral :

- tp3-lundi.c : squelette de code à compléter,
- grey_wolf.bmp : image originale,
- verif-grayscale.bmp : ce que votre traitement « niveaux de gris » doit produire,
- verif-embossage.bmp : ce que votre traitement « embossage » doit produire.

Exercice 1 : Lire l'image originale (chargement en mémoire vive)

Le fichier tp3-lundi.c définit le type pixel, sous forme d'une struct. Il contient aussi deux procédures utiles pour lire un fichier BMP : lireEntete et remplirTableauPixelsDepuisFichier.

Lisez attentivement les entêtes de ces procédures, puis complétez le « main » pour récupérer la largeur et la hauteur de l'image originale (grey_wolf.bmp), allouer un tableau 1D de taille suffisante pour contenir tous les pixels, et remplir ce tableau à partir du fichier. Vérifiez que votre code ne contient pas d'erreur de compilation ni d'exécution.

Pour une image de largeur L et de hauteur H, quelle instruction faudrait-il écrire pour afficher les 3 niveaux R, G, B du pixel qui se trouve à la 3è ligne (en partant du bas), et à la 9è colonne ?

```
printf("red=%u  green=%u  blue=%u\n",  tabOrig[2*L  +  8].red,  tabOrig[2*L  +
8].green, tabOrig[2*L + 8].blue);
```

Exercice 2 : Traitement « niveaux de gris »

Dans le codage RGB, si l'on fixe les trois octets à la même valeur (par exemple 125, 125, 125), on obtient une nuance de gris. La conversion d'une image en niveaux de gris consiste à calculer pour chaque pixel la moyenne de ses trois niveaux R, G, B : dans l'image convertie, le pixel aura ses trois niveaux R, G, B égaux à cette moyenne.

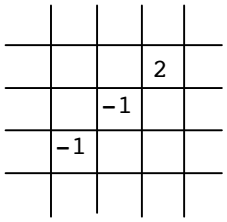
Ecrivez au-dessus du main la procédure traitementNiveauxDeGris, qui prend en paramètres un tableau de pixels correspondant à l'image originale (mode donnée), un tableau de pixels assez grand pour contenir l'image convertie (mode résultat), ainsi que la largeur et la hauteur de l'image (mode donnée). Cette procédure remplit le second tableau de pixels selon l'algorithme décrit ci-dessus. Veillez à indiquer les pré- et post-conditions en commentaires.

Complétez le main pour qu'il appelle correctement cette procédure, puis écrive le tableau de pixels résultant dans un fichier au format BMP que vous appellerez « grayscale.bmp » (utilisez pour cela la procédure ecrireFichier). Vérifiez que votre code ne contient pas d'erreur de compilation ni d'exécution. Vérifiez que l'image produite correspond au résultat attendu.

Exercice 3 : Traitement « embossage »

Le filtre d'embossage donne l'illusion optique que certains objets de l'image sont plus ou moins près du fond, ce qui crée un effet de relief. Cet effet est obtenu de la façon suivante :

- on part d'une image en niveaux de gris,
- pour chaque pixel de l'image de sortie (sauf ceux en bordure de l'image), on calcule le niveau de gris en tenant compte du niveau de ce pixel et de 2 de ses voisins dans l'image originale, selon le calcul suivant :



Niveau brut pour le pixel central = 2*niveau du pixel voisin nord-est
- niveau de ce pixel
- niveau du pixel voisin sud-ouest
+ 128

Si ce niveau brut excède 255, on met le pixel à 255.
S'il est négatif, on met le pixel à zéro.

- ce calcul n'est pas réalisable pour les pixels en bordure de l'image, on les mettra donc en noir.

Est-il pertinent d'utiliser une variable de type « unsigned char » pour stocker le niveau brut ? Pourquoi ? Quel type suggérez-vous ?

Non, car on risque des overflows. Un unsigned char ne permet de stocker des nombres qu'entre 0 et 225, or on peut avoir des nombres négatifs ou supérieurs à 255. On préfèrera donc un signed int.
Exemple 1 : voisin nord-est=250, pixel central=0, voisin sud-ouest=0 → niveau brut = 628
Exemple 2 : voisin nord-est=0, pixel central=200, voisin sud-ouest=210 → niveau brut = -282

Ecrivez au-dessus du main la procédure traitementEmbossage, qui prend en paramètres un tableau de pixels correspondant à une image en niveaux de gris (mode donnée), un tableau de pixels assez grand pour contenir l'image embossée (mode résultat), ainsi que la largeur et la hauteur de l'image (mode donnée). Cette procédure remplit le second tableau de pixels selon l'algorithme décrit ci-dessus. Veillez à indiquer les pré- et post-conditions en commentaires.

Complétez le main pour qu'il appelle correctement cette procédure, puis écrive le tableau de pixels résultant dans un fichier au format BMP que vous appellerez « embossage.bmp ». Vérifiez que votre code ne contient pas d'erreur de compilation ni d'exécution. Vérifiez que l'image produite correspond au résultat attendu.

Votre main doit se terminer proprement et donc libérer la mémoire éventuellement allouée dynamiquement.

```
#include <stdio.h>
#include <stdlib.h>


/* ---------------------------------- */
/*     Definition du type pixel       */
/* ---------------------------------- */

struct sPixel
{
  unsigned char red;
  unsigned char green;
  unsigned char blue;
};
typedef struct sPixel pixel;



/* ------------------------------------------------------------- */
/*     Declarations des procedures que vous pouvez appeler       */
/* (deja ecrites, code disponible pour information apres le main) */
/* ------------------------------------------------------------- */


/* Precondition : nomFichier est le nom d'un fichier au format Bitmap,
   24-bit par pixel (donc pas de palette), non compresse.
   Postconditions : largeur et hauteur contiennent la largeur (nombre de colonnes)
   et la hauteur (nombre de lignes) de l'image, en nombre de pixels.
*/
void lireEntete(const char nomFichier[], unsigned long * largeur, unsigned long *
hauteur);



/* Preconditions : tab est un tableau 1D de pixels, suffisamment grand pour contenir
   tous les pixels de l'image. nomFichier est le nom d'un fichier au format Bitmap,
   24-bit par pixel (donc pas de palette), non compresse. largeur et hauteur
   sont les dimensions de l'image en nombre de pixels.
   Postcondition : Chaque case de tab correspond a un pixel de l'image et contient
   ainsi les valeurs RGB de ce pixel. Les premieres cases de tab contiennent la ligne
   de pixels tout en bas de l'image. Les cases suivantes contiennent les pixels de la
   ligne juste au-dessus (2e ligne en partant du bas), etc, jusqu'a la ligne du haut
   de l'image.
*/
void remplirTableauPixelsDepuisFichier(const char nomFichier[], pixel tab[], \
                                       unsigned long largeur, unsigned long hauteur);



/* Preconditions : tab est un tableau contenant largeur*hauteur pixels.
   Postconditions : Un nouveau fichier est cree, nomme comme precise dans la chaine
   de caracteres nomFic. Il est au format Bitmap, 24 bits par pixel, non compresse.
   Si un fichier de ce nom existait deja, il est ecrase.
 */
void ecrireFichier(const char nomFic[], const pixel tab[], \
                   unsigned long largeur, unsigned long hauteur);



/* ------------------------------------------------------------ */
/*     Ecrivez ci-dessous les codes des procedures demandees    */
/* ------------------------------------------------------------ */
```

```
/* Precond: tabSortie assez grand */
void traitementNiveauxDeGris(const pixel tabEntree[], pixel tabSortie[], \
                             unsigned long largeur, unsigned long hauteur)
{
  unsigned long ligne, colonne, index;
  unsigned int moy;


  for(ligne = 0; ligne < hauteur; ligne ++)
    {
      for (colonne = 0; colonne < largeur; colonne ++)
        {
          index = (ligne*largeur) + colonne;
          moy = (tabEntree[index].red + tabEntree[index].green + tabEntree[index].blue) /
3;
          tabSortie[index].red = moy;
          tabSortie[index].green = moy;
          tabSortie[index].blue = moy;
        }
    }

}



/* Precond : tabSortie assez grand + tabEntree en niveaux de gris */
void traitementEmbossage(const pixel tabEntree[], pixel tabSortie[], \
                         unsigned long largeur, unsigned long hauteur)
{
  unsigned long ligne, colonne, index;
  int nouvcouleur;

  for(ligne = 0; ligne < hauteur; ligne ++)
    {
      for (colonne = 0; colonne < largeur; colonne ++)
        {

          index = largeur*ligne + colonne;

          if ((ligne == 0) || (ligne == hauteur-1) ||(colonne == 0) || (colonne ==
largeur-1) )
            {
              tabSortie[index].red = tabSortie[index].green = tabSortie[index].blue = 0;
            }
          else
            {
              nouvcouleur = 2 * tabEntree[largeur*(ligne+1) + colonne + 1].red \
                - tabEntree[index].red \
                - tabEntree[largeur*(ligne-1) + colonne - 1].red + 128;
              if (nouvcouleur > 255) nouvcouleur = 255;
              if (nouvcouleur < 0) nouvcouleur = 0;

              tabSortie[index].red = tabSortie[index].green = tabSortie[index].blue =
nouvcouleur;
            }
        }
    }
}



/* ------------------------- */
/*     Completez le main     */
/* ------------------------- */


int main()
{

  char nomFicImage1[] = "grey_wolf.bmp";
```

```
  char nomFicImageNivGris[] = "grayscale.bmp";
  char nomFicImageEmbossage[] = "embossage.bmp";
  unsigned long largeur, hauteur;


  pixel *tab1, *tab2, *tab3;

  lireEntete(nomFicImage1, &largeur, &hauteur);

  tab1 = (pixel *) malloc(largeur*hauteur*sizeof(pixel)); /*image originale*/
  tab2 = (pixel *) malloc(largeur*hauteur*sizeof(pixel)); /*image niveaux de gris*/
  tab3 = (pixel *) malloc(largeur*hauteur*sizeof(pixel)); /*image embossee*/

  remplirTableauPixelsDepuisFichier(nomFicImage1, tab1, largeur, hauteur);

  /* niveaux de gris */
  traitementNiveauxDeGris(tab1, tab2, largeur, hauteur);
  ecrireFichier(nomFicImageNivGris, tab2, largeur, hauteur);

  /* embossage */
  traitementEmbossage(tab2, tab3, largeur, hauteur);
  ecrireFichier(nomFicImageEmbossage, tab3, largeur, hauteur);


  /* liberation memoire */
  free(tab1);
  free(tab2);
  free(tab3);

  return 0;
}



/* Procedure auxiliaire utile pour lire les fichiers */
void fskip(FILE *fp, int num_bytes)
{
  int i;
  for (i=0; i<num_bytes; i++) {fgetc(fp);}
}



void lireEntete(const char nomFichier[], unsigned long * largeur, unsigned long *
hauteur)
{

  FILE * fic = fopen(nomFichier, "rb");
  unsigned short bitsparpixel;
  unsigned long compression, nbcolorsinpalette, nbimportantcolors;

  if (fgetc(fic)!='B' || fgetc(fic)!='M')
  {
    fclose(fic);
    fprintf(stderr, "%s n'est pas au format BMP.\n",nomFichier);
    exit(EXIT_FAILURE);
  }

  fskip(fic,16);
  fread(largeur, sizeof(unsigned long), 1, fic);
  fread(hauteur,sizeof(unsigned long), 1, fic);

  fskip(fic, 2); /* skipping the number of color planes (always 1) */
  fread(&bitsparpixel,sizeof(unsigned short), 1, fic);

  if(bitsparpixel != 24)
    {
      fclose(fic);
      fprintf(stderr, "Erreur: le nombre de bits par pixel n'est pas 24, \n");
      fprintf(stderr, "ce format d'image n'est pas supporte par ce programme. \n");
      exit(EXIT_FAILURE);
    }
```

```
  fread(&compression,sizeof(unsigned long), 1, fic);
  if(compression != 0)
    {
      fclose(fic);
      fprintf(stderr, "Erreur: le mode de compression n'est pas 0, \n");
      fprintf(stderr, "ce format d'image n'est pas supporte par ce programme. \n");
      exit(EXIT_FAILURE);
    }

  fskip(fic,12);
  fread(&nbcolorsinpalette,sizeof(unsigned long), 1, fic);
  if(nbcolorsinpalette != 0)
    {
      fclose(fic);
      fprintf(stderr, "Erreur: le nombre de couleurs dans la palette n'est pas 0, \n");
      fprintf(stderr, "ce format d'image n'est pas supporte par ce programme. \n");
      exit(EXIT_FAILURE);
    }

  fread(&nbimportantcolors,sizeof(unsigned long), 1, fic);
  if(nbimportantcolors != 0)
    {
      fclose(fic);
      fprintf(stderr, "Erreur: le nombre de couleurs importantes n'est pas 0, \n");
      fprintf(stderr, "ce format d'image n'est pas supporte par ce programme. \n");
      exit(EXIT_FAILURE);
    }


  fclose(fic);
}



void remplirTableauPixelsDepuisFichier(const char nomFichier[], pixel tab[], \
                                       unsigned long largeur, unsigned long hauteur)
{
  FILE * fic = fopen(nomFichier, "rb");
  unsigned long ligne, colonne, padding;

  fskip(fic, 54);
  for(ligne = 0; ligne < hauteur; ligne ++)
    {
      for (colonne = 0; colonne < largeur; colonne ++)
        {
          tab[(ligne*largeur) + colonne].red = fgetc(fic);
          tab[(ligne*largeur) + colonne].green = fgetc(fic);
          tab[(ligne*largeur) + colonne].blue = fgetc(fic);
        }

      /* Padding for 4 byte alignment */
      if( (3*largeur)%4 == 0) padding = 0;
      else padding = 4 - ((3*largeur)%4);
      fskip(fic, padding);
    }

  fclose(fic);
}


void ecrireFichier(const char nomFic[], const pixel tab[], \
                   unsigned long largeur, unsigned long hauteur)
{
  FILE * fic = fopen(nomFic, "wb");
  unsigned long monLong;
  unsigned short monShort;
  unsigned char monChar, i;
  unsigned long padding, ligne, colonne, index;

  char format[] = {'B', 'M'};
  unsigned long tailleFic = 54 + largeur*hauteur*3;
```

```
  fwrite(format, sizeof(char), 2, fic);

  /* size of the file in bytes */
  fwrite(&tailleFic, sizeof(unsigned long), 1, fic);

  /* Unused, app specific */
  monShort = 0;
  fwrite(&monShort, sizeof(unsigned short), 1, fic);
  fwrite(&monShort, sizeof(unsigned short), 1, fic);

  /* Offset for pixel data */
  monLong = 54;
  fwrite(&monLong, sizeof(unsigned long), 1, fic);

  /* Nb of bytes in the header from this point */
  monLong = 40;
  fwrite(&monLong, sizeof(unsigned long), 1, fic);

  fwrite(&largeur, sizeof(unsigned long), 1, fic);
  fwrite(&hauteur, sizeof(unsigned long), 1, fic);

  /* Nb of color planes (always 1) */
  monShort = 1;
  fwrite(&monShort, sizeof(unsigned short), 1, fic);

  /* Nb of bits per pixel */
  monShort = 24;
  fwrite(&monShort, sizeof(unsigned short), 1, fic);

  /* Compression mode */
  monLong = 0;
  fwrite(&monLong, sizeof(unsigned long), 1, fic);


  /* Size of the raw BMP data (after this header), including padding */
  if( (3*largeur)%4 == 0) padding = 0;
  else padding = 4 - ((3*largeur)%4);
  monLong = (largeur*3 + padding)*hauteur;
  fwrite(&monLong, sizeof(unsigned long), 1, fic);


  /* Horiz and vertic resolutions (pixel per metre) */
  monLong = 2835;
  fwrite(&monLong, sizeof(unsigned long), 1, fic);
  fwrite(&monLong, sizeof(unsigned long), 1, fic);


  /* Numbers of colors in the palette & number of important colors */
  monLong = 0;
  fwrite(&monLong, sizeof(unsigned long), 1, fic);
  fwrite(&monLong, sizeof(unsigned long), 1, fic);


  /* Pixel data */
    for(ligne = 0; ligne < hauteur; ligne ++)
    {
      for (colonne = 0; colonne < largeur; colonne ++)
        {
          index = largeur*ligne + colonne;
          fwrite(&tab[index].red, sizeof(char), 1, fic);
          fwrite(&tab[index].green, sizeof(char), 1, fic);
          fwrite(&tab[index].blue, sizeof(char), 1, fic);
        }

      /* Padding for 4 byte alignment */
      if( (3*largeur)%4 == 0) padding = 0;
      else padding = 4 - ((3*largeur)%4);
      monChar = 0;
      for (i = 0; i < padding; i++) fwrite(&monChar, sizeof(char), 1, fic);
    }

  fclose(fic);
}
```