INF1005C – PROGRAMMATION PROCÉDURALE
Travail dirigé no 5
Enregistrements et fichiers binaires

Objectif : Permettre à l'étudiant de manipuler des fichiers binaires et des enregistrements, ainsi que de maîtriser les concepts d'accès direct et séquentiel.

Durée : Deux séances de laboratoire.

Remise du travail : Avant 23h30 le dimanche 16 novembre 2025.

Travail préparatoire : Lecture des exercices et de la documentation fournie et rédaction des algorithmes.

Directives : N'oubliez pas les entêtes de fichiers. Vous devez aussi ajouter un entête pour chaque fonction. Dans l'écriture de l'entête d'une fonction, ne pas oublier de donner la description IN, OUT ou IN/OUT pour chacun des paramètres. Il est interdit d'utiliser les variables globales, sauf celles en lecture seule (constantes). De plus, vous devez aussi des for avec range ou des span (**aucun for traditionnel**). Suivez en tout temps le guide de codage.

Documents à remettre : sur le site Moodle des travaux pratiques, vous remettrez les fichiers *CodeDemande.cpp* et *CodeDemande.hpp* compressés dans un fichier .zip en suivant la procédure de remise des TDs.

Mise en contexte

Avec votre expérience acquise au TD4, et vos récents apprentissages sur les fichiers binaires, on vous demande de compléter le module d'un logiciel qui permet de faire des casse-têtes.

Dans le TD, on utilise des images Bitmap, ou BMP. Le BMP est un format de fichier d'image très simple, habituellement sans compression (quoiqu'il le supporte). Un BMP est composé d'un entête général qui a toujours la même forme (structure `EnteteBmp` du TD) suivi d'un entête spécifique (structure `EnteteDib` du TD) et d'un tableau de pixels, une ligne après l'autre. Il existe beaucoup de sous-formats de BMP et c'est l'entête DIB qui indique lequel est utilisé. Dans ce TD, nous utilisons le format `BITMAPINFOHEADER` avec des pixels RGB24, qui est un des formats les plus simples, presque identique au format dans le TD4.

Matériel fourni

Dans le fichier *CodeFourni.hpp* vous trouverez les déclarations des structures, constantes et variables qui vous sont fournies; leur implémentation se trouve dans *CodeFourni.cpp*. Vous devez remplir les fonctions dans *CodeDemande.cpp* et mettre vos prototypes dans *CodeDemande.hpp*. Vous n'avez qu'à suivre les *TODO* dans chaque fonction.

On vous donne aussi l'image *image.bmp.* Notez que vous pouvez aussi utiliser n'importe quelle image BMP pour faire vos tests. La dimension de *image.bmp* est de 3440 par 1440.

Il y a plusieurs structures d'enregistrement avec lesquelles vous devez travailler. Les structures `EnteteBmp` et `EnteteDib` sont les entêtes présents dans les fichiers Bitmap que nous utilisons pour le TD. La structure `Pixel` représente un pixel RGB24. La structure `Image` représente une image générale, c'est-à-dire avec une largeur, une hauteur et un tableau de pixels. Les pixels sont organisés par lignes. Ceci signifie que **LES COORDONNÉES DANS LE TABLEAU SONT EN (Y,X) PAS EN (X,Y).** Finalement, la structure `ImageDecomposee` représente une image séparée en plusieurs morceaux (plusieurs images).

```
struct Pixel
{
    uint8_t b;
    uint8_t g;
    uint8_t r;
};

struct Image
{
    unsigned largeur;
    unsigned hauteur;
    Pixel**  pixels;
};

struct ImageDecomposee
{
    unsigned nMorceauxX;
    unsigned nMorceauxY;
    Image**  morceaux;
};
```

Note : Les types `int`***N***`_t` et `uint`***N***`_t` sont des entiers signés et non signés, respectivement, de *N* bits. Ce sont des alias de types standards qu'on retrouve dans `<cstdint>`. Nous les utilisons quand nous voulons spécifier la taille exacte des entiers (8, 16, 32 ou 64 bits).

Présentation du travail à réaliser

Tout d'abord, regardez bien les fonctions fournies, car vous aurez à vous servir de quelques-unes d'entre elles. On vous fournit entre autres des fonctions pour construire des entêtes à partir d'une `Image` avec les options spécifiques à ce TD. Les fonctions les plus importantes (celles que vous aurez à utiliser) sont `calculerTaillePadding()`, `construireEnteteBmp()` et `construireEnteteDib()`.

Vous avez 11 fonctions à implémenter, certaines plus courtes que d'autres. On vous donne les squelettes dans *CodeDemande.cpp* et vous devez ajouter les prototypes dans *CodeDemande.hpp*. Pour chaque fonction, vous devez remplir les *TODO* qui vous sont donnés dans le code. Vous pouvez écrire plus de fonctions si vous le voulez, mais ce n'est pas nécessaire. Le `main` est aussi fourni et vous permettra de tester vos fonctions.

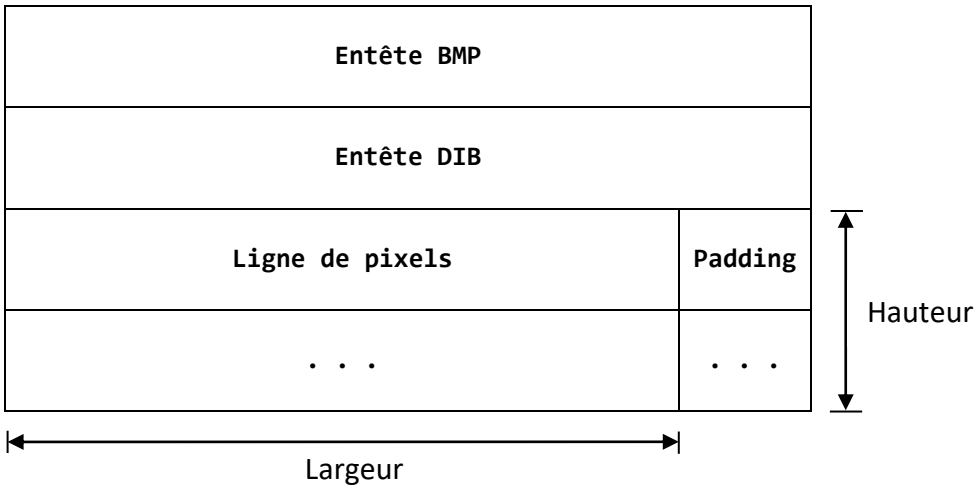
Travail à réaliser

Fonction `lireEnteteFichier()`

Prend en paramètre un fichier valide ouvert en lecture binaire et retourne un `EnteteDib`.

Lit les entêtes d'un fichier bitmap et en extrait l'entête DIB. Vous n'avez pas à vérifier que le format du BMP correspond à celui du TD ou que l'entête contient des données qui ont du sens. Assumez que tous les fichiers qui vous seront donnés sont entièrement valides et compatibles avec l'énoncé.

Rappel: Un fichier BMP est composé d'un entête général (`EnteteBmp`) suivi d'un entête DIB (`EnteteDib`) et d'un tableau de pixels. Le tableau est fait d'une séquence de *hauteur* lignes. Chaque ligne est faite de *largeur* pixels contiguës suivis d'un padding d'octets vides dont le nombre est donné par `calculerTaillePadding()`.



Fonction `lireDonneesImage()`

Prend en paramètre un fichier valide ouvert en lecture binaire et une `Image` déjà allouée.

Lit les pixels d'une image à partir d'un fichier. L'objet `Image` en paramètre donne les dimensions de l'image dans le fichier et doit déjà avoir un tableau alloué avec une taille suffisante.

IMPORTANT : Dans un fichier BMP, les pixels de l'image sont placés un à la suite de l'autre, ligne par ligne, de bas en haut (contrairement à beaucoup d'autres formats) et de gauche à droite. Ceci signifie que l'origine (0,0) est située en bas à gauche de l'image. Le premier indice de `Image::pixels` accède donc à une ligne et le deuxième indice à une colonne de cette ligne. On le rappelle, **les coordonnées dans le tableau sont en (Y,X)**. Chaque ligne doit être alignée sur 4 octets, c'est-à-dire que chaque ligne dans le fichier doit avoir une taille qui est un multiple de 4. Si la taille de la ligne d'une image n'est pas un multiple de 4, il faut ajouter du *padding*, ou *rembourrage*, à la fin de la ligne quand on l'écrit ou lit dans le fichier. Pour savoir quelle est la taille du padding des lignes d'une image, utiliser la fonction `calculerTaillePadding()`. Le padding n'est présent que dans le fichier, pas dans les objets `Image`. Il n'y a donc pas de padding dans `Image::pixels`.

Vous devez donc, pour chaque ligne, lire les pixels et les mettre dans le tableau, puis sauter le nombre d'octets correspondant au padding.

Fonction `ecrireDonneesImage()`

Prend en paramètre un fichier valide ouvert en écriture binaire et une `Image`.

Écrit les pixels d'une image dans un fichier contenant déjà les entêtes. L'objet `Image` en paramètre donne les dimensions de l'image et les pixels à écrire.

Il faut d'abord se positionner juste après les entêtes et, pour chaque ligne, écrire les pixels dans le fichier, puis écrire des octets vides (=0) selon le padding nécessaire. Une fois de plus, utiliser la fonction `calculerTaillePadding()` pour déterminer le padding.

Fonction `ecrireImage()`

Prend en paramètre un nom de fichier, une `Image` et un booléen de sortie (paramètre OUT)

Crée un fichier bitmap à partir d'une `Image`. Un BMP contient un entête BMP, un entête DIB et un tableau de lignes de pixels. Le paramètre booléen indique la réussite ou non de l'ouverture du fichier, `true` signifie une ouverture réussie.

Il faut ouvrir le fichier en écriture binaire, vérifier son ouverture et l'indiquer à l'aide du paramètre de sortie. Si le fichier est correctement ouvert, il faut commencer par écrire les deux entêtes. Pour générer ceux-ci à partir de l'`Image` donnée, utilisez les fonctions `construireEnteteBmp()` et `construireEnteteDib()`. Il faut ensuite écrire les pixels.

Fonction `allouerImage()`

Prend en paramètre une largeur et une hauteur et retourne une `Image` allouée en conséquence ou une `Image` vide en cas d'échec.

Alloue le tableau de pixels pour une image donnée. Si les dimensions ne sont pas valides (égales à 0), L'image retournée est vide.

N'oubliez pas qu'il faut d'abord allouer un tableau de pointeurs correspondant au nombre de lignes, chaque pointeur référant une ligne de la largeur de l'image.

Fonction `desallouerImage()`

Prend en paramètre une `Image`.

Désalloue le tableau de pixels pour une image donnée. Le membre `Image::pixels` est remis à zéro (pointeur nul).

N'oubliez pas qu'il faut d'abord désallouer chaque ligne, puis désallouer le tableau de lignes.

Fonction `lireImage()`

Prend en paramètre un nom de fichier et un booléen de sortie (paramètre OUT) et retourne une `Image`.

Lit un fichier bitmap et crée une `Image` contenant les dimensions et les pixels. Le paramètre booléen indique la réussite ou non de l'ouverture du fichier, `true` signifie une ouverture réussie.

Il faut lire les entêtes du fichier, allouer une `Image` selon les dimensions indiquées et lire les données (pixels).

Fonction `extraireMorceau()`

Prend en paramètre une `Image`, des coordonnées du coin supérieur gauche du morceau et ses dimensions et retourne une `Image`.

Extrait un morceau rectangulaire d'une image en créant une plus petite image qui contient les pixels extraits de la zone donnée.

Fonction `decomposerImage()`

Prend en paramètre une `Image`, et les dimensions des morceaux et retourne une `ImageDecomposee`.

Decoupe une image en la séparant en plusieurs morceaux de même dimension. Vous devez vous servir de `extraireMorceau()` pour obtenir chaque morceau de l'image.

Voici un exemple de son fonctionnement :

Image originale de 1000 x 1000

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Image décomposée en morceaux de 250 x 250

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Fonction `melangerImage()`

Prend en paramètre une `ImageDecomposee` et retourne une `ImageDecomposee`.

Mélange les morceaux en échangeant chaque morceau par un morceau choisi aléatoirement.

Voici un exemple de son fonctionnement :

Image décomposée

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Image décomposée mélangée

N	J	H	C
A	D	P	L
F	O	M	K
G	E	I	B

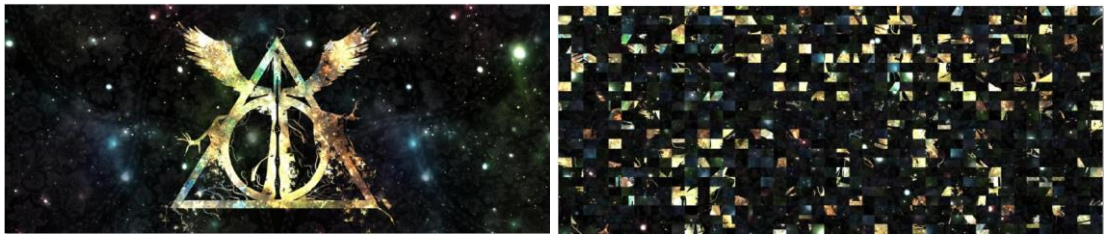
Fonction `recomposerImage()`

Prend en paramètre une `ImageDecomposee`, et retourne une `Image`.

Assemble les morceaux pour recréer une image. Les morceaux ne doivent pas changer d'ordre.

Fonction `main()`

Voici un exemple d'image avant et après le traitement dans le main. Les dimensions entrées pour les morceaux sont 80 x 60.



Annexe 1 : Points du guide de codage à respecter

Les points du **guide de codage** à respecter **impérativement** pour ce TD sont les mêmes que pour le TD4 : (voir le guide de codage sur le site Moodle du cours pour la description détaillée de chacun de ces points)

Points du TD5 :

2 : noms des types en UpperCamelCase
3 : noms des variables en lowerCamelCase
5 : noms des fonctions en lowerCamelCase
21 : pluriel pour les tableaux (`int nombres[];`)
22 : préfixe *n* pour désigner un nombre d'objets (`int nElements;`)
24 : variables d'itération `i`, `j`, `k` mais jamais `l`
27 : éviter les abréviations (les acronymes communs doivent être gardés en acronymes)
29 : éviter la négation dans les noms
33 : entête de fichier
42 : `#include` au début
46 : initialiser à la déclaration
47 : pas plus d'une signification par variable
48 : aucune variable globale (les constantes globales sont tout à fait permises)
50 : mettre le & près du type
51 : test de 0 explicite (`if (nombre != 0)`)
52, 14 : variables vivantes le moins longtemps possible
53-54 : boucles `for` et `while`
58-61 : instructions conditionnelles
62 : pas de nombres magiques dans le code
67-78, 88 : indentation du code et commentaires
83-84 : aligner les variables lors des déclarations ainsi que les énoncés
85 : mieux écrire du code incompréhensible plutôt qu'y ajouter des commentaires
89 : entêtes de fonctions; y indiquer clairement les paramètres [out] et [in,out]