**Lab 2: Traitement d'images simple**

Ce laboratoire a pour but:

* d'appliquer concrètement les fonctions implémentées dans le *Lab 1*
* d'aborder quelques notions d'algorithmie supplémentaires simples

Le programme prend en entrée des images au format *BMP* et en enregistre diverses copies modifiées par des fonctions de traitement d'images simples.

**Fourni**

La fonction main est déjà fournie et ne peut être modifiée. En résumé, elle:

* charge en mémoire les images sources
* appelle diverses fonctions de traitement d'images simples
* persiste les images résultantes

La logique permettant la lecture et l'écriture des fichiers image au format *BMP* vous est déjà fournie dans les fichiers *bmp.h* et *bmp.cpp*.

**Important:** l'indexation des pixels commence à l'indice 0. Elle se fait de gauche à droite et de haut en bas. La valeur est codée sur 8-bits, du noir (0) au blanc (255).

**A faire**

Il s'agit d'implémenter l'ensemble des fonctions de traitement d'images appelées par le main. Pour ce faire l'entête *improc.h* et le fichier source *improc.cpp* seront utilisés.

Nous vous conseillons de suivre les étapes suivantes:

1. Définir les **prototypes de fonction** en analysant les appels effectués
2. Faire quelques analyse, sous la forme de **pseudo-code**, des différentes fonctions
   * Nous montrer si nécessaire vos réflexions
3. Implémenter les fonctions
4. Gérer les différentes **erreurs** qui pourraient survenir

Les spécifications de ces fonctions de traitement d'images peuvent être déduites des images résultantes qui sont attendues.

En plus des fonction de traitement d'image "classiques", il vous est également demandé d'implémenter une fonction simulant un nombre *n* de cycle(s) du [jeu de la vie](https://fr.wikipedia.org/wiki/Jeu_de_la_vie). L'image source chargée pour tester cette fonction un *canon à planeurs de Gosper* qui est une structure périodique bien particulière qui émet des planeurs à intervalle régulier.

**Modalités**

* Travail **individuel**
* **Entraide autorisée** dans la classe
* **Plagiat formellement interdit**, nous analysons vos codes source!

Bonne chance!

Use the following files:

bmp.h

/\*

\* Author: Romain Maffina

\* Date: 11.02.2016

\*

\* BMP image read/write

\*/

//HIDE

#pragma once

#include <cstdint>

#include <string>

#include <vector>

#define CLR\_DEPTH 8

#define CLR\_USED ( 1 << (CLR\_DEPTH)) /\* 256 \*/

// tell the compiler to \*not\* do any memory alignment

#pragma pack(push, 1)

struct bmp\_file\_header\_t {

uint16\_t bfType;

uint32\_t bfSize;

uint16\_t bfReserved1;

uint16\_t bfReserved2;

uint32\_t bfOffBits;

};

struct bmp\_info\_header\_t

{

uint32\_t biSize; // size of the structure

int32\_t biWidth; // image width

int32\_t biHeight; // image height

uint16\_t biPlanes; // bitplanes

uint16\_t biBitCount; // resolution

uint32\_t biCompression; // compression

uint32\_t biSizeImage; // size of the image

int32\_t biXPelsPerMeter; // pixels per meter X

int32\_t biYPelsPerMeter; // pixels per meter Y

uint32\_t biClrUsed; // colors used

uint32\_t biClrImportant; // important colors

};

struct bmp\_color\_index\_t{

uint8\_t r,g,b,pad;

};

// back to normal packing

#pragma pack(pop)

int bmp\_read(const std::string path,

std::vector<uint8\_t> &pixels, uint32\_t &w, uint32\_t &h);

int bmp\_write(const std::string path,

const std::vector<uint8\_t> &pixels, uint32\_t w, uint32\_t h);

main.cpp

/\*

\* Author: Romain Maffina

\* Date: 11.02.2016

\*

\* INF2 - Lab 2

\* Some simple image processing

\*/

#include "bmp.h"

#include "improc.h"

#include <cstdlib>

#include <iostream>

#include <vector>

using namespace std;

//OUT bw.bmp lines.bmp checker.bmp castle2.bmp life2.bmp

int main(void) {

string input("castle.bmp");

string output\_bw("bw.bmp");

string output\_lines("lines.bmp");

string output\_checker("checker.bmp");

string output\_orig("castle2.bmp");

string input\_life("life.bmp");

string output\_life("life2.bmp");

vector<uint8\_t> img\_orig, img\_bw, img\_lines, img\_checker, img\_life;

uint32\_t w, h, w\_life, h\_life;

/\* input \*/

cout << "Reading input BMP image files..." << endl;

if ( !bmp\_read(input, img\_orig, w, h) ) {

cerr << "Failed to read '" << input << "' image!" << endl;

return EXIT\_FAILURE;

}

if ( !bmp\_read(input\_life, img\_life, w\_life, h\_life) ) {

cerr << "Failed to read '" << input\_life << "' image!" << endl;

return EXIT\_FAILURE;

}

cout << "Doing some image processing..." << endl;

/\* black and white \*/

img\_bw = img\_orig;

invert(img\_bw, w, h);

binary(img\_bw, w, h, 128);

/\* lines \*/

img\_lines = img\_orig;

// cross in the middle

draw\_diag(img\_lines, w, h, w/2, h/2, -1, 255);

draw\_diag(img\_lines, w, h, w/2, h/2, 1, 255);

// rule of thirds

draw\_hline(img\_lines, w, h, (h/3), 1, 0);

draw\_hline(img\_lines, w, h, (h/3)\*2, 1, 0);

draw\_vline(img\_lines, w, h, (w/3), 1, 0);

draw\_vline(img\_lines, w, h, (w/3)\*2, 1, 0);

// picture frame

draw\_hline(img\_lines, w, h, 0, 10, 128);

draw\_vline(img\_lines, w, h, w-10, 10, 128);

draw\_hline(img\_lines, w, h, h-10, 10, 128);

draw\_vline(img\_lines, w, h, 0, 10, 128);

/\* black/transparent checker \*/

img\_checker = img\_orig;

checker(img\_checker, w, h, 100);

/\* error checking \*/

draw\_diag(img\_orig, w, h, w, h, -1, 255);

draw\_hline(img\_orig, w, h, h-1, 2, 0);

draw\_vline(img\_orig, w, h, w, 1, 0);

checker(img\_orig, w, h, 0);

/\* life game - Gosper glider gun \*/

life\_game(img\_life, w\_life, h\_life, 200);

/\* output \*/

cout << "Writing back resulting BMP image files..." << endl;

if ( !bmp\_write(output\_bw, img\_bw, w, h) ) {

cerr << "Failed to write '" << output\_bw << "' image!" << endl;

return EXIT\_FAILURE;

}

if ( !bmp\_write(output\_lines, img\_lines, w, h) ) {

cerr << "Failed to write '" << output\_lines << "' image!" << endl;

return EXIT\_FAILURE;

}

if ( !bmp\_write(output\_checker, img\_checker, w, h) ) {

cerr << "Failed to write '" << output\_checker << "' image!" << endl;

return EXIT\_FAILURE;

}

if ( !bmp\_write(output\_orig, img\_orig, w, h) ) {

cerr << "Failed to write '" << output\_orig << "' image!" << endl;

return EXIT\_FAILURE;

}

if ( !bmp\_write(output\_life, img\_life, w\_life, h\_life) ) {

cerr << "Failed to write '" << output\_life << "' image!" << endl;

return EXIT\_FAILURE;

}

return EXIT\_SUCCESS;

}