

TD Outils 1 : Une classe Image

Une image est un tableau à 2 dimensions de largeur dim_x et de hauteur dim_y , dont les éléments sont des pixels. En interne, nous allons allouer un tableau 1D de taille $dim_x * dim_y$ car l'allocateur mémoire du C++ ne sait allouer que des tableaux à une dimension.

En externe (ie. le code utilisant le module Image), l'utilisateur de la classe Image manipulera l'image avec des fonctions de haut niveau. Le fait que le tableau soit 1D doit rester complètement caché. Les fonctions de haut niveau peuvent être toute sorte de fonctions qui transforment une image : seuillage ; changement de palette de couleur ; dessin de ligne, droit, cercle, rectangle ; etc. Nous nous limiterons à quelques fonctions, car l'objectif est d'apprendre cette subtilité entre représentation interne et affichage externe. Il ne s'agit pas de coder une classe Image complète. Pour faciliter l'écriture des fonctions de haut niveau, il est bien pratique d'offrir plusieurs fonctions d'accès au pixel avec deux dimensions (x,y) en passant par des fonctions `getPix` et `setPix`. Les définitions de classes vous sont données ci-dessous en langage algorithmique.

N'oubliez pas de compiler AU FUR ET À MESURE. N'attendez pas d'avoir écrit toutes les fonctions membres pour compiler. Ne modifiez pas les noms des classes, membres, paramètres, etc.

1. Pour gagner du temps plus tard, créez des répertoires
 - LIFAPCD_Image
 - LIFAPCD_Image/src
 - LIFAPCD_Image/bin
2. Créez le fichier **Pixel.h** dans le répertoire src. Traduisez et implémentez la structure en C++. Ici Pixel est un cas particulier : son type est tellement simple que la notion de données privées ou publiques perd un peu de son sens. Les données d'un pixel peuvent être changées par tous les codes sans compromettre la sécurité du code. Un constructeur qui initialise les données est néanmoins bienvenu.

Pixel.h doit inclure des directives de non-inclusion multiple

```
#ifndef _PIXEL_H
#define _PIXEL_H

...
#endif
```

```
struct Pixel
    r,g,b : entier(0..255)    // les composantes du pixel, unsigned char en C++

    // Constructeur par défaut de la classe: initialise le pixel à la couleur noire
    Constructeur Pixel ()

    // Constructeur de la classe: initialise r,g,b avec les paramètres
    Constructeur Pixel (nr, ng, nb : donnée entier(0..255) )
```

3. Créez un fichier `src/main.cpp` qui inclut `Pixel.h` et qui teste que tout est ok.

Ecrivez également dans le répertoire `LIFAPCD_Image` un `makefile` ou un projet `CMake`. Regardez la page « git » de l'UE pour une explication sur `Cmake`, regardez également les exemples `SDL_simple` : https://forge.univ-lyon1.fr/Alexandre.Meyer/L2_ConceptionDevApp/-/blob/master/doc/

```
int main()
{
    Pixel p1;
    Pixel p2(10,20,30) ;
    ... affichage du contenu des pixels pour être sûr que ok
```

4. Créez les fichiers **Image.h** et **Image.cpp**. Traduisez et implémentez la classe en C++. A gauche, nous vous indiquons dans quel ordre écrire le code, après chaque partie, vous devez compléter votre fonction de test de régression pour tester que tout fonctionne. Il y a des explications plus bas pour certaines parties.

Le fichier Image.h doit inclure les directives de non-inclusion multiple `#ifndef` comme pour Pixel.

```

Classe Image
privé
    tab : tableau de Pixel      // le tableau 1D de pixel
    dimx, dimy : entier        // les dimensions de l'image

public

    // Constructeur de la classe : initialise dimx et dimy (après vérification)
    // puis alloue le tableau de pixel dans le tas (image noire)
    Constructeur Image (dimensionX, dimensionY : donnée entier);

    // Destructeur de la classe : déallocation de la mémoire du tableau de pixels
    // et mise à jour des champs dimx et dimy à 0
    Destructeur Image ()

    // Récupère le pixel original de coordonnées (x,y) en vérifiant sa validité.
    // La formule pour passer d'un tab 2D à un tab 1D est tab[y*dimx+x]
    Fonction getPix (x,y : donnée entier) -> 'Pixel' (l'original, pas une copie)

    // Récupère le pixel original de coordonnées (x,y) en vérifiant sa validité.
    Fonction getPix (x,y : donnée entier) const -> 'Pixel' (une copie du Pixel)

    // Modifie le pixel de coordonnées (x,y)
    Procédure setPix (x,y : donnée entier; couleur : donnée Pixel)

    // Dessine un rectangle plein de la couleur dans l'image
    // (en utilisant setPix, indices en paramètre compris)
    Procédure dessinerRectangle (Xmin,Ymin,Xmax,Ymax : donnée entier; couleur : donnée
Pixel)

    // Efface l'image en la remplissant de la couleur en paramètre
    // (en appelant dessinerRectangle avec le bon rectangle)
    Procédure effacer (couleur : donnée Pixel)

    // Effectue une série de tests vérifiant que toutes les fonctions fonctionnent et
    // font bien ce qu'elles sont censées faire, ainsi que les données membres de
    // l'objet sont conformes
    Procédure « static » testRegression ()

FinClasse
  
```

4.a) Ecrivez le constructeur, destructeur et la fonction test de régression. « static » signifie que la fonction de la classe peut être appelée sans avoir d'instance de la classe. Votre main doit ressembler maintenant à ceci.

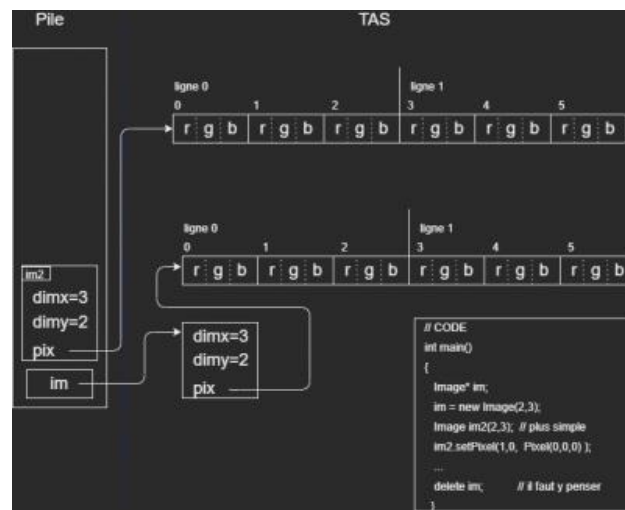
```

#include "Image.h"
int main()
{
    Image::testRegression();
    Return 0.
}
  
```

Testez votre exécutable avec valgrind : outil présenté en détail plus tard, mais c'est le bon moment pour commencer à l'utiliser. Dans un terminal, lancez ceci.

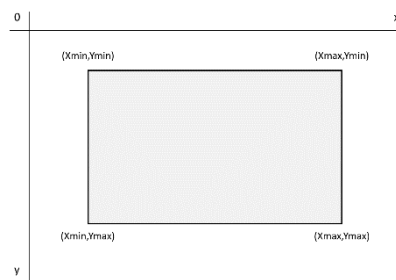
```
/home/alex/LIFAPCD_Image $ valgrind bin/image.out
```

- a) 4.b) Ecrivez les fonctions `getPix` et `setPix`. Il est important de comprendre la différence entre la pile et le tas. De comprendre que l'allocation se fait sur un tableau 1D, car c'est la seule solution qu'offre `new`, mais que de l'extérieur à la classe `Image` les codeurs veulent travailler avec des informations 2D. Les accesseurs font donc la conversion pour gérer les données internes.



Ajoutez des tests à la fonction de test de régression. Recompiler, lancez `valgrind` et recommencez tant que `valgrind` indique des erreurs.

- 4.c) Ecrivez la fonction qui dessine un rectangle. `dessinerRectangle` allume les pixels de l'image qui sont compris dans le rectangle défini par le coin en haut à gauche (`Xmin,Ymin`) jusqu'au point en bas à droite (`Xmax,Ymax`), comme ceci :



Ajoutez des tests à la fonction de test de régression. Recompiler, lancez `valgrind` et recommencez tant que `valgrind` indique des erreurs.