

Module Base de la programmation *C++*

Travaux Pratiques 2

base du langage *C++* : vecteurs et entrées/sorties fichiers

Exercice 2.1 *Entrées/sorties*

Dans cet exercice, nous allons voir comment gérer les entrée-sorties à travers la lecture et l'écriture d'un fichier sur disque. Comme exemple d'application, nous verrons une simple écriture puis lecture de texte dans un fichier. Après avoir fait cet exercice, vous serez capable de :

- Écrire dans un fichier.
- Lire dans un fichier.

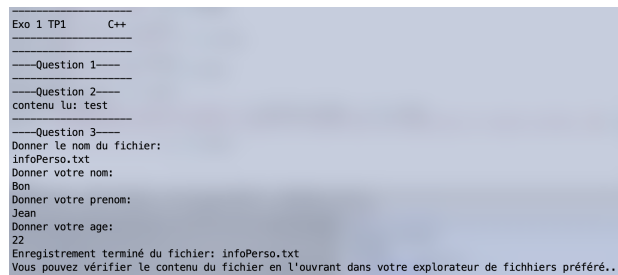
Notions de cours :

De manière comparable à l'affichage dans la sortie standard, il est possible de lire ou écrire dans un fichier. Pour cela il faut utiliser un objet `fstream`. Il s'utilise de la façon suivante :

```
#include <iostream>
#include <fstream>
...
std::ofstream fichierSortie;
// Ouverture de fichier en mode ecriture:
fichierSortie.open("unFichier.txt", std::fstream::out);
fichierSortie << "test_ca_marche_?" << std::endl;
fichierSortie.close();

std::ifstream fichierIn;
std::string contenu = "";
// Ouverture de fichier en mode lecture:
fichierEntree.open("unFichier.txt", fstream::in);
fichierEntree >> contenu;
```

1. Testez l'exemple d'écriture dans un fichier et vérifiez que votre programme vous génère bien un nouveau fichier au même endroit que votre exécutable.
2. De la même façon vérifiez la lecture du premier mot d'un fichier.
3. On souhaite ensuite pouvoir sauvegarder le nom, prénom et âge saisis par l'utilisateur. Rajoutez une invite de saisie pour que l'utilisateur puisse entrer le nom du fichier dans lequel il souhaite sauvegarder les informations puis il pourra saisir les informations de son nom, prénom et âge. L'exécution pourra ressembler à l'illustration suivante :



```
Exo 1 TP1      C++
-----
Question 1-----
Question 2-----
contenu lu: test
Question 3-----
Donner le nom du fichier:
infoPerso.txt
Donner votre nom:
Bon
Donner votre prenom:
Jean
Donner votre age:
22
Enregistrement terminé du fichier: infoPerso.txt
Vous pouvez vérifier le contenu du fichier en l'ouvrant dans votre explorateur de fichiers préféré...
```

Pour la suite, il s'agit de pouvoir lire le contenu d'un fichier donné par l'utilisateur.

4. A la suite, demandez à l'utilisateur de saisir le nom d'un fichier qu'il souhaite lire, puis en utilisant un objet `ifstream`, ouvrez le fichier en mode lecture (mode `fstream::in`).
5. Affichez ensuite le contenu du fichier avec une boucle `while` et en utilisant la méthode `get()` qui prend en paramètre une variable de type `char` et qui renvoie vraie si la lecture du prochain caractère est valide ou non.

Exercice 2.2 Génération d'images

Dans cet exercice, il s'agit d'exploiter à nouveau la gestion d'entrée/sortie mais avec pour objectif de générer des images.

1. Générez une image de type PGM représentant le dégradé de l'image (a) de la figure ci-dessous. Le format PGM est le suivant :

```
P2
LARGEUR HAUTEUR
VAL_MAX
ValPixel1 ValPixel2 ...
```



(a)

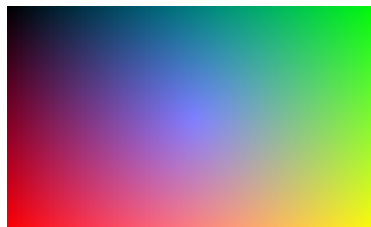


(b)



(c)

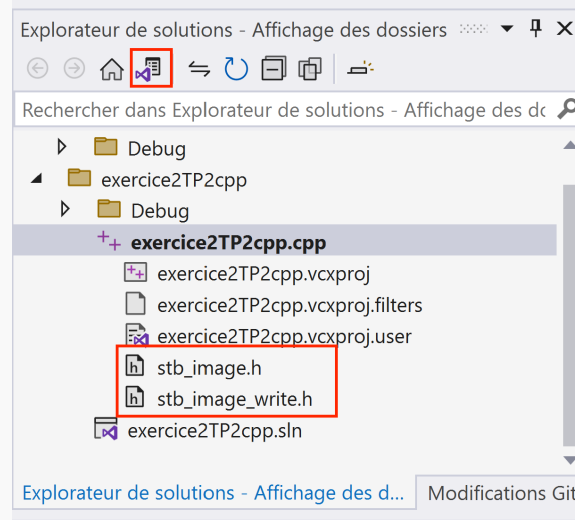
2. Même question que précédemment mais pour obtenir le dégradé de l'image (b).
3. Enfin en modifiant le type d'image en prenant la version couleur (format PPM code P3 au lieu de P2), générez une dernière image semblable à l'image (c).
4. Modifiez votre code pour qu'il puisse demander à l'utilisateur une hauteur et largeur et puisse générer par exemple l'image suivante :



Exercice 2.3 Manipulation d'images : bibliothèque stb

Il existe une bibliothèque populaire qui permet de lire et écrire une image sous différents formats (.png, gif, jpg, ...). Elle est simplement constituée d'un fichier d'entête (header .h) pour la lecture d'images (`stb_image.h`) et d'un autre fichier pour l'écriture (`stb_image_writer.h`).

Pour exploiter une bibliothèque basée uniquement sur les fichiers headers est assez simple à intégrer car il est juste nécessaire que le compilateur puisse trouver leur localisation. Contrairement aux bibliothèque qui nécessite une compilation, il n'y a rien à configurer pour l'étape 5 lié à l'édition de liens de la phase de compilation. Dans l'IDE de *Visual Studio*, vous pouvez rajouter simplement les fichiers comme illustré sur l'image suivante :



L'utilisation de l'image s'effectue de la façon suivante :

```
#include <iostream>

#define STB_IMAGE_IMPLEMENTATION
#define STB_IMAGE_WRITE_IMPLEMENTATION

#include "stb_image.h"
#include "stb_image_write.h"

int
main(){
    int width, height, channels;
    unsigned char *data = stbi_load("image.png", &width, &height, &channels, 0);

    return 0;
}
```

1. Testez la lecture d'image en incluant l'image de tests et les fichiers *header* disponibles sur *moodle*. Affichez les dimensions de l'image tout comme son nombre de canaux.
2. Afin de tester les fonctions d'écriture d'images, utilisez la fonction suivante qui permet l'écriture en format png. La déclaration de la fonction est la suivante :

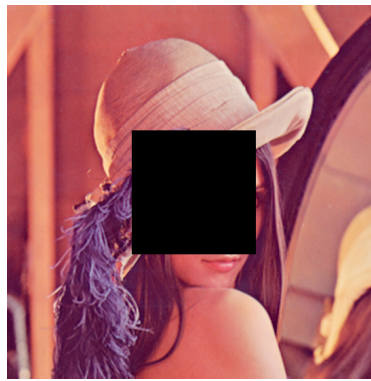
```
int stbi_write_png(char const *filename, int w, int h, int comp, const void *data, int stride_in_bytes)
```

Il s'agira simplement d'écrire la même image sous un nom différent.

3. Dans la suite de votre programme écrivez des instructions de façon à incruster un carré noir au centre de l'image. La zone noire doit s'étendre du premier tiers de l'image jusqu'au second tiers comme sur l'image (b) ci-dessous.



(a)



(b)



(c)

4. Rajoutez du code de façon à construire une nouvelle image à partir de l'image originale et dont tous les pixels présentant une moyenne des niveaux d'intensités sur les quatre canaux compris entre 100 et 150, seront transformés avec la valeur 255. Les autres pixels seront laissés avec leurs valeurs originales. L'image résultante devra être celle de l'image (c) ci-dessus.

Exercice 2.4 *Filtres d'images*

L'idée de cet exercice est d'effectuer un filtre un peu plus avancé à travers le filtre moyenneur. Ce filtre se définit par une transformation ayant comme support un pixel et son voisinage. La nouvelle valeur d'un pixel P transformé est calculée comme la moyenne de ses voisins et de lui-même. Un filtre est souvent représenté par les coefficients utilisés pour le calcul de la moyenne :

$$\begin{array}{ccc} & 1 & 1 & 1 \\ 1/9 & 1 & \boxed{1} & 1 \\ & 1 & 1 & 1 \end{array}$$

1. Appliquez le filtre sur l'image et enregistrez le résultat dans une nouvelle image.
2. Enfin une autre catégorie de filtre existe comme le filtre médian qui consiste à classer les pixels de la fenêtre glissante et de retenir comme nouvelle valeur, la valeur centrale du classement. En enregistrant les valeurs de pixels voisins et la fonction `std::sort`). Cette fonction s'utilise en donnant deux itérateurs. Par exemple, pour filtrer un vecteur `v` contenant une série de nombre il suffit d'écrire : `sort(v.begin(), v.end());`. Implémentez le filtre médian en considérant un voisinage de taille 3x3 et enregistrez le résultat dans une nouvelle image.
3. Reprendre la question précédente pour faire des variantes avec prenant la valeur maximale puis minimale (filtre d'érosion et dilatation).
4. Modifiez votre programme de façon à ce qu'il puisse considérer un voisinage de taille variable. Cette taille sera un paramètre que l'utilisateur pourra choisir de modifier.
5. Tester votre filtre avec une taille de voisinage égale à 5.