Lecture - Écriture dans un fichier

Préalable

Le rendu des TP se fera via GitLab. Avant de commencer le TP il vous faudra donc impérativement suivre les étapes décrites dans le document décrivant les consignes Git.

Sisi, **il faut aller lire les consignes**.

Une fois votre dépôt créé et avant d'ajouter le répertoire pour le premier TP, créez un fichier .gitignore à la racine de votre dépôt contenant *.o.

```
echo "*.o" >> .gitignore
```

Ce fichier `.gitignore` doit donc se trouver à la racine de votre dépôt, valable pour tous les TPs.

Une fois les dépôts Git (local et distant) créés, vous pouvez passer à la suite :

Le répertoire de ce TP dans votre dépôt s'appellera "LectureEcriture".

```
mkdir LectureEcriture
```

Créez ensuite un fichier `.gitignore` dans ce répertoire, contenant le nom des exécutables : `firstPPM`, `gradient` et `filtre`, *un par ligne*.

Un étudiant sera désigné pour chaque groupe de TP et sera en charge de récupérer les URLs des dépôts de tous les binômes et de les transférer à l'encadrant. Vous devez envoyer un mail à cet étudiant avec l'url de votre dépôt sur GitLab ainsi que les urls permettant de le cloner par https et ssh.

Attendu

Vous avez **deux séances** pour réaliser ce TP.

Fichiers attendus:

- `OS_Nom1_Nom2/`
  - `.gitignore` (général `*.o` notamment)
  - `LectureEcriture/`
    - `.gitignore` (avec tout ce qui n'est pas attendu, ligne par ligne)
    - `Makefile`

- `firstPPM.c`
- `gradient.c`
- `darker.c`
- `arguments.c`

Ce qui n'est pas poussé sur votre dépôt GitLab n'est pas corrigé.

Pensez donc bien à effectuer des `commit` et des `push` régulièrement.

Afin de pratiquer les entrées-sorties en C, vous allez manipuler des images au format PPM.

1. Création d'une image - Écriture dans un fichier

La première étape va consister à **créer** une image en **écrivant** dans un fichier.

Le format

Vous allez travailler avec le format PPM qui permet de stocker simplement les valeurs des différents pixels directement sous forme matricielle.

Par exemple, l'image avec 6 pixels suivante:

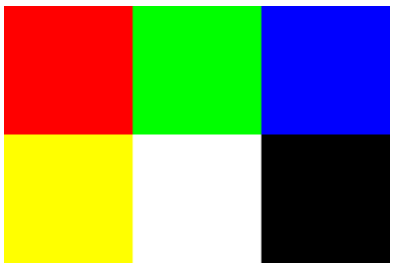

Figure 1. Simple 6 pixels

Donne le fichier PPM :

```
P3                                  ①
3 2                                 ②
255                                 ③
255 0   0  0   255 0    0 0 255     ④
255 255 0  255 255 255  0 0 0
```

① "P3"" pour le format ASCII, RGB.

② 3 pixel de large, 2 pixels de haut

③ Valeur maximal possible pour les composants (rouge, vert ou bleu)

④ Les valeurs des composantes sont ensuite mise les unes à la suite des autres (les retours à la ligne n'ont pas besoin de correspondre au lignes de l'image). Les lignes ne doivent normalement pas dépasser 74 caractères (mais cela semble fonctionner quand même). Tous les caractères

d'espacement (espace, tabulation, `\n`) sont acceptés pour séparer les valeurs.

Il est possible d'ajouter des commentaires dans l'en-tête en commençant la ligne par `#`.

```
# Ceci est un commentaire
```

1.1. Écrire dans un fichier

Avant de pouvoir écrire dans un fichier, il faudra l'ouvrir. Pour cela vous utiliserez la fonction `open`:

```
1 int open(const char *pathname, int flags, mode_t mode);
```

Cette fonction prend en paramètre :

- le nom du fichier (*pathname*) qui est une chaîne de caractères,
- les *flags* d'ouverture qui précisent les options d'ouverture,
- et le *mode* d'ouverture, qui précise les droits d'accès du fichier s'il doit être créé.

`open` retourne un **descripteur de fichier** : c'est entier qui désigne un fichier.

Une fois ce descripteur de fichier obtenu, vous pourrez l'utiliser pour écrire dans le fichier à l'aide de la fonction `write` :

```
1 ssize_t write(int fd, const void buf[.count], size_t count);
```

`write` écrit dans le fichier les "count" premiers octets contenus dans le buffer "buf" et retourne le nombre d'octets effectivement écrit ou -1 en cas d'erreur.

1.1.1. Première image : firstPPM.c

Créez et ajoutez à Git un fichier C nommé `firstPPM.c`.

En partant du code ci-dessous, écrivez dans un fichier nommé "firstPPM.ppm" le texte ci-dessus qui correspond à l'image de la Figure 1.

```
1 #include <fcntl.h>  // For 'open'
2 #include <unistd.h> // For 'write'
3 #include <string.h> // For 'strlen'
4 #include <stdio.h>  // For 'snprintf'
5
6 int main(int argc, char const *argv[])
7 {
8     int fd = open("firstPPM.ppm", O_CREAT|O_TRUNC|O_RDWR, S_IRUSR|S_IWUSR);
9
```

```
10      char* en_tete = "P3\n3 2\n# A completer\n";
11
12      write(fd, en_tete, strlen(en_tete));
13
14      char buf[255];
15      int red, green, blue;
16
17      red = 255;
18      green = 0;
19      blue = 0;
20
21      snprintf(buf, 255, "%d %d %d ", red, green, blue); // put ints in buf
22      write(fd, buf, strlen(buf)); // Write buf to file
23
24      // TODO: A completer
25
26      write(fd, "\n", 1);
27
28      close(fd); // close the file to free resources
29
30      return 0;
31  }
```

Ce programme compile et peut être exécuté mais ne donne pas le résultat attendu.

1.1.2. Makefile

Créez et ajoutez à Git un fichier Makefile.

Pour la compilation vous **devez** utiliser un *Makefile*. Un support de cours sur Make est disponible sur ce site.

1.1.3. Vérification

Une fois le Makefile en place le programme compilé puis exécuté, vous pouvez ouvrir le fichier produit, `firstPPM.ppm`, avec un **éditeur de texte** pour en examiner le contenu.

Vérifiez que ce qui est obtenu correspond partiellement à ce qui est attendu.

Complétez ensuite le programme pour obtenir le fichier attendu.

2. Création d'un dégradé

> Vous passerez à un fichier ***gradient.c*** pour cette question.

N'oubliez pas de l'ajouter à votre Git et à votre Makefile.

L'objectif de cette partie est de produire le fichier PPM correspondant à l'image ci-dessous :

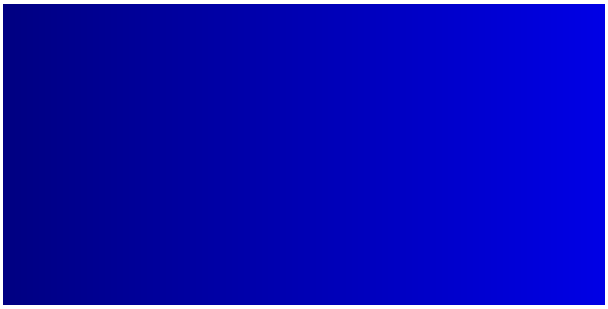

Figure 2. Dégradé de bleu

L'image écrite dans le fichier devra faire au moins 200 pixels de large et 100 de haut.

> Pour observer vos images, vous utiliserez un logiciel d'affichage d'image (gimp, eog, gwenview,...) et non plus un éditeur de texte.

Si vous obtenez quelque comme l'image ci-dessous, vous avez probablement inversé l'ordre de parcours lignes/colonnes.

Figure 3. Dégradé de bleu raté

3. Copie d'une image

> Vous passerez à un fichier ***copy.c*** pour cette question.

N'oubliez pas de l'ajouter à votre Git et à votre Makefile.

L'objectif de cet exercice est de copier une image, octet par octet. Pour cela vous allez devoir lire les pixels d'une image et les écrire au fur et à mesure dans une nouvelle image. Il n'est pas demandé de stocker les pixels en mémoire.

Pour ouvrir l'image et lire les entiers correspondants aux couleurs, il sera (beaucoup) plus aisé d'utiliser les fonctions utilisant les flux (`FILE*`).

```
1 FILE* fopen(const char* pathname, const char* mode);
```

Pour ouvrir puis lire un entier depuis un fichier, il vous faudra faire quelque chose comme cela:

```
1 // Open file for reading
```

```
2 FILE* f = fopen("Chaton.ppm", "r");
3
4 // Read a line from the file
5 char line[200];
6 fgets(line, 200, f);
7
8 // Read an int from the line
9 int entier;
10 sscanf(line, "%d", &entier);
```

En utilisant le code de la question précédente pour écrire dans un nouveau fichier et le code ci-dessus, faites en sorte de copier l'image "Chaton.ppm" (sauvegardez cette image dans le répertoire de votre TP) dans un fichier nommé "ChatonCopy.ppm".

4. Modification d'une image - Assombrir

> Vous passerez à un fichier ***darker.c*** pour cette question.

N'oubliez pas de l'ajouter à votre Git et à votre Makefile.

L'objectif ici est de modifier chaque valeur de couleur avant de l'enregistrer dans un nouveau fichier.

À partir de votre programme précédent, faites en sorte que chaque valeur de couleur soit plus sombre. Vous écrirez l'image assombrie dans un fichier "DarkChaton.ppm".

Vous devriez obtenir quelque chose comme ça :

Figure 4. Un chaton assombri

5. Récupérer les arguments de votre programme

> Vous passerez à un fichier ***arguments.c*** pour cette question.

Afin de rendre plus simple l'utilisation votre programme, vous allez faire en sorte que les noms des fichiers à lire et écrire soient passés en paramètres de votre programme.

La liste des arguments passés en paramètre à un programme est fournie à votre `main` dans le

paramètre `argv`.

Pour récupérer le premier argument du programme dans une chaîne vous ferez donc :

```
int main(int argc, char* argv[])
{
   char* inputFilename;
   char* ouputFilename;
   if (argc < 3){
      fprintf(stderr, "You should provide an input and an output filenames.
Aborting");
      exit(-1);
   }

   inputFilename = argv[1];
   outputFilename = argv[2];

}
```

6. Sauvegarde au format binaire

- `binaryPPM.c`

Le format d'image PPM permet de sauver les images au format binaire. Pour cela, il faudra indiquer 'P6' à la première ligne de l'en-tête. Les dimensions et la valeur maximale sont écrites en ascii comme pour le format P3.

En revanche, les valeurs des pixels ne sont plus écrites en ascii mais en binaire. Par exemple, vous n'allez plus écrire : `200 150 89` pour un pixel dans le fichier mais 3 octets contenant la représentation binaire des 3 valeurs. Il n'y a plus non plus d'espace à écrire dans le fichier.

Copiez une image en la sauvegardant au format binaire. Comparez les tailles des fichiers.

7. Expert

7.1. Autres modifications - bonus

- `tuner.c`

Testez d'autres modifications des couleurs de l'image, par exemple:

- Éclaircissement de l'image,
- Intensification du vert uniquement,
- Passage en noir et blanc
- Etc.

Faites ces modifications dans des nouveaux fichiers voir, idéalement, prenez en compte un

troisième paramètre qui précise la modification à faire.