TP Programmation en C : introduction au traitement d'images

Quelques consignes pour débuter :

1. *Les commandes UNIX et les mécanismes nécessaires à ces manipulations ont été rappelés dans la documentation fournie en préliminaire aux TP et sont rappelés ici, en PARTIE 9.*
2. *Certaines commandes dépendent du shell utilisé et peuvent être différentes d'un shell à l'autre (`sh, bash, tcsh, csh...`). Pour obtenir des précisions* ***CONSULTEZ le manuel en ligne*** *(**`man commande`** `ou` **`man fonction`**)*
3. *Plusieurs fonctions ou modules développés dans le cadre de ce TP seront réutilisés pendant le projet. Ne négligez pas la qualité des programmes que vous développez (réutilisabilité, clarté et lisibilité, validité, ...). D'autres étudiants auront à s'en servir.*
4. *N'oubliez pas de travailler sous le répertoire sauvegardé sur le serveur dont le chemin est de la forme :* **`/home/abc1234a/Documents`**
5. *Chaque TP devra être réalisé dans un répertoire spécifique, ici :* ***`VOTRE_NOM_TP_IMAGE`***
6. *Vous pouvez commenter vos programmes et ajouter en fin de fichier, les réponses aux différentes questions intermédiaires ou rendre un fichier de compte-rendu séparément, au choix.*

ATTENTION : lors de l'examen de TP portant sur le « TP Image », il vous sera demandé de réutiliser votre code. Vous devez donc garder la trace de la totalité de votre code et des tests correspondants. Ne JAMAIS écraser le fichier source correspondant à une question, lorsque vous passez à une autre question.

PARTIE 1- Editer, compiler et tester un programme

1) Exécuter la commande **`date`**

2) Se placer dans le répertoire qui sera sauvegardé (cf. point 4 ci-dessus)

3) Créer le répertoire **`TPC`**, puis le sous-répertoire nommé ***`VOTRE_NOM`*`_TP_IMAGE`**

4) Se déplacer dans ***`VOTRE_NOM`*`_TP_IMAGE`** et créer les sous-répertoires **`Test`** et **`Data`**

5) Vérifier que vous avez bien l'arborescence

```
TPC
      VOTRE_NOM_TP_IMAGE
            Data
            Test
```

6) Se positionner dans le sous-répertoire **`Data`**

7) Y copier les fichiers **`image1.dat`** et **`image2.dat`** qui se trouvent sous `moodle` et qui contiennent les exemples de données à traiter. Visualiser le contenu de ces fichiers.

Exemple de contenu de fichier à traiter

```
cat image1.dat
8  8
0  0    0    0    0    0    0    0
0  32   32   0    0    150  150  0
0  32   32   0    150  150  0    0
0  0    0    0    0    150  150  0
0  90   0    0    0    174  0    0
0  90   0    0    0    174  0    0
0  90   90   0    174  174  174  0
0  0    0    0    0    0    0    0
```

Ces fichiers contiennent sur la première ligne le **nombre de lignes** suivi du **nombre de colonnes** de la matrice et sur les lignes suivantes, les lignes de la **matrice représentant l'image** (niveaux de gris des pixels de l'image). Vous vous référerez à l'annexe (section 8) pour les détails concernant les données représentées (image → matrice).

SIMPLIFICATION : Les traitements effectués dans les premiers exercices peuvent être faits « au fil de l'eau », c'est-à-dire que les valeurs des pixels sont lues une à une et traitées au fur et à mesure. Il n'est donc pas besoin, pour le moment, de mémoriser la matrice dans un tableau.

8) Vous placer dans le répertoire ***VOTRE_NOM*_TP_IMAGE**

NOTEBOOK : Faire un résumé des commandes réalisées en copier-collant le résultat de la commande **`history`** dans le notebook qui sert de compte-rendu.

PROGRAMME transform_image.c

8) Ecrire le programme **`transform_image.c`** qui lit un flot d'entiers correspondant à une image (ex : **`image1.dat`**) et effectue une transformation de celle-ci en la ramenant à 4 niveaux de gris (cf. annexe). L'image transformée est affichée à l'écran en respectant le format "matriciel initial".

9) Compiler votre programme de façon :

a) à créer l'exécutable **`transform_image.out`** (option **`-o`** de **`cc`** ou **`gcc`**)

b) à récupérer les messages d'erreur dans un fichier **`erreur_compil`** (mécanisme de redirection de la sortie standard erreur **`>&`**)

10) Tester votre programme sur les fichiers de données récupérés en début de TP.

ATTENTION : la manipulation directe des fichiers par un programme C n'ayant pas encore été vue, vous communiquerez les données à traiter en procédant à une **redirection de l'entrée standard** (commande **< fichier**) du programme lors de son exécution. De même pour créer un fichier résultat (image codée) vous utiliserez le mécanisme de redirection de la sortie standard (commande **> fichier**). Dans le premier cas, vous fournirez le nom du fichier **.dat** (ou chemin d'accès à ce fichier) dont le contenu devra être traité et dans le second cas, le nom du fichier qui contiendra l'image résultat. Ces mécanismes de redirection ont été vus lors du TP UNIX.

Exemple d'appel du programme avec redirection de l'entrée standard :

```
transform_image.out < ./Data/image1.dat
```

Exemple d'appel du programme avec redirection de l'entrée standard ET de la sortie standard :

```
transform_image.out < ./Data/image1.dat > ./Data/image1_4N.dat
```

11) En utilisant le programme **`transform_image.out`**, générer les fichiers **`image1_4N.dat`** et **`image2_4N.dat`** à partir des fichiers **`image1.dat`** et **`image2.dat`**. Ces fichiers seront rangés dans le répertoire **`Data`**.

12) Avant de passer à la suite, faire valider les résultats par l'enseignant(e).

NOTEBOOK : une fois votre programme mis au point, copier-coller (a) le résultat de la commande date (b) le code source correspondant (c) la commande de compilation et son résultat (d) le résultat des tests (e) le contenu du répertoire TPC (fichiers et sous-répertoires)

PARTIE 2- Visualisation du résultat (optionnel)

13) Afin de visualiser les résultats vous pouvez procéder à la petite manipulation suivante à partir du logiciel **octave** disponible sur la machine :

```
% pour chaque image à visualiser, créer un ficher temporaire ne contenant que la matrice
% (fichier .dat sans la première ligne)
tail +2 mon_image.dat > mon_image_seule.dat
(ou bien suivant les versions de UNIX ou du shell utilisé)
tail --lines=+2 mon_image.dat > mon_image_seule.dat

% Appel du logiciel de calcul numérique (équivalent de Matlab)
octave
octave : 1> % sous octave
  % Charger les données (fichiers ne contenant QUE la matrice)
  % Charge le fichier indiqué dans la variable mentionnée
  im1=load('nom_fichier_image1.dat');
  im2=load('nom_fichier_image_transformee.dat');

  % Première figure
  % Ouvre une fenêtre
  figure;
  % Affiche sous forme d'image la variable im1
  image(im1);
  % Deuxième figure (idem)
  figure;
  % Affiche sous forme d'image la variable im2
  image(im2);
  imagesc(im2) ; % même chose avec une légère différence : laquelle ?
  % quitter octave après avoir fermé les fenêtres (images)
  quit
> % retour sous le shell
```

NOTEBOOK : sauvegarder les images générées et les insérer dans le notebook

PARTIE 3- Généraliser un traitement, traiter les cas d'erreur

Objectifs : Proposer une version plus générique du programme précédent.

PROGRAMME transform_image_gen.c

14) A partir d'une copie du programme précédent, que vous appellerez **transform_image_gen.c,** modifier le code pour

a) prendre en compte toute valeur du niveau de codage inférieure à 256.

b) vérifier que la valeur du niveau est aussi une puissance de deux.

Dans le cas contraire, un message d'erreur sera affiché et le traitement ne sera pas effectué. Vous distinguerez ainsi deux cas d'erreur :

```
Cas a) « Erreur 1 : le niveau de gris est supérieur à 256 »
Cas b) « Erreur 2 : le niveau de gris n'est pas une puissance de 2 »
```

ATTENTION :

1. Pour utiliser les fonctions **log2** ou **log,** il faut inclure la bibliothèque de fonctions mathématiques standard grâce à la directive placée dans votre code C :

```
#include <math.h>
```

et aussi compiler avec l'option **-lm** : **gcc xx.c -o xx.out -lm**

2. Une fois l'entrée standard redirigée (**<**), la saisie au clavier n'est plus possible. Le niveau de gris demandé sera donc fourni par le biais d'un fichier appelé **niveau** et qui contiendra uniquement le niveau requis. Ce fichier concaténé (commande unix **cat**) au fichier image (**.dat**) sera envoyé en entrée du processus correspondant au programme (mécanisme de communication de processus basé sur l'utilisation du « pipe » ou **|**) . Comme précédemment, le résultat de la transformation sera mémorisé dans un fichier.

```
/* création du fichier niveau contenant l'entier 16 */
 cat > niveau
 16
 ^D   (touche ctrl D)
```

Exemple d'appel prenant en compte le niveau de gris :

```
cat niveau image1.dat | transform_image.out > image1_gen.dat
```

15) Compiler et tester votre programme avec les images récupérées ainsi qu'avec les niveaux : **0, 2, 4, 15, 32, 260**. Les résultats de tests seront récupérés dans les fichiers respectifs **./Test/test_0.res, ./Test/test_2.res, … ./Test/test_260.res**

16) Se placer dans le répertoire **TPC** et déposer sous moodle l'archive ***VOTRE_NOM*_TP_IMAGE.tar** que vous génèrerez grâce à la commande suivante :

```
$ tar -cvf VOTRE_NOM_TP_IMAGE.tar VOTRE_NOM_TP_IMAGE
```

NOTEBOOK : une fois votre programme mis au point, copier-coller (a) le résultat de la commande date (b) le code source correspondant (c) la commande de compilation et son résultat (d) le résultat des tests – question 15 (e) le contenu du répertoire TPC (fichiers et sous-répertoires)

PARTIE 4- Notion de module et de compilation séparée

Objectifs : On veut créer un programme constitué de plusieurs fonctions déclarées dans un fichier **.h** et définies dans un fichier **.c**. Pour cela il est nécessaire de maîtrise la notion de **compilation séparée**.

COMPILATION SEPARÉE – Utilisation du Makefile

17) Vérifier que vous êtes bien dans le répertoire **TPC**

18) Créer le répertoire ***VOTRE_NOM*_MODULE_IMAGE** sous **TPC** et vous y déplacer.

19) Créer les sous-répertoires **Data** et **Test** propre au répertoire courant.

20) Récupérer l'archive **MODULE_IMAGE_V0.tar** sur moodle et la décompresser avec :

```
$ tar -xvf MODULE_IMAGE_V0.tar
```

20) Après décompression, étudier le contenu du répertoire **`MODULE_IMAGE_V0`** et de tous les fichiers **`.c`** et **`.h`** présents dans ce répertoire. A quoi correspondent-ils ?

21) Etudier le contenu du fichier **`Makefile`**. Que contient-il ? Faire valider votre réponse auprès de l'enseignante.

Remarque : pour plus de détails sur le fichier Makefile à disposition, vous pouvez consulter le support de cours et la section 2 du site (**https://gl.developpez.com/tutoriel/outil/makefile)**

22) Exécuter la commande **`make mon_module_image.out,`** observer ce qu'il se passe. Lister le contenu du répertoire courant. Que constatez-vous ?

23) Exécuter la commande **`./mon_module_image.out .`** Que se passe-t-il ?

24) Exécuter la commande **`make clean`** et observer le contenu du répertoire courant. Que constatez-vous ?

25) Créer le répertoire **`MODULE_IMAGE_V1`** comme une copie de **`MODULE_IMAGE_V0`**

26) Vous déplacer dans ce nouveau répertoire et renommer certains fichiers de façon à ce que ce répertoire contienne **`: main.c Makefile votre_nom_module_image.c et votre_nom_module_image.h`**

27) Faire les modifications nécessaires dans les différents fichiers pour pouvoir compiler votre propre module image, puis refaire les questions 22) et 23)

Vérifier que vous avez bien l'arborescence :

```
...
TPC
      VOTRE_NOM_TP_IMAGE
            Data
            Test
      VOTRE_NOM_MODULE_IMAGE
            Data
            Test
            MODULE_IMAGE_V0
            MODULE_IMAGE_V1
```

NOTEBOOK : copier-coller (a) le résultat de la commande date (b) le contenu du répertoire TPC (fichiers et sous-répertoires) (c) le contenu du Makefile de la question 21 (d) les réponses aux questions 22, 23, 24 (e) le résultat de la commande history

MODULE MODULE_IMAGE_V1

Objectifs : Faire évoluer votre module **`MODULE_IMAGE_V1`** en développant et testant plusieurs fonctions.

REMARQUE : à ce stade on ne mémorise pas la matrice et on traite les données au fil de l'eau

28) Dans le fichier **`votre_nom_module_image.c`** définir la fonction **`afficher_image_codee`**. Cette fonction effectuera la transformation de l'image suivant le niveau de gris passé en paramètre. De plus, elle renverra une valeur entière qui vaudra suivant le

cas, **-1** si la valeur du niveau est supérieure à 256, **0** si le niveau n'est pas une puissance de 2 et **1** sinon (traitement effectué).

29) Modifier le fichier **votre_nom_module_image.h** pour y inclure la déclaration de la fonction créée à la question 28)

30) Modifier le fichier **main.c** pour tester la fonction **afficher_image_codee** en lui transmettant la valeur du niveau de gris lue sur l'entrée standard, puis en testant un grand nombre d'autres valeurs (cf. Question 15). Pour chaque appel de la fonction, le programme principal affichera également un message approprié suivant la valeur de retour renvoyée par la fonction.

NOTEBOOK : copier-coller (a) le résultat de la commande date (b) le code source du programme fichier .c et .h (c) la compilation et son résultat (d) les résultats des tests effectués

PARTIE 5 - Variable globale et Tableaux à deux dimensions

Objectifs : On veut **effectuer plusieurs traitements sur la même image**. Pour cela, il est nécessaire de la mémoriser dans un tableau à deux dimensions. Modifier les fichiers de votre répertoire **MODULE_IMAGE_V1** en fonction des questions suivantes **:**

31) Définir une **constante symbolique** nommée **NB_MAX** de valeur 16 (on se limitera ainsi à des images de 16 x 16 au maximum). Dans quel fichier faut-il placer cette définition ?

32) Définir une **variable globale IMAGE** susceptible de correspondre à une matrice carrée de rang **NB_MAX**. Cette variable sera utilisée pour stocker l'image. Il faudra impérativement s'assurer aux cours des différentes manipulations, qu'on ne « sort pas de la matrice », c'est-à-dire que l'on n'essaye pas de lire ou de mémoriser plus de valeur qu'elle peut en contenir. Quels sont le ou les fichiers impactés par la définition de cette variable globale ?

33) Ajouter la fonction **lire_image.** Celle-ci lit et mémorise la matrice à partir des valeurs lues sur l'entrée standard. Elle reçoit en paramètre les valeurs **N** et **M** correspondant respectivement au nombre de lignes et au nombre de colonnes de la matrice représentant l'image. Ces valeurs auront été lues et testées par le programme principal. De plus, cette fonction renverra **0** si la mémorisation ne peut pas être faite, **1** sinon. Quels sont le ou les fichiers impactés par l'ajout de cette fonction ? Tester la fonction en ajoutant le code nécessaire dans le fichier **main.c**

ATTENTION: **N** et **M** doivent être inférieurs à **NB_MAX** pour ne pas dépasser la capacité du tableau. Ceci doit être testé par le programme principal avant tout traitement sur l'image (et toute manipulation d'un tableau statique en général).

34) Ajouter la fonction **afficher_image_codee_bis** qui effectue le même traitement que la fonction définie à la question 28), mais cette fois à partir de la matrice mémorisée. En plus du niveau de gris, cette fonction recevra les valeurs de **N** et **M**. en paramètre. Quels sont le ou les fichiers impactés par cette modification ? Tester la fonction en ajoutant le code nécessaire dans le fichier **main.c**

35) Réfléchir à une solution qui vous permet, dans le fichier **main.c** de tester soit la version au fil de l'eau avec la fonction **afficher_image_codée** testée dans la question 30) soit de tester la version avec mémorisation de l'image de la question 33) et 34). Implémenter la solution trouvée.

36) Générer un fichier **`Test_module_image_V1.res`** qui contient les résultats de tous les tests réalisés par votre programme **`main.c`**, les tests doivent être explicites, c'est-à-dire précédés d'un message qui explique à quoi correspond chaque test effectué. Si vous devez effectuer plusieurs exécutions pour réaliser différents tests, vous pouvez nommer les fichiers **`Test_module_image_V1_x.res (Test_module_image_V1_1.res, …)`** et placer ces fichiers dans le répertoire **`Test`** sous **`VOTRE_NOM_MODULE_IMAGE`**.

37) Déposer sous `moodle` l'archive ***`VOTRE_NOM`*`_MODULE_IMAGE_V1.tar`** que vous génèrerez grâce à la commande suivante :

`$ tar -cvf` ***`VOTRE_NOM`*`_MODULE_IMAGE_V1.tar`** ***`VOTRE_NOM`*`_MODULE_IMAGE`**

Cette archive contiendra l'état de ***`VOTRE_NOM`*`_MODULE_IMAGE`** à la fin de cette partie.

PARTIE 6- Pour compléter le module MODULE_IMAGE_V1

Objectifs : Continuer à faire évoluer le contenu de votre répertoire **`MODULE_IMAGE_V1`** en ajoutant de nombreuses autres fonctions et en les testant de la même manière que précédemment.

38) Fonction **`image_miroir`** qui affiche l'image en inversant les colonnes.

39) Fonction **`rotation_image`** qui affiche l'image après une rotation de 90° ou 180°. L'angle de rotation étant passé en paramètre.

40) Fonction **`histogramme`** qui calcule et affiche l'histogramme d'une image. Ici un histogramme correspond à la fréquence avec laquelle chaque niveau de gris est présent dans la totalité de l'image. Cette fréquence est ramenée à un pourcentage.

41) Déposer sous `moodle` l'archive ***`VOTRE_NOM`*`_MODULE_IMAGE_V1_2.tar`** que vous génèrerez grâce à la commande suivante :

`$ tar -cvf` ***`VOTRE_NOM`*`_MODULE_IMAGE_V1_2.tar`** ***`VOTRE_NOM`*`_MODULE_IMAGE`**

Cette archive contiendra l'état de ***`VOTRE_NOM`*`_MODULE_IMAGE`** à la fin de cette partie.

PARTIE 7- Pour aller plus loin : traitements avancés

MODULE MODULE_IMAGE_V2

42) Dupliquer **`MODULE_IMAGE_V1`** et renommer la copie **`MDULE_IMAGE_V2.`**

43) Modifier les noms de fichier de façon à ce que ce répertoire contienne les fichiers **`votre_nom_module_image_v2.c votre_nom_module_image_v2.h main.c et Makefile.`** Ajouter le fichier **`test_module_image_v2.c`** copie du fichier **`main.c.`** Ce fichier contiendra un **`main`** de test de chaque fonction du module (ancien rôle de **`main.c`**) tandis que le fichier **`main.c`** sera dédié au traitement d'images proprement dit.

44) Effectuer les modifications nécessaires dans le fichier **`Makefile`** pour que les bons fichiers soient compilés et que l'on obtienne un exécutable de test **`test_mon_module_image_V2.out`** et un exécutable de traitement **`mon_module_image_v2.out`**.

Objectifs : passer l'image à traiter en paramètre = passer un tableau à 2 dimensions en paramètre

45) Faites les modifications nécessaires dans les différents fichiers pour que l'image ne soit plus une variable globale mais un **paramètre de chaque fonction** qui manipule l'image. Ceci permet de traiter potentiellement plusieurs images différentes dans un même programme. On pourra alors réaliser dans le programme principal (fichier **`main.c`**) un traitement qui lit une image, calcule l'histogramme des niveaux de gris, code l'image sur P niveaux de gris (image1), puis calcule l'image miroir de l'image codée (image_miroir1), l'image après une rotation de 180° de l'image codée (image2) et l'image miroir associée (image_miroir2). Les quatre images seront affichées de telle sorte que l'on ait la configuration suivante :

image1	image_miroir1
image2	image_miroir2

46) Sur le même principe que la question **36),** générer un fichier **`Test_module_image_V2.res`** qui contient les résultats de tous les tests réalisés par votre programme **`test_mon_module_image_V2.out.`**

47) Déposer sous `moodle` l'archive ***`VOTRE_NOM`*`_MODULE_IMAGE_V2.tar`** que vous générerez grâce à la commande suivante :

`$ tar -cvf` ***`VOTRE_NOM`*`_MODULE_IMAGE_V2.tar`** ***`VOTRE_NOM`*`_MODULE_IMAGE`**

Cette archive contiendra l'état de ***`VOTRE_NOM`*`_MODULE_IMAGE`** à la fin de cette partie.

ANNEXE

PARTIE 8- : représentation d'une image Noir et Blanc

En traitement d'images, une image est représentée par un ensemble de points. Ces points sont appelés pixels. Un pixel est codé par une valeur entière représentant le niveau de couleur du point. On parlera de **niveaux de gris** lorsqu'il s'agit d'une image noir et blanc.

Dans un système permettant de représenter **256 niveaux de gris**, le codage du pixel sera compris entre 0 et 255 (0 = noir …dégradé de gris … 255 = blanc). Une image est représentée par une matrice de valeurs entières comportant **`N`** lignes et **`M`** colonnes.

On veut traiter des images codées sur 256 niveaux de gris. Cependant, si le système dont on dispose est trop basique et ne peut pas afficher tous les niveaux de gris, il faut alors transformer l'image. Le codage minimal consiste à représenter 2 valeurs possibles : 0 si le point est éteint (= noir) ou 1 si le point est allumé (= blanc). On parle d'**image binarisée**. Avec 4 niveaux on aura : 0 pour le noir, 1 pour le gris foncé, 2 pour le gris clair et 3 pour le blanc.

PARTIE 9- UNIX : Rappel - Commandes et redirection

man : manuel en ligne : **man commande_Unix** ou **man fonction_C** pour obtenir les détails (syntaxe, utilisation, options, ...)

mkdir : création d'un ou plusieurs répertoires

cd : changer de répertoire de travail

pwd : savoir où on se trouve

ls : lister les fichiers d'un répertoire

nedit, **xemacs**, ... : éditeur de texte

cc ou **gcc** : compilateur langage C

more ou **cat** : visualiser le contenu d'un fichier

(... plus de détails dans le support de cours disponible sous moodle)

Répertoires spécifiques :

~ : répertoire **home**

. : désigne le répertoire courant

.. : répertoire père du répertoire courant.

/ : répertoire racine

rediriger l'entrée standard = ne pas lire les données au clavier mais dans un fichier

```
commande < fichier_entree
```

rediriger la sortie standard = ne pas écrire les résultats à l'écran mais dans un fichier

```
commande > fichier_sortie
commande >> fichier_sortie
```

rediriger la sortie standard erreur = ne pas écrire les messages d'erreur à l'écran mais dans un fichier (si le message est écrit avec **fprintf(stderr, "message") ;**)

```
commande >& fichier_erreur
commande >>& fichier_erreur
```

communication de processus = combiner deux commandes pour que la seconde utilise les résultats de la première

```
commande1 | commande2
```