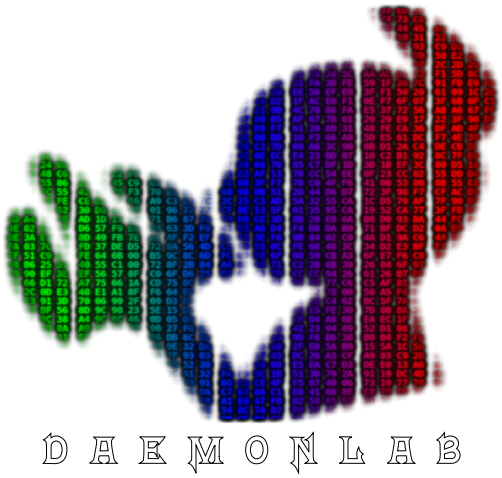

Journée 07

Manipulations binaires, tableaux complexes

- DaemonLab -
daemonlab@ecole-89.com

De nouveaux aspects du langage C sont abordés : les manipulations ayant trait au binaire lui-même ! Vous manipulerez également des tableaux plus complexes.

Nom de code : !pac1j07
Clôture du ramassage : 12/12/2019 23:59

Médailles accessibles :

Définition des médailles à venir

INDEX

Avant-propos :

01 – Détails administratifs
02 – Propreté de votre rendu
03 – Règlement quant à la rédaction du code C
04 – Construction partielle de votre rendu
05 – Fonctions interdites

Exercices principaux :

06 – Manipulations binaires
07 – UTF-8
08 – Matrices
09 – Hexadecimal

01 – Détails administratifs

Votre travail doit être envoyé via l'interface de ramassage du **TechnoCentre** :

http://technocentre.ecole89.com/ramassage

Le numéro de code présent sur ce sujet vous est propre : vous devrez le renseigner en rendant votre travail. En cas d'erreur, votre travail ne sera pas associée à l'activité et votre travail ne sera pas ramassé.

Pour cette activité, vous rendrez votre travail sous la forme d'une archive au format tar.gz. Cette archive devra contenir l'ensemble de votre travail tel que demandé dans la section 4.

Pour créer cette archive .tar.gz, il vous suffit d'utilliser la commande suivante :

```
$> tar cvfz mon_fichier.tar.gz fichier1 fichier2 fichier3
```

Le nom « mon_fichier.tar.gz » étant à remplacé par le nom que vous souhaitez donner votre fichier archive, et « fichier1 », « fichier2 », « fichier3 » par les fichiers ou dossiers que vous souhaitez mettre dans cette archive. Vous pouvez vérifier le contenu de votre archive à l'aide de la commande « **`tar -t mon_archive.tar.gz`** ».

Ce travail est a effectuer seul. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée de la même manière que la correction : de manière **automatique**. Prenez garde si vous pensez pouvoir passer au travers.

Médailles accessibles :

Réussir à rendre :
Vous avez réussi à envoyer votre travail au système de correction.

02 – Propreté de votre rendu

Votre rendu, c'est à dire le contenu de l'archive ou du dépôt que vous entrez sur l'interface du TechnoCentre, doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** ichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale (programme ou bibliothèque)

La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.

Médailles accessibles :

Rendu propre
Votre rendu respecte les règles de propretés imposées.

03 – Règlement quant à la rédaction du code C

En général, le code source de votre programme doit impérativement respecter un ensemble de règles de mise en page définie par les Table de la Norme.

Pour cette activité, nous vous libérons de cette contrainte qui vous sera néanmoins très bientôt imposée pour l'ensemble de vos programmes écrits en C.

04 – Construction partielle de votre rendu

Le programme de correction va construire une sous-partie déterminée de votre rendu afin d'effectuer des tests dessus. En voici les paramètres :

- Les fichiers qui seront compilés sont ceux qui auront l'extension *.c.
- Seuls les fichiers dans le(s) dossier(s) ./ seront compilés.
- Les fichiers seront compilés avec **-W -Wall -Werror**.

- Tous les fichiers seront compilés **ensemble**, cela signifie donc que chaque fonction doit être unique, et que vous pouvez utiliser les fonctions des autres exercices dans chaque exercice.

L'ensemble du code que vous rendez doit pouvoir être compilé.
En cas d'échec de la compilation, vous ne serez pas évalué.

Médailles accessibles :

Construction partielle
Les éléments requis de votre projet se construisent séparement.

05 – Fonctions interdites

Vous n'avez le droit à aucune autre fonction que celle précisée dans la liste ci-dessous :

- write
- malloc
- free

L'utilisation d'une fonction interdite est assimilée à de la triche.
La triche provoque l'arrêt de l'évaluation et la perte des médailles.

06 – Manipulations binaires

Mettre dans le fichier : binary.h et binary.c

Écrivez les macros suivantes, ainsi que leurs tests :

```
#define BITSET(data, bit_nbr)     ...
#define BITRESET(data, bit_nbr)   ...
#define BITGET(data, bit_nbr)     ...
#define BITREV(data, bit_nbr)     ...
```

La macro BITSET met à 1 le bit numéroté bit_nbr dans data.

La macro BITRESET met à 0 le bit numéroté bit_nbr dans data.

La macro BITGET renvoit la valeur du bit numéroté bit_nbr depuis data.

La macro BITREV inverse la valeur du bit numéroté bit_nbr dans data.

Écrivez les fonctions suivantes, ainsi que leur tests :

```
void byte_reverse(char *c);
bool parity(int x);
```

La fonction byte_reverse inverse l'ordre des bits situés dans *c. Par exemple, si *c contient 1001 0101, alors après l'appel de cette fonction, *c contiendra 1010 1001.

La fonction parity renvoi vrai si x contient un nombre pair de bit, sinon elle renvoi faux.

07 – UTF-8

Mettre dans le fichier : utf8.c

Écrivez les fonctions suivantes, ainsi que leurs tests :

```
int chrlen(const char *str);
int wstrlen(const char *str);
int wstrnlen(const char *str, size_t max);
```

La fonction chrlen renvoi la longueur du caractère qui débute à str. Ce caractère est un caractère UTF-8 et peut donc faire de 1 à 4 octets. Si le caractère n'est pas entier, la fonction renvoi un code erreur et exploite errno correctement.

La fonction wstrlen renvoi le nombre de caractères UTF-8 que contient str. Les caractères incomplet en fin de chaîne ne sont pas pris en compte.

La fonction wstrnlen fonctionne comme wstrlen sauf qu'elle s'arrête à maximum max bytes parcouru dans str. Les caractères incomplets en fin de chaîne ne sont pas pris en compte.

08 – Matrices

Fichier : matrix.h et matrix.c

Écrivez la structure **struct s_matrix** disposant d'un alias **t_matrix** et contenant dans l'ordre, un pointeur sur entier **matrix** ainsi que deux entiers : **width** et **height**.

Écrivez ensuite les fonctions suivantes, avec les tests associés :

```
int print_matrix(const t_matrix *mtx);
t_matrix *load_matrix(const char *str);
char *save_matrix(t_matrix *mtx);
void free_matrix(t_matrix *mtx);
int get_matrix(t_matrix *mtx, int x, int y);
t_matrix *sub_matrix(
  t_matrix *mtx,
   int x,
   int y,
   int w,
   int h
);
```

La fonction **print_matrix** affiche la matrice. Passée en paramètre : elle affiche **height** lignes séparés par des saut de lignes de **width** entiers séparés par des espaces. Elle renvoit le nombre de caractères qui ont été écrit.

La fonction **load_matrix** lit dans la chaîne de caractères passée en paramètre la matrice, écrite de la manière suivante : un saut de ligne entre chaque ligne et un espace entre chaque entier. Elle renvoi la matrice ainsi allouée et remplie.

La fonction **save_matrix** écrit une chaîne de caractère basée sur la matrice passée en paramètre. Elle renvoit **NULL** en cas d'erreur.

La fonction **free_matrix** libère l'espace mémoire occupé par la matrice.

La fonction **get_matrix** renvoi l'entier situé à la position **x**, **y**.

La fonction **sub_matrix** crée une nouvelle matrice à partir d'une sous partie décrite d'une autre. En cas de dépassement ou de valeur invalide, un code erreur sera renvoyé et errno sera utilisé.

09 – Hexadecimal

Fichier : hexadecimal.c

Écrivez les fonctions suivantes ainsi que leurs tests :

```
char *base16encode(const void *data, size_t len);
void *base16decode(const char *str, size_t *len);
int hexdump(const void *data, size_t len);
```

La fonction **base16encode** renvoi une chaîne de caractère contenant des caractères hexadécimaux représentant **len** octet depuis **data**. Chaque octet évoluant en deux chiffres hexadécimaux. La chaîne est terminée par un terminateur nul classique.

La fonction **base16decode** renvoi un pointeur vers la donnée lue depuis **str**. La longueur de la donnée renvoyée est écrite dans ***len** en octet.

La fonction **hexdump** affiche les données situés dans **data** sur **len** octets de la manière suivante :

```
$> ./a.out
#################### 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 0123456789ABCDEF
0000 0000 0000 0000: 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 a...............
0000 0000 0000 0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
$>
```

Ici, l'appel a **hexdump** s'est fait sur une plage de donnée ne contenant que des zéros, à l'exception de l'octet 0 qui vaut le caractère **'a'**, commençant à zéro et sur une longueur de 32 octets.

La première ligne est toujours affichée en entier : elle indique la plage où l'adresse est écrite, la numérotation des octets écrit en **hexadecimal** et la numérotation des octets affiché sous leur forme ASCII.

La fonction **hexdump** renvoi le nombre de caractère qui ont été écrit.