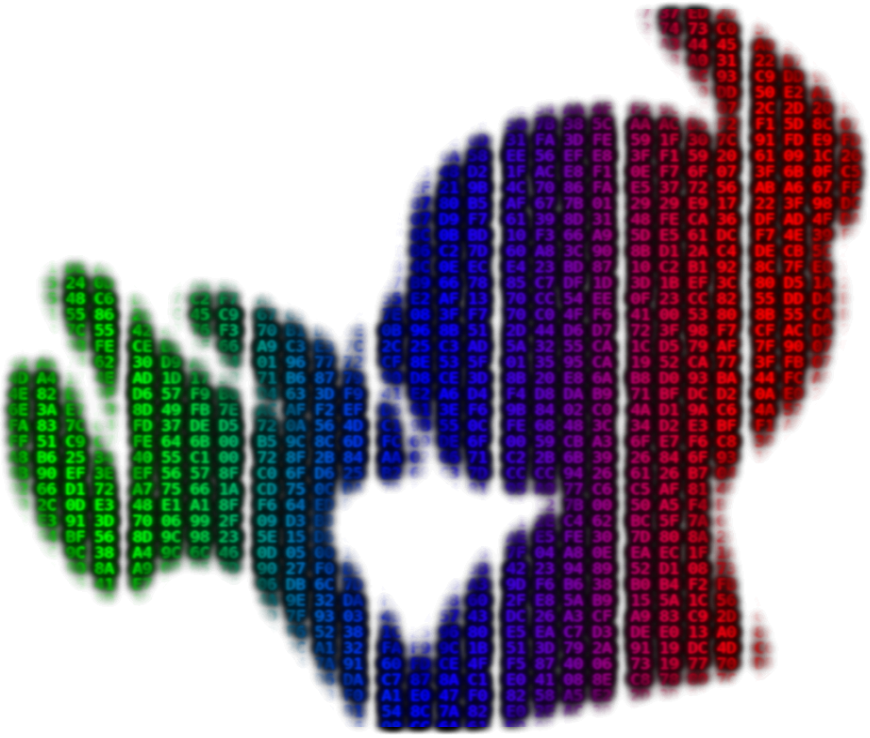

D A E M O N L A B

# Journée 8

## Manipulation de **fichiers**. Paramètre du **main**. Enumerations, unions, boutisme, bits, utf8...

- DaemonLab -

*Dans cette journée, vous allez apprendre à manipuler des fichiers à l'aide des interfaces POSIX.* ***Jetez un œil à vos fonctions autorisées.*** *Nous allons également voir les énumérations et les unions, derniers éléments d'envergures du C qui vous manquent encore.*



# INDEX



# 01 – Avant-propos

Votre travail doit être rendu via le dossier **~/hyperespace/j8/** dans votre espace personnel.

**Si vous faites erreur et que le dossier que vous utilisez pour votre rendu est différent, vous ne serez pas évalué faute d'avoir pu trouver votre travail.**

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.

---

Votre rendu doit respecter **strictement** l'ensemble des règles suivantes :

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

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

Votre programme doit respecter les Tables de la Norme dans leur intégralité. Vous êtes invité à les observer depuis **l'Infosphère**. Elles sont disponibles comme ressource de cette activité.



# 02 – Fonctions autorisées

La bibliothèque logicielle venant avec le C est vaste et disponible. Nous avons cependant fait le choix de vous interdire son utilisation, afin de vous amener progressivement à reprogrammer vous même ses fonctionnalités les plus utiles.

**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.**

Vous n'avez le droit d'utiliser aucune fonction issue de la LibC à l'exception de celles que nous vous autoriserons explicitement.

Pour cette activité, vous avez le droit aux interfaces POSIX, hors **ioctl** :

- open
- close
- read
- write

Ainsi qu'aux fonctions **malloc** et **free**. Notez que dans certains projets, ces fonctions seront interdites totalement, et parfois remplacés par des alter-égo pédagogique : **bunny_malloc** et **bunny_free**. Le modèle de projet qui vous a été communiqué dispose d'une option **BMALLOC** permettant de remplacer automatiquement les appels a **malloc** et **free** par des appels a **bunny_malloc** et **bunny_free**.

Vous avez également le droit à la fonction **alloca**.



# 03 – Méthode de construction

Il peut vous être demandé d'écrire des programmes ou des fonctions.

Dans le cas des programmes, il vous sera toujours demandé de fournir un **dossier** pour l'exercice le requérant. Un **Makefile** vous sera également demandé. Le **nom du programme** de sortie vous sera précisé à chaque fois. Un Makefile incorrect, un mauvais nom de programme, et votre correction n'aura pas lieu…

Dans le cadre des fonctions, il vous ai demandé de fournir le fichier dans votre **dossier de bibliothèque personnelle**, de sorte à ce que vous puissiez utiliser toutes les fonctions que vous avez déjà réalisé jusqu'ici. Pour rappel, le dossier de votre bibliothèque doit être placé à la racine de votre espace personnel et s'appeler **libstd/.**

N'oubliez pas d'entretenir avec soin votre dossier **libstd/** de sorte à ce qu'il soit toujours propre, respecte la norme et soit en état de compiler… sans quoi elle fera obstacle à la correction.

Votre compilation devra toujours comporter les options **-W**, **-Wall** et **-Werror**.



# 06 – cat

**Mettre dans le dossier: cat/ dans le rendu**

Écrivez le programme **std_cat**. *(C'est un programme, donc créez un dossier dans le rendu)*

Le programme **std_cat** prend 0 paramètres ou plus. Tous ces paramètres sont des fichiers. Il affiche ces fichiers en l'état et quitte. Si aucun paramètre n'est donné, **std_cat** lit sur l'entrée standard tant que celle-ci est ouverte. Le symbole '-' peut également être utilisé pour lire sur l'entrée standard.

Voici quelques exemples de tests qui seront effectués sur votre programme :

```
$> man malloc > manmalloc
$> man gcc > mangcc
$> ./std_cat manmalloc mangcc
…
$> ./std_cat
Je tape des caractères
Je tape des caractères
^D
$> cat /dev/null | ./e89_cat
```

Votre programme, en somme, doit réagir comme la commande **cat**. Vous savez donc a quel programme comparer le votre pour savoir si il marche…

*Un test de performance sera effectué sur votre programme.*
*Celui-ci doit être le plus rapide possible !*



# 07 – Manipulations de fichiers

include/ et src/ dans libstd

Écrivez les fonctions suivantes, ainsi que les tests associés. *(Ce sont des fonctions, donc ajoutez les à votre bibliothèque)*

```
typedef enum    e_action
{
      MEASURE_FILE,
      READ_FILE,
      OVERWRITE_FILE,
      EXTEND_FILE
}               t_action;

int             std_manage_file(t_action      action,
                                const char    *file,
                                char          *buffer,
                                size_t        len);
```

La fonction manage_file prend en paramètre l'action à effectuer : renvoyer la taille du fichier file, lire le contenu du fichier et remplir buffer jusqu'à un maximum de len octets, effacer puis écrire buffer de taille len dans file, ou étendre celui-ci en ajoutant les données à la fin.

Pour cet exercice, vous avez besoin de vous renseigner en détail sur les options d'ouverture de fichier avec open.



# 08 – Boutisme

**Mettre dans : endian.h et endian/ dans libstd**

Les micro-processeurs manipulent des séries d'octets afin de produire des données longues. Par exemple, **l'int** mesure 4 octets. En hexadécimal, chaque octet est représentable par un couple de chiffre hexadécimaux, par exemple : 0xAABBCCDD. Une question se pose de ce fait : en mémoire, AA est il avant BB qui est avant CC qui est avant DD ? Ou est ce l'inverse ?

Dans un système **« grand boutiste »** / **« big endian »**, nous avons d'abord AA, puis BB, puis CC et enfin DD en mémoire, occupant chacun un octet. Nous commençons donc par le « grand bout ». Dans un système **« petite boutiste »** / **« little endian »**, c'est l'inverse : d'abord DD, puis CC, BB et enfin AA. C'est le « petit bout » qui arrive en premier. Les micro-processeurs Motorola/Freescale 68x0, les Sun SPARC… sont **big endian**. Les micro-processeurs **Intel**/Zylog/MOS/AMD/**ARM** sont **little endian.** Les **ARM** et Sun peuvent être configuré en big endian, cela dit.

Il existe également des architecture que vous ne rencontrerez probablement jamais ou les octets des entiers vont être dans un style plus baroque : 0xCCDDAABB, par exemple… d'autres combinaisons sont possibles…

Voyant **ARM** et **Intel** du coté des little endian, vous vous dites probablement que finalement vous n'aurez jamais à vous préoccuper du boutisme : perdu. En effet, les entiers envoyés sur le **réseau** doivent dans de très nombreux protocoles voyager en **big endian…**

C'est l'ordre des octets qui détermine le boutisme.
Pas l'ordre des bits.

Vous allez programmer les fonctions suivantes :

```
typedef union       u_intorder
{
      int           integer;
      char          bytes[4];
}                   t_intorder;

typedef enum        e_endian
{
      IS_BIG_ENDIAN,
      IS_LITTLE_ENDIAN
}                   t_endian;

t_endian            std_endian_get(void);
int                 std_endian_revert(int     i);
int                 std_endian_to_big(int     i);
int                 std_endian_to_little(int  i);
```

La fonction **endian_get** regarde quel est la politique du micro-processeur. La fonction **endian_revert** inverse les octets. La fonction **endian_to_big** renvoi une version big endian de **i** (Elle ne fait rien si l'ordinateur est déjà big endian). La fonction **endian_to_little** renvoi une version little_endian de **i** (Elle ne fait rien si l'ordinateur est déjà little_endian)



# 09 – Manipulations binaires

Mettre dans : binary.h et binary/ dans libstd

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 std_binary_reverse(char *c);
bool std_binary_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**.

Si l'opération vous semble complexe, n'oubliez pas que vous avez 25 lignes que vous pouvez exploiter. Les boucles ne sont pas toujours la solution.



# 10 – UTF-8

Mettre dans : utf8.h et utf/ dans libstd

Écrivez les fonctions suivantes, ainsi que leurs tests :

```
int std_utf8_chrlen(const char *str);
int std_utf8_wstrlen(const char *str);
int std_utf8_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.