DAEMONLAB

Journée 2

Boucles avancées, conversion mathématique, erreurs

- DaemonLab -

Durant cette journée, vous construirez des boucles plus complexes et programmerez des conversions à la complexité inattendue.

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/hyperespace/j2/** 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. Dans le cadre de cette activité, la seule fonction à laquelle vous aurez le droit est l'appel système **write**.

Vous pouvez observer le fonctionnement de **write** en tapant « **man 2 write** » dans votre terminal, néanmoins, afin de vous faciliter son utilisation au départ, nous vous avons préparé une fonction que vous êtes libre d'intégrer à votre code.

Cette fonction est **tc_putchar**, elle tente d'écrire le caractère passé en paramètre sur la sortie standard (le terminal, par défaut). Elle renvoi 1 si elle a réussie à écrire le caractère, sinon elle renvoi 0. Ci-dessous, le code de **tc_putchar** suivi d'un programme de test.

```
#include <unistd.h>

int tc_putchar(char c)
{
     return (write(1, &c, 1) > 0);
}
```

```
int main(void)
{
     tc_putchar('A');
     tc_putchar('\n');
     return (0);
}
```

03 – print_square

Fichier à rendre : print_square.c

Écrivez la fonction suivante :

```
void std_print_square(int size);
```

Cette fonction affiche un carré constitué de '-' sur le terminal. Chaque ligne affiche autant de '-' que la valeur de **size** et autant de ligne que **size**. Ci-dessous, un exemple de sortie avec **size** valant 5 :

```
$> ./a.out
-----
-----
-----
-----
-----
$>
```

Cet exercice est **fondamental**. Il n'a peut-être l'air de rien de l'extérieur, mais il vous fait mettre en pratique une construction algorithmique extrêmement banale que vous retrouverez **partout**.

Médailles accessibles :

Double boucle
Vous êtes parvenu à programmer une boucle dans une boucle.

04 – is_upper

Fichier à rendre : is_upper.c

Écrivez la fonction suivante :

```
int std_is_upper(char c);
```

Cette fonction renvoi 1 si le caractère envoyé est en majuscule. Sinon 0.

N'oubliez pas de joindre votre fonction de test.

05 – sum

Fichier à rendre : sum.c

Écrivez la fonction suivante ainsi que son test :

```
int std_sum(int c);
```

Cette fonction renvoi la somme de tous les entiers allant de 0 à **c** inclus, que c soit négatif ou positif.

06 – factoriel

Fichier à rendre : factoriel.c

Écrivez la fonction suivante ainsi que son test :

```
int std_factoriel(int c);
```

Cette fonction renvoi la multiplication de tous les entiers allant de 1 à **c** inclus.

Un int ne peut compter que jusqu'à 2147483647.
Aller au-delà produit un résultat indeterminé.

Vous devez être en mesure d'éviter le dépassement et renvoyer -1 si vous n'êtes pas en mesure de calculer la valeur attendue. N'oubliez pas de tester les cas d'erreur **aussi**.

Si la valeur passée en paramètre génère une erreur, en plus de renvoyer le code d'erreur -1, vous devez également mettre dans **errno**, variable globale accessible si vous avez inclus <errno.h>, la valeur **ERANGE** qui signifie « Capacité de variable dépassée ».

N'oubliez pas dans votre test, de vérifier aussi **errno**.

07 – to_digit

Fichier à rendre : to_digit.c

Vous vous êtes entraîné à manipuler des caractères dans la précédente journée, mais êtes vous tout à fait clair de la relation qui existe entre la table ASCII et les entiers ?

Écrivez la fonction suivante, qui transforme un int dont la valeur est comprise entre 0 et 9 compris en son chiffre, caractère ASCII, correspondant.

```
int std_to_digit(int nbr);
```

N'oubliez pas d'écrire également le test associé, et en cas d'erreur, de renvoyer le code d'erreur -1 et de mettre **EINVAL** dans **errno**. **EINVAL** signifie « paramètre invalide ».

Cet exercice est une dépendance de l'exercice std_print_base10.
ous aurez besoin de cette fonction plus tard.

08 – roll_2d6

Fichier à rendre : roll_2d6.c

Écrivez la fonction suivante :

```
void std_roll_2d6();
```

Cette fonction affiche toutes les combinaisons possible du score de deux dés à 6 faces, dans l'ordre croissant. Par exemple : 11, 12, 13, 14, 15, 16, 22, 23, etc. Notez l'absence de 21, qui n'est pas présent car 12 a déjà été tiré.

Chaque combinaison sera écrite suivie d'un saut de ligne, sans aucun autre séparateur.

09 – get_numeral

Fichier à rendre : get_numeral.c

Écrivez la fonction suivante :

```
int std_get_numeral(int nbr, int mul);
```

Cette fonction récupère dans **nbr** le chiffre correspondant à la puissance de 10 situé dans mul. Par exemple :

- Pour **nbr** valant 12345 et **mul** valant 0, la fonction renverra 5.

- Pour **nbr** valant -12345 et **mul** valant 1, la fonction renverra 4.

- Pour **nbr** valant 12345 et **mul** valant 2, la fonction renverra 3.

- Pour **nbr** valant -12345 et **mul** valant 3, la fonction renverra 2.

- Pour **nbr** valant 12345 et **mul** valant 4, la fonction renverra 1.

- Pour **nbr** valant 12345 et **mul** valant 5, la fonction renverra -1 et mettra **ERANGE** dans **errno**.

- Si **mul** est négatif, la fonction renverra -1 et mettra **EINVAL** dans **errno**.

Cet exercice est une dépendance de l'exercice std_print_base10.
ous aurez besoin de cette fonction plus tard.

Pour réussir cet exercice, pensez division : résultat et reste.

10 – pow

Fichier à rendre : pow.c

Écrivez la fonction suivante :

```
int std_pow(int a, int b);
```

Cette fonction renvoi le nombre a mit à la puissance b. Par exemple, si a vaut 3 et b vaut 4, alors la valeur de retour de la fonction vaudra 81. N'oubliez pas de tester votre fonction.

Concernant les valeurs autorisées de b, seules des valeurs positives seront testés.

**Cet exercice est une dépendance de l'exercice std_print_base10.
ous aurez besoin de cette fonction plus tard.**

C'est également une fonction très utile en général.

11 – print_base10

Fichier à rendre : print_base10.c

Écrivez la fonction suivante :

```
int std_print_base10(int nbr);
```

Cette fonction affiche **nbr** sur le terminal (en base 10) et renvoi le nombre de caractère qui ont été écrit. Attention : ces nombres peuvent être positifs comme négatif. Toutes les valeurs de **nbr** doivent être gérées.

Nous attendons une fonction de test où vous testerez au moins la valeur de retour de cette fonction, à défaut de tester ce qui est affiché.

Cet exercice est beaucoup plus simple si vous disposez des fonctions std_to_digit, std_get_numeral et std_pow.

12 – roll_3d10

Fichier à rendre : roll_3d10.c

Écrivez la fonction suivante, fonctionnant sur le même principe que roll_2d6.

```
void std_roll_3d10(void);
```

Les valeurs des dés lancés sont comprises dans [0 ; 9]. Voici le début et la fin de la sortie attendue : 000, 001, 002, 003, 004, 005, 006, 007, 008, 009, 011, 012, 013, 014, 015, 016, 017, 018, 019, 022, 023, 024, 025, 026, 027, 028, 029, 033 … 889, 899, 999.

Chaque combinaison sera écrite suivie d'un saut de ligne, sans aucun autre séparateur.

13 – print_base2

Fichier à rendre : print_base2.c

Écrivez la fonction suivante :

```
int std_print_base2(int nbr);
```

Cette fonction affiche nbr (**en base 2, en binaire**) sur le terminal et renvoit le nombre de caractère qui ont été écrit. Seul des nombres positifs et 0 seront testés.

Nous attendons une fonction de test où vous testerez au moins la valeur de retour de cette fonction, à défaut de tester ce qui est affiché.

14 – from_digit

Fichier à rendre : from_digit.c

Écrivez la fonction suivante :

```
int std_from_digit(char c);
```

Cette fonction renvoi l'entier symbolisé par le caractère c. Par exemple, si '4' est envoyé à cette fonction, elle renverra l'entier 4. N'oubliez pas d'écrire le test et de gérer les erreurs avec **errno** et **EINVAL**. -1 est le code d'erreur de cette fonction

15 – is_lower

Fichier à rendre : is_lower.c

Écrivez la fonction suivante ainsi que son test :

```
int std_is_lower(char c);
```

Cette fonction renvoi 1 si le caractère envoyé en paramètre est en minuscule, sinon 0.

16 – to_lower

Fichier à rendre : to_lower.c

Écrivez la fonction suivante ainsi que son test :

```
char std_to_lower(char c);
```

Cette fonction renvoi le caractère c passé en minuscule, si c'est pertinent. Sinon il renvoi le caractère c.

17 – to_upper

Fichier à rendre : to_upper.c

Écrivez la fonction suivante ainsi que son test :

```
char std_to_upper(char c);
```

Cette fonction renvoi le caractère c passé en majuscule, si c'est pertinent. Sinon il renvoi le caractère c.

18 – is_space

Fichier à rendre : is_space.c

Écrivez la fonction suivante ainsi que son test :

```
int std_is_space(char c);
```

Cette fonction renvoi 1 si le caractère envoyé en paramètre est un « espace », sinon 0. Afin de vous renseigner sur ce qu'est un « espace » dans le contexte actuel, vous êtes invités à rechercher dans le manuel de **isspace**.

19 – is_blank

Fichier à rendre : is_blank.c

Écrivez la fonction suivante ainsi que son test :

```
int std_is_blank(char c);
```

Cette fonction renvoi 1 si le caractère envoyé en paramètre est un « blanc ». Afin de vous renseigner sur ce qu'est un « blanc » dans le contexte actuel, vous êtes invités à rechercher dans le manuel de **isblank**.

20 – is_digit

Fichier à rendre : is_digit.c

Écrivez la fonction suivante ainsi que son test :

```
int std_is_digit(char c);
```

Cette fonction renvoi 1 si le caractère envoyé en paramètre est un chiffre, sinon 0.

21 – is_alpha

Fichier à rendre : is_alpha.c

Écrivez la fonction suivante ainsi que son test :

```
int std_is_alpha(char c);
```

Cette fonction renvoi 1 si le caractère envoyé en paramètre est une lettre, sinon 0.