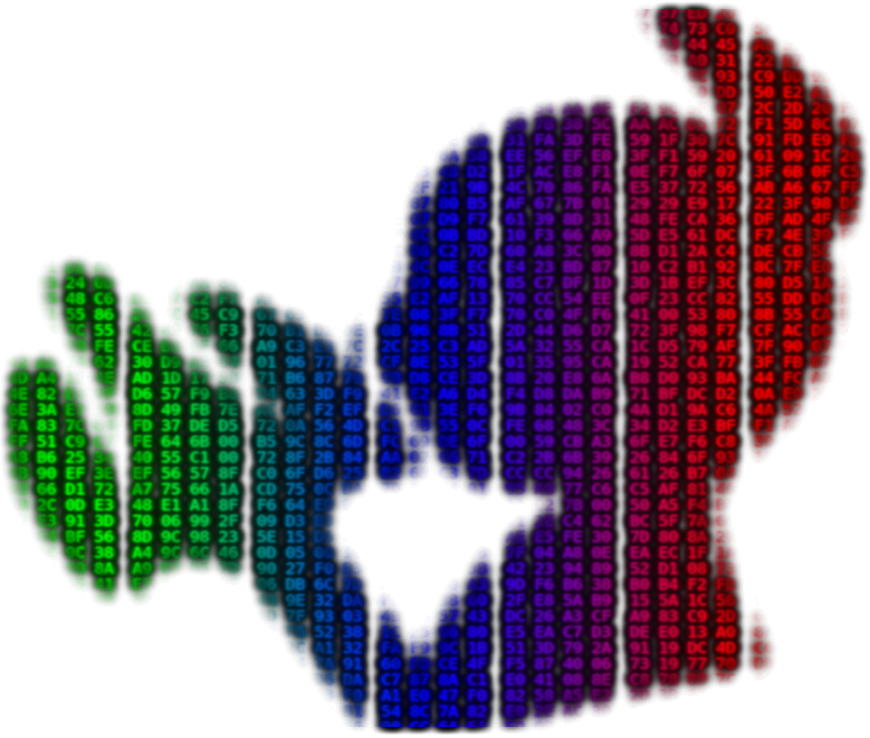

D A E M O N L A B

Journée 3

Chaînes de caractères

- DaemonLab -

Durant cette journée, vous allez commencer à faire de la magie avec votre ordinateur. Nous abordons les chaînes de caractères et donc les manipulations manuelles de la mémoire.

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/hyperespace/j3/** 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 – strlen

Fichier à rendre : strlen.c

Écrivez la fonction suivante, ainsi que son test :

```
size_t std_strlen(const char *str);
```

Cette fonction prend en paramètre une chaîne de caractère dont elle va renvoyer la longueur calculée. Tout comme la véritable fonction **strlen**, vous n'avez pas à gérer le pointeur NULL : c'est à l'appelant de faire attention. Vous pouvez le faire si vous le souhaitez, mais la correction automatique ne le fera pas. Tant qu'à faire, utiliser une valeur de **errno** que vous jugez pertinente.

Le mot clef **const** indique l'interdiction de modifier un élément. Il indique l'interdiction de modifier l'élément situé à sa gauche, ou si il n'y en a pas, celui à sa droite. Ici, il indique donc que la fonction est interdite de modifier les **char** pointées par **str**.

La signification du type **size_t** est : un nombre entier sans signe. **size_t** est utilisé pour représenter des longueurs en mémoires ou des positions.

Ci-dessous, un main de test pour vous aider à vous lancer dans cette journée :

```
int main()
{
	if (std_strlen("chaine de 12") == 12)
		tc_putchar('y') ;
	else
		tc_putchar('n') ;
	return (0);
}
```

04 – puts

Fichier à rendre : puts.c

Écrivez la fonction suivante, ainsi qu'un test minimal vérifiant la valeur de retour :

```
int std_puts(const char *str);
```

Cette fonction affiche la chaîne de caractère passée en paramètre **suivi d'un saut de ligne**. Si une erreur a lieu, la fonction renverra -1 et laissera les fonctions sous-jacentes définir **errno**.

La fonction **tc_putchar** renvoi le nombre de caractère écrit, c'est donc l'occasion pour la première fois d'exploiter cette information.

05 – rputs

Fichier à rendre : rputs.c

Écrivez la fonction suivante, ainsi qu'un test minimal vérifiant la valeur de retour :

```
int std_rputs(const char *str);
```

Cette fonction affiche la chaîne de caractère passée en paramètre **à l'envers** suivi d'un saut de ligne. Si une erreur a lieu, la fonction renverra -1 et laissera les fonctions sous-jacentes définir **errno**.

06 – strcmp

Fichier à rendre : strcmp.c

Nous entrons dans une nouvelle phrase d'extension de votre part du travail !

Vous allez programmer la fonction **std_strcmp**, la fonction **test_strcmp** qui testera **tc_strcmp** comme précédemment, mais je ne vous détaillerai plus les fonctions à faire désormais ! (Pas celle qui existent dans votre système, du moins).

De ce fait, pour savoir ce que fait la fonction **strcmp**, tapez **« man strcmp »** dans votre terminal (ou sur internet) et reproduisez-en le fonctionnement.

07 – strchr

Fichier à rendre : strchr.c

Implémentez les fonctions **std_strchr** et **test_strchr**, basé sur la fonction **strchr**.

Vous avez un problème car le paramètre de std_strchr est un **const char*** et le type de retour un **char*** ? Faites un ***cast,*** la syntaxe est la suivante **(type)variable** et sert à considérer temporairement **variable** comme étant du type **type**.

08 – strrchr

Fichier à rendre : strrchr.c

Implémentez les fonctions **std_strrchr** et **test_strrchr**, basé sur la fonction **strrchr**.

09 – strcpy

Fichier à rendre : strcpy.c

Implémentez les fonctions **std_strcpy** et **test_strcpy**, basé sur la fonction **strcpy**.

Voici un main de test pour vous aider.

```
int main(void)
{
     const char *str = "Bonjour!";
     char buffer[32];

     std_strcpy(buffer, str);
     if (strcmp(buffer, str) == 0)
          tc_putchar('y');
     else
          tc_putchar('n');
     return (0);
}
```

10 – memset

Fichier à rendre : memset.c

Implémentez les fonctions **std_memset** et **test_memset**, basé sur la fonction **memset**.

11 – Aller plus loin

Implémenter les fonctions suivantes, ainsi que leurs tests :

- strnlen

- strncpy

- strlcpy

- strncmp

- strcasecmp

- strncasecmp

- strchrnul

15 – En cas de difficulté

Consultez le manuel de la fonction **bzero** et écrivez **std_bzero** qui en est un clone ainsi que **test_bzero** qui teste la fonction **tc_bzero** qui sera apportée par le système de correction.