D A E M O N L A B

Panneau défilant

Peu d'espace, un long message, que faire ?

- DaemonLab -

Cet exercice à réaliser en équipe consiste à afficher sur un cours espace une chaîne de caractère défilante.

Pour cette ruée, votre équipe sera… aléatoire.

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/hyperespace/rue1/** 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.

Chaque équipe doit rendre un travail original. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'une autre équipe, 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.

L'administrateur de votre équipe est responsable du rendu.

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**.

04 – text_scrolling

Écrivez la fonction suivante :

```
void e89_text_scrolling(const char      *str,
                        int             size,
                        int             stop);
```

Cette fonction effectue **stop** action. Une action consiste à afficher du texte suivi d'un **retour chariot '\r'.** Le texte affiché est celui situé dans **str** et celui-ci est affiché maximum de **size** caractère.

La façon d'afficher ce texte est particulière : au départ, la ligne affichée est entièrement blanche. La chaîne de caractère commence à défiler depuis la droite de la ligne et se déplace caractère par caractère jusqu'à ce que son dernier caractère est disparu sur la gauche.

Lorsque le texte a fait un tour entier, cela recommence. On ne s'arrête que si **stop** actions ont été effectuées.

La ligne affiché doit toujours contenir **size** caractère. Utilisez des espaces comme bourrage.

Ci-dessous, un exemple avec « abc » comme **str**, 5 comme **size** et 10 comme **stop**.
Pour des raisons de lisibilité, le bourrage est effectué avec un tiret« - » et un saut de ligne est réalisé à la place d'un retour chariot.

```
$> ./a.out
-----
----a
---ab
--abc
-abc-
abc--
bc---
c----
-----
----a
$>
```

07 – text_bouncing

Écrivez la fonction suivante :

```
void e89_text_boucing(const char        *str,
                      int               stop);
```

Cette fonction effectue **stop** action. Une action consiste à afficher du texte après avoir effectué un mouvement Le texte affiché est celui situé dans **str.** La zone d'affichage fera deux fois la taille de str et le texte se déplacera de gauche à droite puis de droite à gauche.

A chaque fois, c'est le contact d'un des bords avec une des extrémités des lettres qui effectue le rebond.

La ligne écrite fera toujours deux fois la taille de **str**. **Bourrez avec des espaces si nécessaire.**