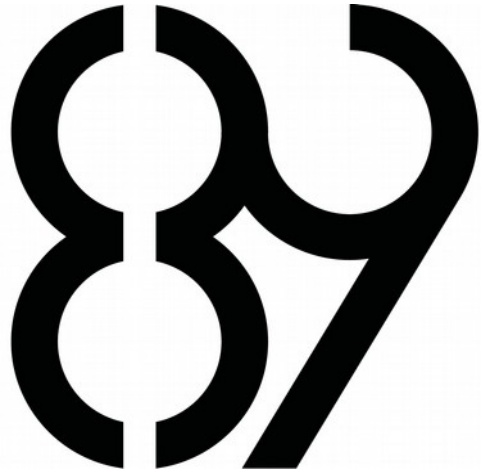
DAEMONLAB



EXAM A/B

Examen de difficulté A

- DaemonLab -
daemonlab@ecole-89.com

Médailles accessibles :

Définition des médailles à venir

INDEX

Avant-propos :

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

Vous devez respecter les règles des Tables de la Norme, sans exception.

04 – Construction 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) ./ et sous-dossiers 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 séparément.

05 – Fonctions interdites

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

- write

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.

Voici le prototype de la fonction write, que vous trouverez dans unistd.h

```
#include <unistd.h>
ssize_t write(int fd, const char* data, size_t count);
```

La fonction write sert à écrire des données et fonctionne de la façon suivante :
- Son premier paramètre, **fd**, permet de choisir où est ce que l'on écrit. Si l'on envoi la valeur 1, cela signifie sur « la sortie standard du programme ».
- Son second paramètre, **data**, est l'adresse où se situe la donnée que l'on souhaite écrire.
- Son troisième paramètre, **count**, est la taille que l'on souhaite écrire. Généralement, c'est la taille de **data**. size_t est un type specifique de nombre entier.
- La fonction write renvoi -1 en cas d'erreur, sinon elle renvoi le nombre de bytes qu'elle à écrit. Pour plus d'information, vous pouvez taper « main 2 write » dans votre terminal.

Pour finir, write est un **appel système**. C'est un outils que vous apporte votre **système d'exploitation UNIX** et non une fonction du langage C. Les fonctions du langage C sont construites à l'aide de ce genre d'éléments. Ci-dessous, une implémentation de e89_putchar qui renvoi vrai si l'on a bien écrit le caractère demandé, afin de vous donner un exemple :

```
bool e89_putchar(char c)
{
	// Si write renvoit 1, alors 1 == 1 est vrai.
	return (write(1, &c, 1) == 1);
}
```


06 – Affichage de lettre

Implémentez la fonction suivante :

```
int             e89_until(char          letter)
```

Cette fonction affiche tous les caractères depuis **letter** jusqu'à '?', suivi d'un saut de ligne.

Le paramètre **letter** peut valoir n'importe quelle valeur situé entre 32 et 126.

07 – Tube

Implémentez la fonction suivante :

```
void            e89_tube(int                    width,
                         int                    height)
```

Cette fonction affiche un rectangle faisant **width** de largeur et **height** de hauteur.

Ce rectangle utilise des numéros pour indiquer sa profondeur.
Le paramètre **width** peut varier de 0 à 9 seulement.

Ci-dessous, un exemple de rectangle faisant 5 de largeur et 7 de hauteur

```
01210
01210
01210
01210
01210
01210
01210
```


08 – Majuscule

Implémentez la fonction suivante :

```
void            e89_upper_those(char          *s,
                                const char    *t)
```

Cette fonction met en majuscule les caractères de s que l'on peut trouver dans t.

Par exemple, si s vaut « abc » et que t vaut « bgh », alors la chaîne s vaudra « aBc » après l'appel à e89_upper_those.

Ci-dessous, un programme de test. Vous ne devez pas le rendre. Encore en dessous, son exécution.

```
int main(void)
{
	char buf[16] = {'S', 'a', 'l', 'u', 't', 0};

	e89_upper_those(buf, "lt");
	e89_puts(buf);
}
```

```
$> ./a.out
SaLuT
$>
```

09 – Triangle rectangle

Implémentez la fonction suivante :

```
void            e89_rectriangle(int                 siz)
```

Cette fonction affiche un triangle rectangle dont les deux cotés perpendiculaires font siz.

Ci-dessous, un triangle rectangle de taille 5, dans le sens attendu, suivi d'un triangle rectangle de taille 2 et 1.

```
+----
|   /
|  /
| /
|/

+-
|/

+
```