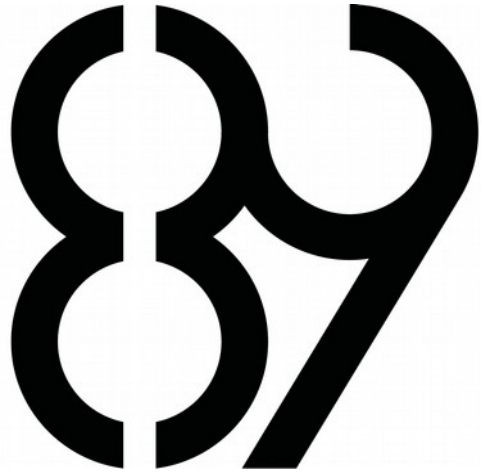
DAEMONLAB



CLAVIERCRAN

Marathon de programmation système

- DaemonLab -
pedagogie@ecole-89.com

Deux exercices d'une heure, un de deux. Top chrono. Go !

INDEX

Avant-propos :

01 – Détails administratifs

Votre travail doit être envoyé via l'interface de ramassage de l'Infosphère :

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

Votre programme doit respecter la **Table des Normes**.

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) ./ 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 :

opendir	stat	close
readdir	malloc	read
closedir	free	write
chdir	open	lseek

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.

06 – Exercice 1

Votre programme prendra en paramètre un nombre indéterminé de fichiers.

Il devra afficher sur le terminal leur taille en **octet** avant de sauter une ligne puis d'afficher le contenu de ces fichiers. Un saut de ligne final clôturera l'affichage du fichier. Votre programme continuera tant qu'il n'aura pas épuisé ses paramètres.

L'ordre de lecture des paramètres sera fait de 1 jusqu'à **argc** non compris.

07 – Exercice 2

Votre programme prendra en paramètre des chemins de **dossiers**. Vous afficherez le contenu de ces dossiers tel qu'il sera renvoyé par les fonctions de parcours de dossiers. *Il n'est **pas** demandé de parcourir récursivement les sous-dossiers que vous rencontrerez, seulement les dossiers passés en paramètre !*

Votre affichage fonctionnera de la façon suivante : vous afficherez le paramètre suivi du symbole ':' suivi d'un saut de ligne. Ensuite, chaque fichier ou dossier sera affiché. Les fichiers et dossiers seront séparé par un unique **espace**. Il ne **devra pas** y avoir d'espace en fin de ligne, mais un **saut de ligne** qui conclura le parcours du paramètre.

Les noms des dossiers **rencontrés** seront suffixés par '/'.

Vous recommencerez tant que vous n'aurez pas parcouru **argv** de 1 jusqu'à **argc** exclu.

08 – Exercice 3

Cet exercice dure 2 heures

Votre programme prendra en paramètre deux fichiers. Sa tache sera de parcourir octet par octet les deux fichiers. Tant que les octets sont identiques dans les deux fichiers, il se contente d'afficher les informations suivantes : à gauche le premier fichier, à droite le second. Les octets sont affichés **en base 10, et seulement en positif**.

Dans un premier temps, réalisez seulement cette partie là : lisez les deux fichiers **en même temps**, en affichant les valeurs de leurs octets.

Une fois terminé cette première partie, vous allez travailler sur ce qu'il se passe lorsque la différence est rencontrée :

Si il y a une différence, il affiche un prompt '!' et attend une entrée de l'utilisateur. L'utilisateur peut alors entrer un symbole au choix parmi '<' et '>'. Tout autre symbole est rejeté et le prompt est affiché de nouveau.

Le symbole '<' indique que c'est le premier fichier, celui de gauche, qui est « bon » et celui de droite qui présente un « problème ». Cela provoque le déplacement de la tête de lecture du fichier de droite de 1 octet, sans faire progresser celle de gauche. Le symbole '>' indique que c'est le second fichier, celui de droite, qui est « bon » et donc c'est à l'inverse le fichier de gauche qui verra sa tête de lecture avancer d'un octet.

Si après avoir progressé, les octets sont de nouveau identique, on continue à avancer automatiquement jusqu'à la fin ou la prochaine différence.

Si la fin de l'un des deux fichiers devait être atteinte avant l'autre, il serait simplement affiché quel fichier est terminé et combien d'octets restait-il pour celui qui ne l'était pas. Si les deux fichiers sont terminés en même temps, rien n'est affiché.

Voici un exemple qui vous permettra de comprendre les subtilités de la méthode d'affichage attendue. En noir, ce sont les entrées utilisateurs.

```
$> ./manualdiff fichier1 fichier2
127 127
240   3
!<
240 240
 25  25
174 174
 56 100
!>
 88 100
!>
100 100
fichier1 terminated
fichier2 12 bytes before end
$>
```