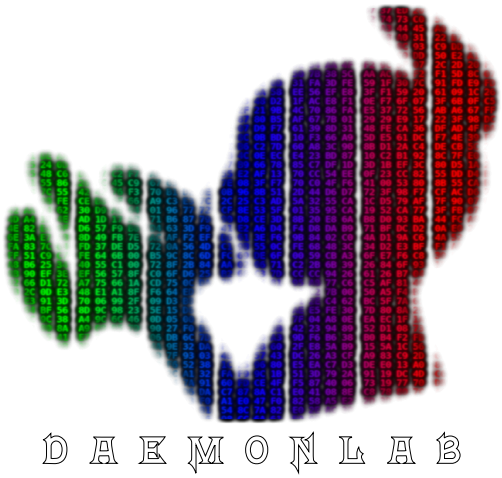

Journée 03

Analyse grammaticale, allocation dynamique

- DaemonLab -
daemonlab@ecole-89.com

L'analyse grammaticale est l'action qui consiste à faire la lecture d'une suite d'octets non pour en déduire le sens. L'allocation dynamique est l'acte de réservation de blocs mémoire pour le programme détaché des variables des fonctions

Nom de code : !pac1j03
Clôture du ramassage : 26/11/2019 23:59

Médailles accessibles :

Définition des médailles à venir

INDEX

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

En général, le code source de votre programme doit impérativement respecter un ensemble de règles de mise en page définie par les Table de la Norme.

Pour cette activité, nous vous libérons de cette contrainte qui vous sera néanmoins très bientôt imposée pour l'ensemble de vos programmes écrits en C.

04 – Construction partielle 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 sans règles de compilation particulière.

- 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 séparement.

05 – Fonctions interdites

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

- write
- malloc
- free

06 – Exercices principaux

Programmez les fonctions **e89_fonction** et **test_fonction**, situé dans les fichiers **fonction.c** pour chacune des fonctions situés ci-dessous :

```
int atoi(const char *str);
char *strcat(char *target, const char *src);
char *strdup(const char *str);
char *strndup(const char *src, size_t max);
char *substr(const char *src, size_t beg, size_t end);
char *strstr(const char *str, const char *token);
void *memcpy(char *dest, const void *src, size_t len);
```

Les informations dont vous avez besoin se situent pour beaucoup dans les manuels UNIX : « man fonction » est votre allié. Si vous n'avez pas les manuels, vous pourrez les trouver sur internet.

La fonction **substr** génère une nouvelle chaîne contenant les caractères situés entre **end** et **beg**. Elle retourne **NULL** et met **EINVAL** dans **errno** si **beg** est plus grand que **end**.

07 – Aller plus loin

Programmez les fonctions **e89_fonction** et **test_fonction**, situé dans les fichiers **fonction.c** pour chacune des fonctions situés ci-dessous :

```
char *strnstr(char *str, const char *tok, size_t len);
char *strncasestr(char *s, const char *t, size_t l);
void *memmove(void *dest, const void *src, size_t len);
void *memmem(const void *str, size_t str_len,
             const void *tok, size_t tok_len);
```

La fonction **strnstr** fonctionne comme **strstr**, limité à **len** caractère dans **str**.

La fonction **strncasestr** fonctionne comme **strnstr** sauf qu'elle ignore la casse.

Concernant **memmove** et **memmem**, **man** est votre allié.

08 – En cas de difficulté

Programmez les fonctions **e89_fonction** et **test_fonction**, situé dans les fichiers **fonction.c** pour chacune des fonctions situés ci-dessous :

```
void *memdup(const void *data, size_t len);
int memcmp(const void *a, const void *b, size_t len);
```

La fonction **memdup** duplique **len** octets situés dans **data** et les renvoi.

La fonction **memcmp** est documentée. Ouvrez vos **manuels**.