D A E M O N L A B

Journée 4

Analyse grammaticale, allocation dynamique

- DaemonLab -

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

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/hyperespace/j4/** 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.

Pour cette activité, vous avez le droit à l'appel système **write** ainsi qu'aux fonctions **malloc** et **free**.

03 – Exercices principaux

Programmez les fonctions **std_fonction** et **test_fonction**, situé dans les fichiers **fonction.c** pour chacune des fonctions situés ci-dessous. Par exemple, pour **atoi**, vous écrivez **std_atoi** et **test_atoi** dans **atoi.c**.

```
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 *strstr(const char *str, const char *token);
void *memcpy(char *dest, const void *src, size_t len);

char *substr(const char *src, size_t beg, size_t end);
```

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** n'est pas une fonction standard et n'a donc pas de manuel. Elle 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**.

04 – Aller plus loin

Programmez les fonctions **std_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.

05 – En cas de difficulté

Programmez les fonctions **std_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**.