Basics

Programmation

infosphere@ecole-89.com

Ce document est strictement personnel et ne doit en aucun cas être diffusé.

Table des matières

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 zippée. Cette archive devra contenir l'ensemble de votre travail.

Ce travail est à 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.

Propreté de votre rendu

Votre rendu, c'est à dire le contenu de l'archive ou du dépôt que vous entrez sur l'interface de l'infosphere (intra.ecole-89.com), doit respecter strictement l'ensemble des règles suivantes :

- Il ne doit contenir aucun fichier tampon. (*~, #*#)
- Seuls les fichiers demandés doivent être rendus

Toute production de code HTML ou CSS doit respecter les normes de W3C.

Règlement quant à la rédaction du code C

Votre programme doit respecter la Table des Norme

Compilation et fichiers à rendre

Chaque exercice est à rendre dans un fichier séparé et sera compilé indépendamment des autres. Cela veut dire que depuis un exercice donné, vous ne pouvez pas faire appel à des fonctions implémentées dans un autre.

Exemple, soit un exercice appelé `strcmp` : vous devez rendre un fichier `strcmp.c`, à la racine de votre `.tar.gz` de rendu (c.à.d. qui n'est pas dans un sous-dossier de votre rendu.) Depuis cet exercice, admettons vous avez besoin de `e89_puts`, vous ne pourrez pas appeler une fonction contenue dans `puts.c` et devrez la recopier, si tenté qu'elle ne contienne pas d'appels interdits.

Toujours pour un exercice appelé `strcmp` : il sera compilé comme ceci :

```
gcc strcmp.c main_de_correction.c
```

Fonctions autorisées

Aucune fonction des bibliothèques standards ne sont autorisées si elles ne sont pas explicitement listées dans l'énoncé d'un exercice.

Attention, `sizeof` est un mot-clé, pas une fonction, et est de fait autorisé.

Exercice 00 - `puts`

fichier à rendre : `puts.c`

Implémentez la fonction `e89_puts` qui doit se comporter comme la fonction `puts` de la bibliothèque standard.
Son prototype étant :

```
int puts(const char *s);
```

La fonction doit écrire le contenu de la chaîne de caractère sur la sortie standard et le faire suivre d'un retour à la ligne.

Fonction autorisée : `write`.

Exercice 01 - `strcmp`

fichier à rendre : `strcmp.c`

Implémentez la fonction `e89_strcmp` qui doit se comporter comme la fonction `strcpm` de la bibliothèque standard.
Son prototype étant :

```
int e89_strcmp(const char *s1, const char *s2);
```

Il est attendu que la fonction renvoie `0` dans le cas où les deux chaînes tiennent la même succession de caractères.
Pour rappel, vous pouvez consulter le manuel de `strcmp` à l'aide de la commande `man strcmp`.

Fonction autorisée : aucune.

Exercice 02 - `split`

fichier à rendre : `split.c`

Implémentez la fonction `split`. Elle doit éclater la chaîne passée en paramètre en plusieurs sous-chaînes. Le délimiteur étant le caractère `token`.
Chaque sous-chaîne doit être allouée dynamiquement.

Son prototype est :

```
char **split(const char *str, char token);
```

Fonction autorisée :

- `malloc`,
- `free`.

Split : Exemple 1

Soit :

- une chaîne `str` contenant la succession de caractères suivante : `"abc;def;ghi"` ;
- un caractère `token` contenant `';'`.

Un appel à `split` avec pour paramètres la chaîne `str` et le caractère `token` renverrait : un tableau de chaînes de caractères contenant :

- `"abc"`,
- `"def"`,
- `"ghi"`,
- `NULL`.

Split : Exemple 2

Soit un `token` de valeur `'d'` et la même chaîne de caractères `str`.
Un appel à `split` avec pour paramètres la chaîne `str` et le caractère `token` renverrait : un tableau de chaînes de caractères contenant :

- `"abc;"`,
- `"ef;ghi"`,
- `NULL`.

Le caractère passé en jeton `token` n'apparaît pas dans les sous-chaînes. On retrouve dans la première chaîne tout ce qui le précède et dans la seconde, tout ce qui le suit. Le troisième et dernier pointeur est `NULL` pour nous signifier qu'il ne reste plus rien à lire dans le tableau.

Conseil

Vous êtes très fortement encouragés à écrire la fonction qui vous permet de visualiser le contenu de votre `char **`. Auquel cas, la présence d'un ou plusieurs appels à `write`, dans le fichier que vous rendez, sera tolérée.

Exercice 03 - print_base10

fichier à rendre : `print_base10.c`

Implémentez la fonction suivante :

```
int e89_print_base10(int nbr);
```

Elle :

- doit afficher le nombre `nbr` sur la sortie standard du terminal ;
- renvoie le nombre de caractères qu'elle a écrit au total ;
- fonctionne sur des nombres positifs et négatifs.

L'utilisation de `pow` est conseillée, mais vous n'avez pas le droit à la `libm` (bibliothèque mathématique.) Vous êtes encouragés à réécrire `e89_pow`.

Fonctions autorisées :

- `malloc`,
- `free`,
- `write`.