CODING CLUB

EPITECH LYON

CPool Day -1

VERSION 1.2.7

DIVERSITY by EPITECH

Prémices

L'objectif de ce Coding club est de vous familiariser avec le langage C et de vous offrir un avant-goût de la forme mais aussi de la rigueur de programmation qui vous sera demandée en octobre prochain

Et pour que votre périple à travers la journée -1 de votre piscine de C se passe à merveille il y a quelques règles à respecter !

Règles :

- Chaque exercice a été pensé afin d'être réalisable avec pour **seul syscall la fonction write()** (sauf exceptions mentionnées) de la libraire **unistd**. Toute fonction externe utilisée dans votre exercice [*exemple printf(), puts()*] déclenchera une erreur dans notre moulinette et refusera catégoriquement la validation de votre exercice (aussi juste soit-il).
- Vous devez organiser votre travail dans une arborescence de dossiers cohérente afin de faciliter le fonctionnement de la moulinette. Au début de chaque exercice le nom du fichier à rendre ainsi que le dossier de rendu vous seront communiqués.

Exemple d'arborescence:

```
.
├── my_name
│   └── my_name.c
└── digits
    └── digits.c
```

- Si vous avez une question, demandez à votre voisin de droite. Si malgré tous les efforts de votre cher voisin le problème persiste, rabattez-vous par défaut sur votre voisin de gauche. Si vous êtes en bout de table et donc dans l'impossibilité d'avoir un voisin de droite ou de gauche il vous reste toujours celui de devant !
- Si malgré cela une question reste en suspens ou un problème vous ennuie un peu trop, n'hésitez pas à appeler un cobra ils sont là pour ça !
- Tout au long de ces exercices, nous vous demanderons de réaliser des **fonctions**. Cela signifie que notre moulinette amènera son propre main() pour tester votre travail.

Si par hasard vous laissez votre fonction main() il se peut que cela ne marche pas et il se peut que cela soit de votre faute !

> Vous ne savez pas ce qu'est une fonction main() ? Google it ;)

Moulinette

Tout au long de cette journée les cobras se chargeront de lancer des scripts de correction vous permettant de savoir si votre exercice est correct ou non.

Vous pourrez alors consulter votre résultat sous la forme suivante:

```
[Nom du test]:  PASSED ou FAILED
```

> PASSED signifie que vous avez validé le test.
> FAILED signifie tout logiquement l'inverse.

Ce script, que nous appellerons moulinette, est un programme qui va venir corriger votre travail.

Il a été réalisé avec soin et ne peut pas avoir tort. Si un exercice fonctionne chez vous mais que la moulinette dit le contraire, essayez d'imaginer tous les cas possibles en suivant les noms de tests pour deviner. (On fait même des devinettes dans le Coding club trop cool !)

Si vraiment vous avez l'impression que la moulinette se trompe. *(NB: Hautement improbable)*

Ou bien que malgré **toute votre réflexion** un test reste un mystère absolu pour vous, n'hésitez pas à appeler un Cobra : ils sont là pour ça :)

Prêt ?

Si tout ceci est clair comme de l'eau de roche et bien je crois qu'il est grand temps pour vous de commencer votre avant premier jour de piscine !

À vos marques... prêts... **Codez !**

My_Name

Fichier de rendu: `my_name/my_name.c`

Sujet :

L'objectif de cet exercice est d'écrire une fonction qui affichera votre prénom (**suivi d'un retour à la ligne**) dans le terminal.

La fonction devra être prototypée de la façon suivante :

```
void my_name(void);
```

Exemple :

Un appel de votre fonction devra afficher ces résultats :

```
my_name();
```

```
$> Timothée
```

Comment faire pour afficher un retour à la ligne déjà ?

Digits

Fichier de rendu: `digits/digits.c`

Sujet :

Écrivez une fonction qui affiche dans le terminal tous les chiffres dans l'ordre croissant **suivi d'un retour à la ligne**.

La fonction devra être prototypée de la façon suivante :

```
void digits(void);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
digits();
```

```
$> 0123456789
```

Revalpha

Fichier de rendu: `revalpha/revalpha.c`

Sujet :

Écrivez une fonction qui affiche dans le terminal l'alphabet à l'envers, une lettre sur deux en majuscules, **suivi d'un retour à la ligne**

La fonction devra être prototypée de la façon suivante :

```
void revalpha(void);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
revalpha();
```

```
$> zYxWvUtSrQpOnMlKjIhGfEdCbA
```

Alandroit

Fichier de rendu: `alandroit/alandroit.c`

Sujet :

Écrivez une fonction qui affiche dans le terminal la chaîne de caractères passée en paramètre de la fonction.

La fonction devra être prototypée de la façon suivante :

```
void alandroit(char *str);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
alandroit("EPITECH");
```

```
$> EPITECH
```

```
alandroit("BEEP BOOP");
```

```
$> BEEP BOOP
```

> Vous avez le droit à la fonction `strlen` !

Alanver

Fichier de rendu: `alanver/alanver.c`

Sujet :

Écrivez une fonction qui affiche à l'envers dans le terminal la chaîne de caractères passée en paramètre de la fonction.

La fonction devra être prototypée de la façon suivante :

```
void alanver(char *str);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
alanver("EPITECH");
```

```
$> HCETIPE
```

```
alanver("BEEP BOOP");
```

```
$> POOB PEEB
```

```
alanver("bob");
```

```
$> bob
```

Humm... Il faudrait sûrement écrire la chaîne de caractères en partant de la fin, mais comment savoir où est la fin ?

Countoc

Fichier de rendu: `countoc/countoc.c`

Sujet :

Écrivez une fonction qui renvoie le nombre d'occurrences du caractère **'to_find'** trouvées dans la chaîne de caractères **'str'**.

La fonction devra être prototypée de la façon suivante :

```
int countoc(char *str, char to_find);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
printf("%d\n", countoc("EPITECH", 'E'));
```

```
$> 2
```

```
printf("%d\n",countoc("##3..##.##....###4#", '.'));
```

```
$> 7
```

⚠ Bien que `printf` soit utilisable à tout moment pour tester vos fonctions, cette fonction reste bien évidemment interdite et sera flag par la moulinette.

Repeat_alpha

Fichier de rendu: `repeat_alpha/repeat_alpha.c`

Sujet :

Pour cet exercice je vous laisse deviner le sujet par vous-même !

Vous verrez, c'est pas sorcier !

La fonction devra être prototypée de la façon suivante :

```
void repeat_alpha(char *str);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
repeat_alpha("abc");
```

```
$> abbccc
```

```
repeat_alpha("Alex");
```

```
$> Allllllllllleeeeexxxxxxxxxxxxxxxxxxxxxxxx
```

```
repeat_alpha("abacadaba 42!");
```

```
$> abbacccaddddabba 42!
```

Si vous n'arrivez vraiment pas à trouver une corrélation entre les exemples et après avoir bien réfléchi, vous pouvez appeler un cobra (On n'oublie pas aussi la règle avec les voisins ;p)

HidenTF

Fichier de rendu: `hidentf/hidentf.c`

Sujet :

Écrivez une fonction qui renvoie 1 si toutes les lettres de s1 se cachent dans s2 et 0 dans le cas inverse.

La fonction devra être prototypée de la façon suivante :

```
int hidentf(char *s1, char *s2);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
printf("%d",hidentf("abc", "abcdefghjikqsdblablabla"));
```

```
$>  1
```

```
printf("%d",hidentf("fgex", "tyf34gdgf;'ektufjhgdgex.;.;rtjynur6"));
```

```
$> 1
```

```
printf("%d",hidentf("azbc", "btarc"));
```

```
$> 0
```

Rot_42

Fichier de rendu: `rot_42/rot_42.c`

Sujet :

Écrivez une fonction qui prend une chaîne de caractères alphabétiques en paramètre et va incrémenter chacune de ses lettres de 42 puis l'afficher dans le terminal **suivi d'un retour à la ligne**. Si le caractère dépasse l'alphabet, on repart au début.

La fonction devra être prototypée de la façon suivante :

```
void rot_42(char *str);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
rot_42("abc");
```

```
$> qrs
```

```
rot_42("abcdefghijklmnopqrstuvwxyz");
```

```
$> qrstuvwxyzabcdefghijklmnop
```

```
rot_42("Vive le coding club 2022");
```

```
$> Lylu bu setydw sbkr 2022
```

N.b: Une lettre revient à 'a' lorsqu'elle dépasse 'z' et à 'A' lorsqu'elle dépasse 'Z'

Last_word

fichier de rendu : `last_word/last_word.c`

Sujet :

Écrirez une fonction qui prend une chaîne de caractères en paramètre, et qui affiche le dernier mot de cette chaîne, **suivi d'un '\n'**.

On appelle "mot" une portion de chaîne de caractères délimitée soit par des espaces et/ou des tabulations, soit par le début / la fin de la chaîne.

La fonction devra être prototypée de la façon suivante :

```
void last_word(char *str);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
last_word("EPITECH C'EST PLUTOT COOL QUAND MEME");
```

```
$> MEME
```

```
last_word("Bientot      vous           serez     des     tek    1     ");
```

```
$> 1
```

```
last_word("       lorem,ipsum      ");
```

```
$> lorem,ipsum
```

R_capitalize

Fichier de rendu: `r_capitalize/r_capitalize.c`

Sujet :

Écrivez une fonction reproduisant le comportement de la fonction **"capitalize" du JavaScript**, puis imprimez la chaîne modifiée dans le terminal.

La fonction prendra 2 paramètres, la chaîne à modifier et un entier définissant le type de transformation.

- Si order vaut 0 vous devez mettre en majuscules les premières lettres de chaque mot et passer en minuscule les autres.
- Si order vaut 1 vous devez procéder de la même manière mais en appliquant la majuscule à la dernière lettre.

Si order ne vaut ni 0 ni 1 vous devez simplement afficher un **'\n'**

Elle sera prototypée de la façon suivante :

```c
void r_capitalize(char *str, int order);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```c
r_capitalize("EPITECH C'EST PLUTOT COOL QUAND MEME !", 0);
```

```
$> Epitech C'est Plutot Cool Quand Meme !
```

```c
r_capitalize("Epitech C'est Plutot Cool Quand Meme !", 1);
```

```
$> epitecH c'esT plutoT cooL quanD memE !
```

Vous y êtes presque !

Toutes mes félicitations pour être arrivé jusqu'ici !

Que vous ayez tout réussi, skip un exercice sur 2 ou bien qu'il soit 10h10 et que tu sois en train de lire tout le pdf par simple curiosité tu es le ou la bienvenue ici !

Il ne reste plus que 2 exercices, Au bout !

Bon courage pour la suite jeune Tek 0 !

Pgcd

Fichier de rendu: `pgcd/pgcd.c`

Sujet :

Écrivez une fonction qui renvoie le plus grand dénominateur commun entre 2 nombres.

La fonction devra être prototypée de la façon suivante :

```
int pgcd(int nb1, int nb2);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
printf("%d",pgcd(42, 10))
```

```
$> 2
```

```
printf("%d",pgcd(14, 77))
```

```
$> 7
```

```
printf("%d",pgcd(17, 3))
```

```
$>1
```

Fprime

Fichier de rendu: `fprime/fprime.c`

Sujet :

Écrirez une fonction qui prend en paramètre un entier strictement positif, et qui affiche sa décomposition en facteurs premiers dans le terminal, suivie d'un '\n'.

Les facteurs doivent être affichés dans l'ordre croissant et séparés par des '*', de telle sorte que l'expression affichée donne le bon résultat.

La fonction devra être prototypée de la façon suivante :

```
void fprime(int nb);
```

Exemple :

Un appel de votre fonction avec les arguments suivants devra afficher ces résultats :

```
fprime(6)
```

```
$>2*3
```

```
fprime(225225)
```

```
$> 3*3*5*5*7*11*13
```

```
fprime(8333325)
```

```
$> 3*3*5*5*7*11*13*37
```

```
fprime(-1)
```

```
$> -1
```