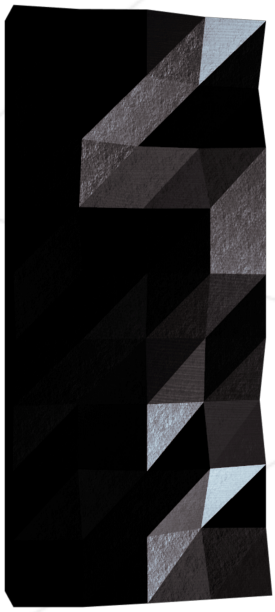

Piscine C

Jour 10

Staff 42 piscine@42.fr

Résumé: Ce document est le sujet du jour 10 de la piscine C de 42.

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : Makefile	5
IV	Exercice 01 : ft_foreach	6
V	Exercice 02 : ft_map	7
VI	Exercice 03 : ft_any	8
VII	Exercice 04 : ft_count_if	9
VIII	Exercice 05 : ft_is_sort	10
IX	Exercice 06 : do-op	11
X	Exercice 07 : ft_sort_wordtab	13
XI	Exercice 08 : ft_advanced_sort_wordtab	14
XII	Exercice 09 : ft_advanced_do-op	15

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices.
- Vos exercices seront corrigés par vos camarades de piscine.
- En plus de vos camarades, vous serez corrigés par un programme appelé la Moulinette.
- La Moulinette est très stricte dans sa notation. Elle est totalement automatisée. Il est impossible de discuter de sa note avec elle. Soyez d'une rigueur irréprochable pour éviter les surprises.
- La Moulinette n'est pas très ouverte d'esprit. Elle ne cherche pas à comprendre le code qui ne respecte pas la Norme.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de `-42`.
- Si ft_putchar() est une fonction autorisée, nous compilerons avec notre `ft_putchar.c`.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- La Moulinette compile avec les flags -Wall -Wextra -Werror.
- Si votre programme ne compile pas, vous aurez `0`.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.

- Votre manuel de référence s'appelle `Google / man / Internet / ...`
- Pensez à discuter sur le forum Piscine de votre Intra !
- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet...
- Réfléchissez. Par pitié, par Odin ! Nom d'une pipe.

Chapitre II

Préambule

Citation issue du film `V pour Vendetta` :

`Voilà ! Vois en moi l'image d'un humble Vétéran de VaudeVille, distribué Vicieusement dans les rôles de Victime et de Vilain par les Vicissitudes de la Vie. Ce Visage, plus qu'un Vil Vernis de Vanité, est un Vestige de la Vox populi aujourd'hui Vacante, éVanouie. Cependant, cette Vaillante Visite d'une Vexation passée se retrouVe ViVifiée et a fait Vœu de Vaincre cette Vénale et Virulente Vermine Vantant le Vice et Versant dans la Vicieusement Violente et Vorace Violation de la Volition. Un seul Verdict : la Vengeance. Une Vendetta telle une offrande VotiVe mais pas en Vain car sa Valeur et sa Véracité Viendront un jour faire Valoir le Vigilant et le Vertueux. En Vérité ce Velouté de Verbiage Vire Vraiment au Verbeux, alors laisse-moi simplement ajouter que c'est un Véritable honneur que de te rencontrer.`

`Appelle-moi V.`

`Avoid Aliterations. Always.`

.

Chapitre III

Exercice 00 : Makefile

	Exercice : 00
Makefile	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `Makefile`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire le `Makefile` qui compile votre `libft.a`.
- Le `Makefile` ira chercher les fichiers sources dans le dossier srcs.
- Le `Makefile` ira chercher les fichiers headers dans le dossier includes.
- La lib sera à la racine de l'exercice.
- Le `Makefile` devra egalement implémenter des règles `clean`, `fclean` et `re` en plus de la règle `all`.
- La règle `fclean` fait l'équivalent d'un make clean et efface aussi le binaire crée lors du make. La règle `re` fait l'équivalent d'un make fclean puis un make
- Nous ne ramasserons que votre Makefile et testerons avec nos fichiers. Dans le cadre de cet exercice, ne gérez que les 5 fonctions obligatoires pour votre lib (`ft_putchar`, `ft_putstr`, `ft_strcmp`, `ft_strlen` et `ft_swap`).

`Attention aux wildcards!`

Chapitre IV

Exercice 01 : ft_foreach

	Exercice : 01
ft_foreach	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `ft_foreach.c`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire une fonction `ft_foreach` qui, pour un tableau d'entiers donné, appliquera une fonction sur tous les éléments de ce tableau. Cette fonction sera appliquée dans l'ordre du tableau.
- La fonction sera prototypée de la manière suivante :

```
void        ft_foreach(int *tab, int length, void(*f)(int));
```

- Par exemple, la fonction `ft_foreach` pourra être appelée de la façon suivante pour afficher l'ensemble des entiers du tableau :

```
ft_foreach(tab, 1337, &ft_putnbr);
```

Chapitre V

Exercice 02 : ft_map

	Exercice : 02
ft_map	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `ft_map.c`	
Fonctions Autorisées : `malloc`	
Remarques : `n/a`	

- Écrire une fonction `ft_map` qui, pour un tableau d'entiers donné, appliquera une fonction sur tous les éléments de ce tableau (dans l'ordre) et retournera un tableau de toutes les valeurs de retour. Cette fonction sera appliquée dans l'ordre du tableau.
- La fonction sera prototypée de la manière suivante :

```
int        *ft_map(int *tab, int length, int(*f)(int));
```

Chapitre VI

Exercice 03 : ft_any

	Exercice : 03
ft_any	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `ft_any.c`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire une fonction `ft_any` qui renverra 1 si, en le passant à la fonction `f`, au moins un élément du tableau renvoie 1, 0 sinon.
- La fonction sera prototypée de la manière suivante :

```
int                ft_any(char **tab, int(*f)(char*));
```

- Le tableau sera délimité par 0.

Chapitre VII

Exercice 04 : ft_count_if

	Exercice : 04
ft_count_if	
Dossier de rendu : *ex*04/	
Fichiers à rendre : `ft_count_if.c`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire une fonction `ft_count_if` qui renverra le nombre d'éléments du tableau qui, en le passant à la fonction `f`, renvoient 1.
- La fonction sera prototypée de la manière suivante :

```
int                ft_count_if(char **tab, int(*f)(char*));
```

- Le tableau sera délimité par 0.

Chapitre VIII

Exercice 05 : ft_is_sort

	Exercice : 05
ft_is_sort	
Dossier de rendu : *ex*05/	
Fichiers à rendre : `ft_is_sort.c`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire une fonction `ft_is_sort` qui renverra 1 si le tableau est trié et 0 dans le cas contraire.
- La fonction passée en paramètre renverra un entier négatif si le premier argument est inférieur au deuxième, 0 s'ils sont égaux et un entier positif autrement.
- La fonction sera prototypée de la manière suivante :

```
int		ft_is_sort(int *tab, int length, int(*f)(int, int));
```

Chapitre IX

Exercice 06 : do-op

	Exercice : 06
do-op	
Dossier de rendu : $ex06/$	
Fichiers à rendre : `Makefile, et les fichiers de votre programme`	
Fonctions Autorisées : `write`	
Remarques : `n/a`	

- Écrire un programme qui s'appelle `do-op`.
- Le programme devra être lancé avec trois arguments : `do-op valeur1 operateur valeur2`
- Exemple :

```
$>./do-op 42 "+" 21
63
$>
```

- Le caractère `operateur` correspondra à la fonction appropriée dans un tableau de pointeurs sur fonction.
- Votre répertoire comportera un `Makefile` avec une règle `all` et une règle `clean`.
- Dans le cas d'une expression fausse comme `./do-op foo devide bar`, le programme affiche `0`.
- Si le nombre d'arguments n'est pas correct, `do-op` n'affiche rien.

- Voici un exemple de tests de la Moulinette :

```
$> make clean
$> make
$> ./do-op
$> ./do-op 1 + 1
2
$> ./do-op 42amis - -20toto12
62
$> ./do-op 1 p 1
0
$> ./do-op 1 + toto3
1
$>
$> ./do-op toto3 + 4
4
$> ./do-op foo plus bar
0
$> ./do-op 25 / 0
Stop : division by zero
$> ./do-op 25 % 0
Stop : modulo by zero
$>
```

Chapitre X

Exercice 07 : ft_sort_wordtab

	Exercice : 07
ft_sort_wordtab	
Dossier de rendu : $ex07/$	
Fichiers à rendre : `ft_sort_wordtab.c`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire la fonction `ft_sort_wordtab` qui trie par ordre `ascii` les mots obtenus grâce à `ft_split_whitespaces`.
- Le tri s'effectuera en échangeant les pointeurs du tableau.
- Elle devra être prototypée de la façon suivante :

```
void ft_sort_wordtab(char **tab);
```

Chapitre XI

Exercice 08 : ft_advanced_sort_wordtab

	Exercice : 08
ft_advanced_sort_wordtab	
Dossier de rendu : *ex*08/	
Fichiers à rendre : `ft_advanced_sort_wordtab.c`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire la fonction `ft_advanced_sort_wordtab` qui trie, en fonction du retour de la fonction passée en paramètre, les mots obtenus grâce à `ft_split_whitespaces`.
- Le tri s'effectuera en échangeant les pointeurs du tableau.
- Elle devra être prototypée de la façon suivante :

```
void ft_advanced_sort_wordtab(char **tab, int(*cmp)(char *, char *));
```

Un appel à ft_advanced_sort_wordtab() avec en second paramètre ft_strcmp donnera le même resultat que ft_sort_wordtab().

Chapitre XII

Exercice 09 : ft_advanced_do-op

	Exercice : 09
ft_advanced_do-op	
Dossier de rendu : *ex*09/	
Fichiers à rendre : `Makefile, et les fichiers de votre programme`	
Fonctions Autorisées : `write`	
Remarques : `n/a`	

- Écrire un programme fonctionnant exactement comme le do-op à un détail près : vous devez inclure le fichier `ft_opp.h` qui definira quel pointeur sur fonction correspond à quel caractère.

```
$>cat ft_opp.h
#ifndef __FT_OPP_H__
# define __FT_OPP_H__

t_opp gl_opptab[] =
{
    {"-", &ft_sub},
    {"+", &ft_add},
    {"*", &ft_mul},
    {"/", &ft_div},
    {"%", &ft_mod},
    {"", &ft_usage}
};

#endif /* __FT_OPP_H__ */
$>
```

- Vous devez créer au moins 6 fonctions : ft_add, ft_sub, ft_mul, ft_div, ft_mod, ft_usage.

- ft_usage affiche les caractères possibles (définis dans ft_opp.h) comme dans l'exemple ci dessous :

```
$> make clean
$> make
$> ./ft_advanced_do-op
$> ./ft_advanced_do-op 1 + 1
2
$> ./ft_advanced_do-op 1 p 1
error : only [ - + * / % ] are accepted.
$> ./ft_advanced_do-op 1 + toto3
1
$> ./ft_advanced_do-op 25 / 0
Stop : division by zero
$> ./ft_advanced_do-op 25 % 0
Stop : modulo by zero
$>
```

- Vous devez définir le type t_opp correspondant à la structure s_opp permettant la compilation de votre projet.
- N'écrivez RIEN dans le fichier `ft_opp.h`, ni même la définition de t_opp. Incluez vos propres fichiers également si nécessaire.
- N'affichez une erreur que pour les operateurs n'ayant pas de correspondance dans `ft_opp.h`.
- Pensez, pour tous les points précédents, que nous changerons surement le fichier `ft_opp.h`...

`Un operateur peut être composé de plusieurs caractères.`