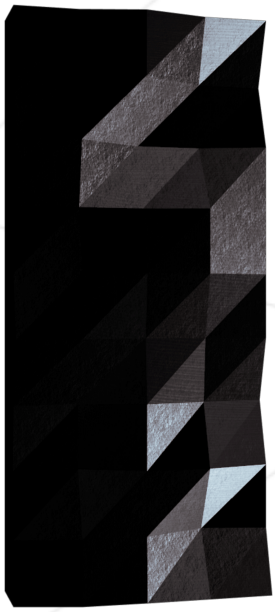

Piscine Reloaded

It's good to be back

Résumé: La piscine c'était bien, mais le temps a passé. Cette petite série d'exercices simples va vous permettre de remettre le pied à létrier pour bien commencer votre scolarité à 42. Fonctions, boucles, pointeurs, structures, retrouvons ensemble les bases syntaxiques et sémantiques du `C`.

Table des matières

Chapitre I

Preambule

Voici ce que Wikipédia dit sur `Edward Snowden` :

Edward Joseph Snowden, né le 21 juin 1983 à Elizabeth City, Caroline du Nord, est un informaticien américain, ancien employé de la Central Intelligence Agency (CIA) et de la National Security Agency (NSA), qui a révélé les détails de plusieurs programmes de surveillance de masse américains et britanniques.

À partir du 6 juin 2013, Snowden rend publiques par l'intermédiaire des médias, notamment The Guardian et The Washington Post, des informations classées top-secrètes de la NSA concernant la captation des métadonnées des appels téléphoniques aux États-Unis, ainsi que les systèmes d'écoute sur internet des programmes de surveillance PRISM, XKeyscore, Boundless Informant et Bullrun du gouvernement américain et les programmes de surveillance Tempora, Muscular et Optic Nerve du gouvernement britannique.

Pour justifier ses révélations, il a indiqué que son "seul objectif est de dire au public ce qui est fait en son nom et ce qui est fait contre lui.“.

Je vous invite à regarder ce reportage de `HBO` sur Edward ici.

Chapitre II

Introduction

La Piscine `Reloaded` est un best-of des notions vues pendant la piscine `C` pour vous permettre de vous remettre à la programmation en `C` dans un environnement moins hostile. Ces quelques exercices sont à réaliser en entier et parfaitement pour débloquer l'accès au projet suivant.

Si vous avez deja réussi à faire certains de ces exercices pendant la piscine, nous vous recommandons très vivement de ne pas céder à la tentation de récupérer votre ancien code. L'apprentissage de la programmation passe par la pratique et rendre un code existant n'a aucun intéret.

Chapitre III

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Attention aux droits de vos fichiers et dossiers.
- Vous devez suivre la procédure de rendu pour tous vos exercices.
- Ce projet sera corrigé uniquement par `Deepthoughts`. Une fois n'est pas coutume, il n'y aura pas de peer-évaluation pour ce projet. Vous devez donc respecter les noms des fichiers et de fonctions au caractère prèt.
- `Deepthoughts` 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.
- Commencez par lire le sujet en entier. Lisez également attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet.
- Lorsque le sujet d'un exercice demande de réaliser "une fonction", cela signifie que vous ne devez pas inclure de fonction `main` dans votre code car `Deepthoughts` utilisera la sienne. Si au contraire le sujet d'un exercice vous demande de réaliser "un programme", vous devez inclure une fonction `main` dans votre, et `Deepthoughts` compilera ce code en respectant les contraintes listées dans ce sujet.
- Vous devez coder en `C` et à la Norme. La norminette fait foi.
- Vous devez toujours utiliser la commande `gcc` avec les flags de compilation `-Wall`, `-Wextra` et `-Werror` pour compiler. `Deepthoughts` en fera de même.
- Si votre programme ne compile pas, vous aurez `0`.
- Les exercices shell doivent s'exécuter avec /bin/`sh`.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices.
- 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 avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche. Sinon, vous pouvez poser vos questions sur le forum ou sur slack.
- Réfléchissez. Par pitié, par Odin ! Nom d'une pipe.

Chapitre IV

Exercice 00 : Owi, encore...

	Exercice : 00
Owi, encore...	
Dossier de rendu : *ex*00/	
Fichiers à rendre : `exo.tar`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Créez tous ces fichiers et répertoires. Faites le nécessaire pour que l'affichage d'un `ls -l` dans votre répertoire ressemble à cela :

```
$> ls -l
total 42
drwx--xr-x  2 login  wheel  XX Jun  1 20:47 test0
-rwx--xr--  1 login  wheel   4 Jun  1 21:46 test1
dr-x---r--  2 login  wheel  XX Jun  1 22:45 test2
-r-----r--  2 login  wheel   1 Jun  1 23:44 test3
-rw-r----x  1 login  wheel   2 Jun  1 23:43 test4
-r-----r--  2 login  wheel   1 Jun  1 23:44 test5
lrwxr-xr-x  1 login  wheel   5 Jun  1 22:20 test6 -> test0
$>
```

- Pour les heures, il sera toléré que ce soit l'année qui s'affiche si la date de l'exercice (1 juin) est dépassée de six mois ou plus.

- Une fois l'exercice résolu, vous exécuterez la commande `tar -cf exo.tar *` pour créer le fichier à rendre.

`"login" et "wheel" seront remplacés respectivement par votre login et votre groupe.`

Vous ne pourrez pas imiter la ligne "total 42".

Les XX ne seront pas pris en compte.

Chapitre V

Exercice 01 : Z

	Exercice : 01
Les vrais savent afficher un Z	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `z`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Créez un fichier `z` qui, lorsque l'on fait un `cat` dessus, affiche `"Z"` suivi d'un retour à la ligne.

```
$>cat z
Z
$>
```

Chapitre VI

Exercice 02 : Clean

	Exercice : 02
Find And Clean It	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `clean`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Placez dans un fichier `clean` une ligne de commande qui va rechercher, à partir du répertoire courant et dans tous ses sous-répertoires, les fichiers dont le nom se termine par ~, ou commence et se termine par `#`.
- La ligne de commande affichera et effacera les fichiers trouvés.
- Une seule commande est autorisée : pas de `';'`, de `'&&'` ou autres.

```
man find
```

Chapitre VII

Exercice 03 : find_sh

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

- Écrire une ligne de commande qui cherche dans le répertoire courant et dans tous ses sous-répertoires tous les fichiers dont le nom se termine par ".sh" (sans les guillemets) et n'affiche que leurs noms, sans le .sh.
- Exemple de sortie :

```
$>./find_sh.sh | cat -e
find_sh$
file1$
file2$
file3$
$>
```

Chapitre VIII

Exercice 04 : MAC

	Exercice : 04
MAC	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `MAC.sh`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire une ligne de commande qui affiche les adresses MAC de votre machine. Chaque adresse sera suivi d'un retour à la ligne.

```
man ifconfig
```

Chapitre IX

Exercice 05 : Can you create it ?

	Exercice : 05
Can you create it ?	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `"\?$*'KwaMe'*$?\"`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Créer un fichier contenant uniquement "42" et **RIEN** d'autre.
- Il se nommera :

```
"\?$*'KwaMe'*$?\"
```

- Exemple :

```
$>ls -lRa *waM* | cat -e
-rw---xr-- 1 75355 32015 2 Oct 2 12:21 "\?$*'KwaMe'*$?\"$
$>
```

Chapitre X

Exercice 06 : ft_print_alphabet

	Exercice : 06
ft_print_alphabet	
Dossier de rendu : $ex06/$	
Fichiers à rendre : `ft_print_alphabet.c`	
Fonctions Autorisées : `ft_putchar`	
Remarques : `n/a`	

- Écrire une fonction qui affiche l'alphabet en minuscule sur une seule ligne, dans l'ordre croissant, à partir de la lettre 'a'.
- Elle devra être prototypée de la façon suivante :

```
void ft_print_alphabet(void);
```

Chapitre XI

Exercice 07 : ft_print_numbers

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

- Écrire une fonction qui affiche tous les chiffres sur une seule ligne, dans l'ordre croissant.
- Elle devra être prototypée de la façon suivante :

```
void ft_print_numbers(void);
```

Chapitre XII

Exercice 08 : ft_is_negative

	Exercice : 08
ft_is_negative	
Dossier de rendu : $ex08/$	
Fichiers à rendre : `ft_is_negative.c`	
Fonctions Autorisées : `ft_putchar`	
Remarques : `n/a`	

- Écrire une fonction qui affiche `'N'` ou `'P'` suivant le signe de l'entier passé en paramètre. Si `n` est négatif, alors afficher `'N'`. Si `n` est positif ou nul, alors afficher `'P'`.
- Elle devra être prototypée de la façon suivante :

```
void ft_is_negative(int n);
```

Chapitre XIII

Exercice 09 : ft_ft

	Exercice : 09
ft_ft	
Dossier de rendu : *ex*09/	
Fichiers à rendre : `ft_ft.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire une fonction qui prend un pointeur sur `int` en paramètre et qui donne à cet `int` la valeur `42`.
- Elle devra être prototypée de la façon suivante :

```
void        ft_ft(int *nbr);
```

Chapitre XIV

Exercice 10 : ft_swap

	Exercice : 10
ft_swap	
Dossier de rendu : $ex10/$	
Fichiers à rendre : `ft_swap.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire une fonction qui échange le contenu de deux entiers dont les adresses sont données en paramètres.
- Elle devra être prototypée de la façon suivante :

```
void    ft_swap(int *a, int *b);
```

Chapitre XV

Exercice 11 : ft_div_mod

	Exercice : 11
ft_div_mod	
Dossier de rendu : *ex*11/	
Fichiers à rendre : `ft_div_mod.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire une fonction `ft_div_mod` qui a le prototypage suivant :

```
void    ft_div_mod(int a, int b, int *div, int *mod);
```

- Cette fonction divise les deux paramètres `a` par `b` et stocke le resultat dans l'`int` pointé par `div`.
  Elle stocke également le reste de la division de `a` et `b` dans l'int pointé par `mod`.

Chapitre XVI

Exercice 12 : ft_iterative_factorial

	Exercice : 12
ft_iterative_factorial	
Dossier de rendu : $ex12/$	
Fichiers à rendre : `ft_iterative_factorial.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire une fonction itérative qui renvoie un nombre. Ce nombre est le résultat de l'opération factorielle à partir du nombre passé en paramètre.
- En cas d'erreur, la fonction devra retourner `0`.
- Elle devra être prototypée de la façon suivante :

```
int ft_iterative_factorial(int nb);
```

- Votre fonction doit donner son résultat en moins de deux secondes.

Chapitre XVII

Exercice 13 : ft_recursive_factorial

	Exercice : 13
ft_recursive_factorial	
Dossier de rendu : $ex13/$	
Fichiers à rendre : `ft_recursive_factorial.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire une fonction récursive qui renvoie la factorielle du nombre passé en paramètre.
- La fonction doit donc s'appeler elle même et ne pas utiliser de boucle `while`.
- Elle doit gérer les mêmes cas que la fonction précédente.
- Elle devra être prototypée de la façon suivante :

```
int ft_recursive_factorial(int nb);
```

Chapitre XVIII

Exercice 14 : ft_sqrt

	Exercice : 14
ft_sqrt	
Dossier de rendu : $ex14/$	
Fichiers à rendre : `ft_sqrt.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire une fonction qui renvoie la racine carrée entière d'un nombre si elle existe, 0 si la racine carrée n'est pas entière.
- Elle devra être prototypée de la façon suivante :

```
int ft_sqrt(int nb);
```

- Votre fonction doit donner son résultat en moins de deux secondes.

Chapitre XIX

Exercice 15 : ft_putstr

	Exercice : 15
ft_putstr	
Dossier de rendu : $ex15/$	
Fichiers à rendre : `ft_putstr.c`	
Fonctions Autorisées : `ft_putchar`	
Remarques : `n/a`	

- Écrire une fonction qui affiche un à un les caractères d'une chaîne à l'écran.
- L'adresse du premier caractère de la chaîne est contenue dans le pointeur passé en paramètre à la fonction.
- Elle devra être prototypée de la façon suivante :

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

Chapitre XX

Exercice 16 : ft_strlen

	Exercice : 16
ft_strlen	
Dossier de rendu : $ex16/$	
Fichiers à rendre : `ft_strlen.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Reproduire à l'identique le fonctionnement de la fonction `strlen` (man strlen).
- Elle devra être prototypée de la façon suivante :

```
int ft_strlen(char *str);
```

Chapitre XXI

Exercice 17 : ft_strcmp

	Exercice : 17
ft_strcmp	
Dossier de rendu : $ex17/$	
Fichiers à rendre : `ft_strcmp.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Reproduire à l'identique le fonctionnement de la fonction `strcmp` (man strcmp).
- Elle devra être prototypée de la façon suivante :

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

Chapitre XXII

Exercice 18 : ft_print_params

	Exercice : 18
ft_print_params	
Dossier de rendu : $ex18/$	
Fichiers à rendre : `ft_print_params.c`	
Fonctions Autorisées : `ft_putchar`	
Remarques : `n/a`	

- Il s'agit ici d'un programme, vous devrez donc avoir une fonction `main` dans votre `fichier.c`.
- Écrire un programme qui affiche les arguments reçus en ligne de commande.
- Exemple :

```
$>./a.out test1 test2 test3
test1
test2
test3
$>
```

Chapitre XXIII

Exercice 19 : ft_sort_params

	Exercice : 19
ft_sort_params	
Dossier de rendu : $ex19/$	
Fichiers à rendre : `ft_sort_params.c`	
Fonctions Autorisées : `ft_putchar`	
Remarques : `n/a`	

- Il s'agit ici d'un programme, vous devrez donc avoir une fonction `main` dans votre `fichier.c`.
- Écrire un programme qui affiche les arguments reçus en ligne de commande triés par ordre ascii.
- Vous devez afficher tous les arguments, sauf `argv[0]`.
- Chaque argument doit être affiché sur une ligne différente.

Chapitre XXIV

Exercice 20 : ft_strdup

	Exercice : 20
ft_strdup	
Dossier de rendu : $ex20/$	
Fichiers à rendre : `ft_strdup.c`	
Fonctions Autorisées : `malloc`	
Remarques : `n/a`	

- Reproduire à l'identique le fonctionnement de la fonction `strdup` (man strdup).
- Elle devra être prototypée de la façon suivante :

```
char    *ft_strdup(char *src);
```

Chapitre XXV

Exercice 21 : ft_range

	Exercice : 21
ft_range	
Dossier de rendu : $ex21/$	
Fichiers à rendre : `ft_range.c`	
Fonctions Autorisées : `malloc`	
Remarques : `n/a`	

- Écrire une fonction `ft_range` qui retourne un tableau d'`int`. Ce tableau d'`int` contiendra toutes les valeurs entre `min` et `max`.
- `Min` inclu et `max` exclu.
- Elle devra être prototypée de la façon suivante :

```
int    *ft_range(int min, int max);
```

- Si la valeur `min` est supérieure ou égale à la valeur `max`, un pointeur nul sera retourné.

Chapitre XXVI

Exercice 22 : ft_abs.h

	Exercice : 22
ft_abs.h	
Dossier de rendu : $ex22/$	
Fichiers à rendre : `ft_abs.h`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire une **macro** `ABS` qui remplace son paramètre par sa valeur absolue :

```
#define ABS(Value)
```

On vous demande bien de faire quelque chose de normalement interdit par la norme, mais ce sera la seule entorse à la norme que l'on autorise pour cet exercice.

Chapitre XXVII

Exercice 23 : ft_point.h

	Exercice : 23
ft_point.h	
Dossier de rendu : *ex*23/	
Fichiers à rendre : `ft_point.h`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire un fichier `ft_point.h` qui fera compiler le main suivant :

```c
#include "ft_point.h"

void        set_point(t_point *point)
{
        point->x = 42;
        point->y = 21;
}

int                main(void)
{
        t_point                point;

        set_point(&point);
        return (0);
}
```

Chapitre XXVIII

Exercice 24 : Makefile

	Exercice : 24
Makefile	
Dossier de rendu : $ex24/$	
Fichiers à rendre : `Makefile`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire le `Makefile` qui compile une bibliothèque nommée `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 la cible crée lors du make, ici `libft.a`. 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 suivantes : `ft_putchar`, `ft_putstr`, `ft_strcmp`, `ft_strlen` et `ft_swap`.

`Attention aux wildcards !`

Chapitre XXIX

Exercice 25 : ft_foreach

	Exercice : 25
ft_foreach	
Dossier de rendu : $ex25/$	
Fichiers à rendre : `ft_foreach.c`	
Fonctions Autorisées : `Rien`	
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 XXX

Exercice 26 : ft_count_if

	Exercice : 26
ft_count_if	
Dossier de rendu : $ex26/$	
Fichiers à rendre : `ft_count_if.c`	
Fonctions Autorisées : `Rien`	
Remarques : `n/a`	

- Écrire une fonction `ft_count_if` qui renverra le nombre d'éléments du tableau qui, en les 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 XXXI

Exercice 27 : display_file

	Exercice : 27
display_file	
Dossier de rendu : $ex27/$	
Fichiers à rendre : `Makefile, et les fichiers de votre programme`	
Fonctions Autorisées : `close, open, read, write`	
Remarques : `n/a`	

- Écrire un programme appelé `ft_display_file` qui affiche sur la sortie standard uniquement le contenu du fichier passé en argument.
- Le répertoire de rendu aura un `Makefile` avec une règle `all`, une règle `clean`, et une règle `fclean`.
- La fonction `malloc` est explicitement interdite. Vous pouvez faire l'exercice uniquement en déclarant un tableau de taille fixe.
- Tous les fichiers passées en paramètre seront valides.
- Les messages d'erreurs devront être affichés sur la sortie leur étant réservée.

```
$> ./ft_display_file
File name missing.
$> ./ft_display_file Makefile
*contenu du Makefile*
$> ./ft_display_file Makefile display_file.c
Too many arguments.
$>
```

Chapitre XXXII

Rendu et peer-évaluation

Rendez votre travail sur votre dépôt `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué par `Deepthoughts`.

Je vous rappelle qu'exceptionnellement, ce projet sera corrigé uniquement par `Deepthoughts`. Il n'y aura donc pas de peer-évaluation.

La seule note acceptable pour réussir ce projet est 100%. Si `Deepthoughts` vous attribue une note inférieure, c'est que vous avez échoué et que vous devez réessayer de faire mieux en cliquant sur le bouton `Retry` de votre passage de projet.

Bon courage à tous et n'oubliez pas votre fichier auteur !