



Piscine C

C 00

Résumé: CE document est le sujet du module C 00 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_putchar	7
IV	Exercice 01 : ft_print_alphabet	8
V	Exercice 02 : ft_print_reverse_alphabet	9
VI	Exercice 03 : ft_print_numbers	10
VII	Exercice 04 : ft_is_negative	11
VIII	Exercice 05 : ft_print_comb	12
IX	Exercice 06 : ft_print_comb2	13
X	Exercice 07 : ft_putnbr	14
XI	Exercice 08 : ft_print_combn	15
XII	Rendu et peer-evaluation	16

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour ce module, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

La confiture de nouilles, selon Pierre Dac

Avant d'utiliser la nouille pour la confection de la confiture, il faut évidemment la récolter; avant de la récolter, il faut qu'elle pousse, et pour qu'elle pousse, il va de soi qu'il faut d'abord la semer.

Les semaines de la graine de nouille, c'est-à-dire les senouilles, représentent une opération extrêmement délicate. Tout d'abord, le choix d'un terrain propice à la fécondation de la nouille demande une étude judicieusement approfondie. Le terrain nouillifère type doit être, autant que possible, situé en bordure de la route départementale et à proximité de la gendarmerie nationale.

Avant de semer la graine de nouille, les nouilliculteurs préparent longuement le champ nouillifère pour le rendre idoine à la fécondation. Ils retournent la terre avec une charrue spéciale dont le soc est remplacé par une lame Gillette, ensuite délaissant les engrains chimiques, nettement contre-indiqués dans le cas présent, ils fument le champ nouillifère avec du fromage râpé. Cette opération s'effectue indifféremment avec une seringue ou une pompe à vélo.

Lorsque le champ est suffisamment imprégné de fromage râpé, on verse sur toute sa surface de l'alcool de menthe dans la proportion d'un verre à Bordeaux par hectare de superficie; cette opération qui est confiée à des spécialistes de l'École de Nouilliculture, est effectuée avec un compte-gouttes.

Après cela, on laisse fermenter la terre pendant toute la durée de la nouvelle lune et dès l'apparition du premier quartier, on procède alors aux senouilles de la graine de nouilles. Il ne faudrait pas vous imaginer, Mesdames et Messieurs, que la graine de nouilles est d'un commerce courant et qu'on la trouve communément chez les grainetiers ; si vous croyez cela, il est indiscutable que vous broutez les coteaux de l'erreur. La graine de nouilles ne s'obtient qu'après une très longue préparation de laboratoire, car elle est le produit d'un croisement de foie de veau avec le concombre adulte; voici d'ailleurs quelques précisions sur cette merveilleuse con]onction qui est la gloire de nos chimistes, dont la science n'a d'égale que la modestie.

On met côté à côté, dans une lessiveuse, une tranche de foie de veau et un

concombre adulte, on place le tout dans un autoclave et on l'y laisse 45 jours à une température de 120° sous la bienveillance d'un contrôleur de la Compagnie du Gaz; au bout de ce laps de temps, on ouvre l'appareil et on n'a plus qu'à recueillir la précieuse graine que l'on va verser dans la terre prête à la recevoir et qu'elle va féconder.

Les senouilles s'effectuent à l'aide d'un poêle mobile dans lequel est versée la graine, laquelle est projetée dans la terre par un dispositif spécial dont il ne nous est pas permis de révéler le secret pour des raisons de défense nationale que l'on comprendra aisément. Après ça, on arrose entièrement le champ avec des siphons d'eau de seltz, on sèche ensuite avec du papier buvard, on donne un coup de plumeau et on n'a plus qu'à s'en remettre au travail de la terre nourricière et à la nature immortelle, généreuse et démocratique. Lorsque les senouilles sont terminées, les nouilliculteurs qui sont encore entachés de superstition, consultent les présages; ils prennent une petite taupe, la font courir dans l'herbe et si elle fait : "ouh!" c'est que la récolte sera bonne; si elle ne fait pas "ouh!" c'est que la récolte sera bonne tout de même, mais comme cela les croyances sont respectées, et tout le monde est content.

Pendant la germination, il n'y a presque rien à faire ; tous les huit jours seulement, on arrose le champ avec de l'huile de cade, de la cendre de cigare, du jus de citron et de la glycérine pour éviter que la terre ne se crevasse.

Pendant la moisson, les nuits sont témoins de saines réjouissances auxquelles se livrent les travailleurs de la nouilliculture, la jeunesse danse et s'en donne à cœur joie aux sons d'un orchestre composé d'un harmonium, d'une mandoline et d'une trompette de cavalerie ; les jeunes gens revêtent leur costume régional composé d'une redingote, d'une culotte cycliste, d'espadrilles et d'un chapeau Cronstadt ; les jeunes filles, rougissantes de joie pudique, sont revêtues de ravissantes robes de toile à cataplasme, ornées d'empîcements en schpoutnoutz, et se ceignent le front d'une couronne d'œufs durs du plus gracieux effet Un feu d'artifice tiré avec des lampes Pigeon clôture la série des réjouissances et chacun rentre chez soi, content du labeur accompli, pour procéder alors à la confection de la confiture de nouilles, objet de la présente étude.

La nouille encore à l'état brut, est alors soigneusement triée et débarrassée de ses impuretés; après un premier stade, elle est expédiée à l'usine et passée immédiatement au laminouille qui va lui donner l'aspect définitif que nous lui connaissons - le laminouille est une machine extrêmement perfectionnée, qui marche au guignolet-cassis et qui peut débiter jusqu'à 80 kilomètres de nouilles à l'heure - ; à la sortie du laminouille, la nouille est passée au vernis cellulosique qui la rend imperméable et souple; elle est ensuite hachée menue à la hache d'abordage et râpée. Le râpage se fait encore à la main et avec une râpe à bois. Après le râpage, la nouille est alors mise en bouteilles, opération très délicate qui demande énormément d'attention ; on met ensuite les bouteilles dans un appareil appelé électronouille, dans lequel passe un courant de 210 volts; après un séjour de 12 heures dans cet appareil, les bouteilles sont

sorties et on vide la nouille désormais électrifiée dans un récipient placé lui-même sur un réchaud à alcool à haute tension.

On verse alors dans ledit récipient : du sel, du sucre, du poivre de Cayenne, du gingembre, de la cannelle, de l'huile, de la pomme de terre pilée, un flocon de magnésie bismurée, du riz, des carottes, des peaux de saucisson, des tomates, du vin blanc, et des piments rouges, on mélange lentement ces ingrédients avec la nouille à l'aide d'une cuiller à pot et on laisse mitonner à petit feu pendant 21 jours. La confiture de nouilles est alors virtuellement terminée. Lorsque les 21 jours sont écoulés, que la cuisson est parvenue à son point culminant et définitif, on place le récipient dans un placard, afin que la confiture se solidifie et devienne gélatineuse; quand elle est complètement refroidie, on soulève le récipient très délicatement, avec d'infinites précautions et le maximum de prudence et on balance le tout par la fenêtre parce que c'est pas bon!

Contrairement à la confiture de nouilles, le C c'est bon, mangez-en !

Chapitre III

Exercice 00 : ft_putchar

	Exercice : 00
	ft_putchar
Dossier de rendu :	<i>ex00/</i>
Fichiers à rendre :	ft_putchar.c
Fonctions Autorisées :	write

- Écrire une fonction qui affiche le caractère passé en paramètre.
- Elle devra être prototypée de la façon suivante :

```
void ft_putchar(char c);
```

Pour afficher le caractère, vous devez utiliser la fonction **write** de la manière suivante.

```
write(1, &c, 1);
```

Chapitre IV

Exercice 01 : ft_print_alphabet

	Exercice : 01
	ft_print_alphabet
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	ft_print_alphabet.c
Fonctions Autorisées :	write

- 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 V

Exercice 02 : ft_print_reverse_alphabet

	Exercice : 02
	ft_print_reverse_alphabet
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : ft_print_reverse_alphabet.c	
Fonctions Autorisées : write	

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

```
void ft_print_reverse_alphabet(void);
```

Chapitre VI

Exercice 03 : ft_print_numbers

	Exercice : 03
	ft_print_numbers
Dossier de rendu :	<i>ex03/</i>
Fichiers à rendre :	ft_print_numbers.c
Fonctions Autorisées :	write

- 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 VII

Exercise 04 : ft_is_negative

	Exercice : 04
	ft_is_negative
Dossier de rendu :	<i>ex04/</i>
Fichiers à rendre :	ft_is_negative.c
Fonctions Autorisées :	write

- 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 VIII

Exercice 05 : ft_print_comb

	Exercice : 05
	ft_print_comb
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : ft_print_comb.c	
Fonctions Autorisées : write	

- Écrire une fonction qui affiche, dans l'ordre croissant, toutes les différentes combinaisons de trois chiffres différents dans l'ordre croissant - oui, la répétition est volontaire.
- Cela donne quelque chose comme ça :

```
$>./a.out | cat -e  
012, 013, 014, 015, 016, 017, 018, 019, 023, ..., 789$>
```

- 987 n'est pas là car 789 est déjà présent
- 999 n'est pas là car ce nombre ne comporte pas exclusivement des chiffres différents les uns des autres
- Elle devra être prototypée de la façon suivante :

```
void ft_print_comb(void);
```

Chapitre IX

Exercice 06 : ft_print_comb2

	Exercice : 06
	ft_print_comb2
Dossier de rendu :	<i>ex06/</i>
Fichiers à rendre :	ft_print_comb2.c
Fonctions Autorisées :	write

- Écrire une fonction qui affiche toutes les différentes combinaisons de deux nombres entre 0 et 99, dans l'ordre croissant.
- Cela donne quelque chose comme ça :

```
$>./a.out | cat -e  
00 01, 00 02, 00 03, 00 04, 00 05, ..., 00 99, 01 02, ..., 97 99, 98 99$>
```

- Elle devra être prototypée de la façon suivante :

```
void ft_print_comb2(void);
```

Chapitre X

Exercice 07 : ft_putnbr

	Exercice : 07
	ft_putnbr
Dossier de rendu :	<i>ex07/</i>
Fichiers à rendre :	ft_putnbr.c
Fonctions Autorisées :	write

- Écrire une fonction qui affiche un nombre passé en paramètre. La fonction devra être capable d'afficher la totalité des valeurs possibles dans une variable de type **int**.
- Elle devra être prototypée de la façon suivante :

```
void ft_putnbr(int nb);
```

- Par exemple :
 - `ft_putnbr(42)` affiche "42".

Chapitre XI

Exercice 08 : ft_print_combn

	Exercice : 08
	ft_print_combn
Dossier de rendu :	<i>ex08/</i>
Fichiers à rendre :	ft_print_combn.c
Fonctions Autorisées :	write

- Écrire une fonction qui affiche toutes les différentes combinaisons de **n** chiffres dans l'ordre croissant.
- **n** sera tel que : $0 < n < 10$.
- Si $n = 2$, cela donne quelque chose comme ça :

```
$>./a.out | cat -e  
01, 02, 03, ..., 09, 12, ..., 79, 89$>
```

- Elle devra être prototypée de la façon suivante :

```
void ft_print_combn(int n);
```

Chapitre XII

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 01

Résumé: Ce document est le sujet du module C 01 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_ft	5
IV	Exercice 01 : ft_ultimate_ft	6
V	Exercice 02 : ft_swap	7
VI	Exercice 03 : ft_div_mod	8
VII	Exercice 04 : ft_ultimate_div_mod	9
VIII	Exercice 05 : ft_putstr	10
IX	Exercice 06 : ft_strlen	11
X	Exercice 07 : ft_rev_int_tab	12
XI	Exercice 08 : ft_sort_int_tab	13
XII	Rendu et peer-evaluation	14

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour cette journée, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

Le jeu du Sirop selon Perceval, tiré de la série *Kaamelott* :

"Bon, j'veais vous apprendre les règles simplifiées, parce que les vraies règles, elles sont velues. Bon, le seul truc, c'est que normalement, ça se joue à trois. Mais c'est pas grave on va se débrouiller.

Le principe, c'est de faire des valeurs. Donc là, mettons, on est trois, il y a trois valeurs à distribuer. On va dire, sirop de huit, sirop de quatorze et sirop de vingt-et-un. Vous occupez pas des sirops tout de suite. Ce qu'il faut comprendre d'abord, c'est les valeurs. Si vous lancez une valeur en début de tour, mettons un sirop de huit, pour commencer petit, les autres ont le choix entre laisser filer la mise ou relancer un sirop de quatorze. On tourne dans le sens des valeurs. C'est pour ça, il faut bien comprendre le système des valeurs; après, ça va tout seul.

Bon alors mettons que j'ouvre avec un sirop de huit.

Si c'est vous qu'avez siroté au tour d'avant, ça tourne dans votre sens. Alors soit vous laissez filer, vous dites "file-sirop", soit vous vous sentez de relancer et vous annoncez un sirop de quatorze. Comme on a commencé les annonces, le second joueur a pas le droit de laisser filer. Vous pouvez soit relancer un sirop de vingt-et-un, soit vous abandonnez le tour et vous dites "couche-sirop" ou "sirop Jeannot", ça dépend des régions. Et après, soit on fait la partie soit je fais un "contre-sirop" ! Et à partir de là, sirop de pomme sur vingt-et-un donc on fait la partie en quatre tours jusqu'à qu'il y en ait un qui sirote.

À la gagne, il n'y a que trois possibilités : soit vous faites votre sirop de huit, vous dites "beau sirop" et on recompte, soit vous faites votre sirop de quatorze, vous dites "beau sirop, sirop gagnant" et on vous rajoute la moitié, soit vous faites votre sirop de vingt-et-un et vous dites "beau sirop, mi-sirop, siroté, gagne-sirop, sirop-grelot, passe-montagne, sirop au bon goût".

Normalement ça se joue avec des cartes mais si vous avez que des dés, vous pouvez aussi jouer avec des dés puisque ce qui compte c'est les valeurs."

Au moins un des exercices suivants n'a aucun rapport avec le jeu du Sirop.

Chapitre III

Exercice 00 : ft_ft

	Exercice : 00
	ft_ft
Dossier de rendu :	ex00/
Fichiers à rendre :	ft_ft.c
Fonctions Autorisées :	Aucune

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

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

Chapitre IV

Exercice 01 : ft_ultimate_ft

	Exercice : 01
	ft_ultimate_ft
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	ft_ultimate_ft.c
Fonctions Autorisées :	Aucune

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

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

Chapitre V

Exercice 02 : ft_swap

	Exercice : 02
	ft_swap
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_swap.c
Fonctions Autorisées :	Aucune

- 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 VI

Exercice 03 : ft_div_mod

	Exercice : 03
	ft_div_mod
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : ft_div_mod.c	
Fonctions Autorisées : Aucune	

- 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` et `b` et stocke le résultat 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 VII

Exercice 04 : ft_ultimate_div_mod

	Exercice : 04
	ft_ultimate_div_mod
Dossier de rendu :	<i>ex04/</i>
Fichiers à rendre :	ft_ultimate_div_mod.c
Fonctions Autorisées :	Aucune

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

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

- Cette fonction divise les int pointés par `a` et `b`.
Le résultat de la division est stocké dans l'int pointé par `a`.
Le résultat du reste de la division est stocké dans l'int pointé par `b`.

Chapitre VIII

Exercice 05 : ft_putstr

	Exercice : 05
	ft_putstr
Dossier de rendu :	<i>ex05/</i>
Fichiers à rendre :	ft_putstr.c
Fonctions Autorisées :	write

- Écrire une fonction qui affiche une chaîne de caractères à l'écran.
- Elle devra être prototypée de la façon suivante :

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

Chapitre IX

Exercice 06 : ft_strlen

	Exercice : 06
	ft_strlen
Dossier de rendu :	<i>ex06/</i>
Fichiers à rendre :	ft_strlen.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui compte le nombre de caractères dans une chaîne de caractères et qui retourne le nombre trouvé.
- Elle devra être prototypée de la façon suivante :

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

Chapitre X

Exercice 07 : ft_rev_int_tab

	Exercice : 07
	ft_rev_int_tab
Dossier de rendu : <i>ex07/</i>	
Fichiers à rendre : ft_rev_int_tab.c	
Fonctions Autorisées : Aucune	

- Écrire une fonction qui inverse l'ordre des éléments d'un tableau d'entiers.
- Les paramètres sont un pointeur sur entier et le nombre d'entiers dans le tableau.
- La fonction devra être prototypée de la façon suivante :

```
void    ft_rev_int_tab(int *tab, int size);
```

Chapitre XI

Exercice 08 : ft_sort_int_tab

	Exercice : 08
	ft_sort_int_tab
Dossier de rendu :	ex08/
Fichiers à rendre :	ft_sort_int_tab.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui trie un tableau d'entiers par ordre croissant.
- Les paramètres sont un pointeur sur entier et le nombre d'entiers dans le tableau.
- La fonction devra être prototypée de la façon suivante :

```
void    ft_sort_int_tab(int *tab, int size);
```

Chapitre XII

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 02

Résumé: Ce document est le sujet du module C 02 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_strcpy	5
IV	Exercice 01 : ft_strncpy	6
V	Exercice 02 : ft_str_is_alpha	7
VI	Exercice 03 : ft_str_is_numeric	8
VII	Exercice 04 : ft_str_is_lowercase	9
VIII	Exercice 05 : ft_str_is_uppercase	10
IX	Exercice 06 : ft_str_is_printable	11
X	Exercice 07 : ftstrupcase	12
XI	Exercice 08 : ft_strlowcase	13
XII	Exercice 09 : ft_strcapitalize	14
XIII	Exercice 10 : ft_strlcpy	15
XIV	Exercice 11 : ft_putstr_non_printable	16
XV	Exercice 12 : ft_print_memory	17
XVI	Rendu et peer-evaluation	19

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour cette journée, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

Voici une discussion extraite de la série Silicon Valley :

- I mean, why not just use Vim over Emacs? (CHUCKLES)
- I do use Vim over Emac.
- Oh, God, help us! Okay, uh you know what? I just don't think this is going to work. I'm so sorry. Uh, I mean like, what, we're going to bring kids into this world with that over their heads? That's not really fair to them, don't you think?
- Kids? We haven't even slept together.
- And guess what, it's never going to happen now, because there is no way I'm going to be with someone who uses spaces over tabs.
- Richard! (PRESS SPACE BAR MANY TIMES)
- Wow. Okay. Goodbye.
- One tab saves you eight spaces! - (DOOR SLAMS) - (BANGING)

. . .

(RICHARD MOANS)

- Oh, my God! Richard, what happened?
- I just tried to go down the stairs eight steps at a time. I'm okay, though.
- See you around, Richard.
- Just making a point.

Heureusement, vous n'êtes pas obligé d'utiliser emacs et votre barre espace pour compléter les exercices suivants.

Chapitre III

Exercice 00 : ft_strdup

	Exercice : 00
	ft_strdup
Dossier de rendu :	<i>ex00/</i>
Fichiers à rendre :	ft_strdup.c
Fonctions Autorisées :	Aucune

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

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

Chapitre IV

Exercice 01 : ft_strncpy

	Exercice : 01
	ft_strncpy
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	ft_strncpy.c
Fonctions Autorisées :	Aucune

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

```
char *ft_strncpy(char *dest, char *src, unsigned int n);
```

Chapitre V

Exercice 02 : ft_str_is_alpha

	Exercice : 02
	ft_str_is_alpha
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : ft_str_is_alpha.c	
Fonctions Autorisées : Aucune	

- Écrire une fonction qui renvoie 1 si la chaîne passée en paramètre ne contient que des caractères alphabétiques et renvoie 0 si la chaîne passée en paramètre contient d'autres types de caractères.
- Elle devra être prototypée de la façon suivante :

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

- Elle devra renvoyer 1 si **str** est une chaîne vide.

Chapitre VI

Exercice 03 : ft_str_is_numeric

	Exercice : 03
	ft_str_is_numeric
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : ft_str_is_numeric.c	
Fonctions Autorisées : Aucune	

- Écrire une fonction qui renvoie 1 si la chaîne passée en paramètre ne contient que des chiffres et renvoie 0 si la chaîne passée en paramètre contient d'autres types de caractères.
- Elle devra être prototypée de la façon suivante :

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

- Elle devra renvoyer 1 si **str** est une chaîne vide.

Chapitre VII

Exercice 04 : ft_str_is_lowercase

	Exercice : 04
	ft_str_is_lowercase
Dossier de rendu :	<i>ex04/</i>
Fichiers à rendre :	ft_str_is_lowercase.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui renvoie 1 si la chaîne passée en paramètre ne contient que des caractères alphabétiques minuscules et renvoie 0 si la chaîne passée en paramètre contient d'autres types de caractères.
- Elle devra être prototypée de la façon suivante :

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

- Elle devra renvoyer 1 si **str** est une chaîne vide.

Chapitre VIII

Exercice 05 : ft_str_is_uppercase

	Exercice : 05
	ft_str_is_uppercase
Dossier de rendu :	<i>ex05/</i>
Fichiers à rendre :	ft_str_is_uppercase.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui renvoie 1 si la chaîne passée en paramètre ne contient que des caractères alphabétiques majuscules et renvoie 0 si la chaîne passée en paramètre contient d'autres types de caractères.
- Elle devra être prototypée de la façon suivante :

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

- Elle devra renvoyer 1 si **str** est une chaîne vide.

Chapitre IX

Exercice 06 : ft_str_is_printable

	Exercice : 06
	ft_str_is_printable
Dossier de rendu :	<i>ex06/</i>
Fichiers à rendre :	ft_str_is_printable.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui renvoie 1 si la chaîne passée en paramètre ne contient que des caractères affichables et renvoie 0 si la chaîne passée en paramètre contient d'autres types de caractères.
- Elle devra être prototypée de la façon suivante :

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

- Elle devra renvoyer 1 si **str** est une chaîne vide.

Chapitre X

Exercice 07 : ft_strdupcase

	Exercice : 07
	ft_strdupcase
Dossier de rendu :	<i>ex07/</i>
Fichiers à rendre :	ft_strdupcase.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui met en majuscule chaque lettre.
- Elle devra être prototypée de la façon suivante :

```
char *ft_strdupcase(char *str);
```

- Elle devra renvoyer **str**.

Chapitre XI

Exercice 08 : ft_strlowlcase

	Exercice : 08
	ft_strlowlcase
Dossier de rendu :	<i>ex08/</i>
Fichiers à rendre :	ft_strlowlcase.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui met en minuscule chaque lettre.
- Elle devra être prototypée de la façon suivante :

```
char *ft_strlowlcase(char *str);
```

- Elle devra renvoyer **str**.

Chapitre XII

Exercice 09 : ft_strcapitalize

	Exercice : 09
	ft_strcapitalize
Dossier de rendu :	<i>ex09/</i>
Fichiers à rendre :	ft_strcapitalize.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui met en majuscule la première lettre de chaque mot et le reste du mot en minuscule.
- Un mot est une suite de caractères alphanumériques.
- Elle devra être prototypée de la façon suivante :

```
char *ft_strcapitalize(char *str);
```

- Elle devra renvoyer **str**.
- Par exemple :

```
salut, comment tu vas ? 42mots quarante-deux; cinquante+et+un
```

- Doit donner :

```
Salut, Comment Tu Vas ? 42mots Quarante-Deux; Cinquante+Et+Un
```

Chapitre XIII

Exercice 10 : ft_strlcpy

	Exercice : 10
	ft_strlcpy
Dossier de rendu :	<i>ex10/</i>
Fichiers à rendre :	ft_strlcpy.c
Fonctions Autorisées :	Aucune

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

```
unsigned int ft_strlcpy(char *dest, char *src, unsigned int size);
```

Chapitre XIV

Exercice 11 : ft_putstr_non_printable

	Exercice : 11
	ft_putstr_with_non_printable
Dossier de rendu : <i>ex11/</i>	
Fichiers à rendre : ft_putstr_non_printable.c	
Fonctions Autorisées : write	

- Écrire une fonction qui affiche une chaîne de caractères à l'écran. Si cette chaîne contient des caractères non-imprimables, ils devront être affichés sous forme hexadécimale (en minuscules) en les précédant d'un "backslash".
- Par exemple, avec ce paramètre :

```
Coucou\ntu vas bien ?
```

- La fonction devra afficher :

```
Coucou\0atu vas bien ?
```

- Elle devra être prototypée de la façon suivante :

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

Chapitre XV

Exercice 12 : ft_print_memory

	Exercice : 12
	ft_print_memory
Dossier de rendu :	<i>ex12/</i>
Fichiers à rendre :	ft_print_memory.c
Fonctions Autorisées :	write

- Écrire une fonction qui affiche une zone mémoire à l'écran.
- L'affichage de la zone mémoire est séparée en trois "colonnes" séparées par un espace :
 - L'adresse en hexadécimal du premier caractère de la ligne suivi d'un ':'.
 - Le contenu en hexadécimal avec un espace tous les deux caractères et doit être complété avec des espaces si nécessaire (voir l'exemple en dessous).
 - Le contenu en caractères imprimables.
- Si un caractère est non-imprimable il sera remplacé par un point.
- Chaque ligne doit gérer seize caractères.
- Si **size** est égale à 0, rien ne sera affiché.

- Exemple :

```
$> ./ft_print_memory
000000010a161f40: 426f 6e6a 6f75 7220 6c65 7320 616d 696e Bonjour les amin
000000010a161f50: 6368 6573 090a 0963 0720 6573 7420 666f ches...c. est fo
000000010a161f60: 7509 746f 7574 0963 6520 7175 206f 6e20 u.tout.ce qu on
000000010a161f70: 7065 7574 2066 6169 7265 2061 7665 6309 peut faire avec.
000000010a161f80: 0a09 7072 696e 745f 6d65 6d6f 7279 0a0a ..print_memory..
000000010a161f90: 0a09 6c6f 6c2e 6c6f 6c0a 2000 ..lol.lol. .
$> ./ft_print_memory | cat -te
0000000107ff9f40: 426f 6e6a 6f75 7220 6c65 7320 616d 696e Bonjour les amin$
0000000107ff9f50: 6368 6573 090a 0963 0720 6573 7420 666f ches...c. est fo$#
0000000107ff9f60: 7509 746f 7574 0963 6520 7175 206f 6e20 u.tout.ce qu on $#
0000000107ff9f70: 7065 7574 2066 6169 7265 2061 7665 6309 peut faire avec.$#
0000000107ff9f80: 0a09 7072 696e 745f 6d65 6d6f 7279 0a0a ..print_memory..$#
0000000107ff9f90: 0a09 6c6f 6c2e 6c6f 6c0a 2000 ..lol.lol. .$#
$>
```

- Elle devra être prototypée de la façon suivante :

```
void *ft_print_memory(void *addr, unsigned int size);
```

- Elle devra renvoyer addr.

Chapitre XVI

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 03

Résumé: Ce document est le sujet du module C 03 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_strcmp	5
IV	Exercice 01 : ft_strncmp	6
V	Exercice 02 : ft_strcat	7
VI	Exercice 03 : ft_strncat	8
VII	Exercice 04 : ft_strstr	9
VIII	Exercice 05 : ft_strlcat	10
IX	Rendu et peer-evaluation	11

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour cette journée, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

Le livre Wuzazu contient la première mention connue du jeu "Papier-Caillou-Ciseau". Il a été écrit par l'écrivain de la dynastie Ming Xie Zhaozhi, qui indique alors que ce jeu date de la dynastie Han (206 BC - 220 AD). Dans le livre, le jeu s'appelait shoushiling. Le livre Note of Liuyanzhai mentionne également le jeu, l'appelant shoushiling, huozhitou, or huoquan.

Au travers de l'histoire japonaise, nous retrouverons des références fréquentes au jeux "sansukumi-ken", "ken" signifiant jeu de poing, avec une impasse "sukumi" à trois voies "san". C'est à lire dans le sens A bat B, B bat C et C bat A. Ce jeu est originaire de Chine avant d'être importé au Japon et de devenir populaire.

Au début du 20ème siècle, papier caillou ciseaux s'est répandu au delà de l'Asie, notamment grâce au contact augmenté entre le Japon et l'Ouest. Son nom anglais est alors pris par traduction du nom japonais des gestes utilisés. Dans le reste de l'Asie, le papier est remplacé par tissu. La forme des ciseaux est également adoptée du style japonais.

En 1927, "La vie au patronage", un magazine pour enfants en France, le décrivait en détail, et le considérant comme un "jeu japonais". Son nom français alternatif ("chifou-mi"), est basé sur les anciens mots japonais pour "un, deux, trois" ("hi, fu, mi").

Un article du New York Times de 1932 explique les règles pour les lecteurs américains, indiquant que le jeu n'était alors pas très répandu. L'édition de 1933 du magazine "Compton's Pictured Encyclopedia" le décrivait comme un moyen commun de résolution des conflits entre enfants lors de son article sur le Japon : "This is such a good way of deciding an argument that American boys and girls might like to practice it too."

Chapitre III

Exercice 00 : ft_strcmp

	Exercice : 00
	ft_strcmp
Dossier de rendu :	<i>ex00/</i>
Fichiers à rendre :	ft_strcmp.c
Fonctions Autorisées :	Aucune

- 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 IV

Exercice 01 : ft_strcmp

	Exercice : 01
	ft_strcmp
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	ft_strcmp.c
Fonctions Autorisées :	Aucune

- 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, unsigned int n);
```

Chapitre V

Exercice 02 : ft_strcat

	Exercice : 02
	ft_strcat
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_strcat.c
Fonctions Autorisées :	Aucune

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

```
char *ft_strcat(char *dest, char *src);
```

Chapitre VI

Exercice 03 : ft_strncat

	Exercice : 03
	ft_strncat
Dossier de rendu :	<i>ex03/</i>
Fichiers à rendre :	ft_strncat.c
Fonctions Autorisées :	Aucune

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

```
char *ft_strncat(char *dest, char *src, unsigned int nb);
```

Chapitre VII

Exercice 04 : ft_strcmp

	Exercice : 04
	ft_strcmp
Dossier de rendu :	<i>ex04/</i>
Fichiers à rendre :	ft_strcmp.c
Fonctions Autorisées :	Aucune

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

```
char *ft_strcmp(char *str, char *to_find);
```

Chapitre VIII

Exercice 05 : ft_strlcat

	Exercice : 05
	ft_strlcat
Dossier de rendu :	<i>ex05/</i>
Fichiers à rendre :	ft_strlcat.c
Fonctions Autorisées :	Aucune

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

```
unsigned int ft_strlcat(char *dest, char *src, unsigned int size);
```

Chapitre IX

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 04

Résumé: ce document est le sujet du module C 04 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_strlen	5
IV	Exercice 01 : ft_putstr	6
V	Exercice 02 : ft_putnbr	7
VI	Exercice 03 : ft_atoi	8
VII	Exercice 04 : ft_putnbr_base	9
VIII	Exercice 05 : ft_atoi_base	10
IX	Rendu et peer-evaluation	11

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour cette journée, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

Voici les paroles du générique de Nicky Larson :

Une ombre file dans la nuit
C'est un assassin qui s'enfuit
Et comme un démon il sourit
Son crime restera impuni
Une voiture qui surgit
Un coup de frein, des pneus qui crient
Un coup de feu qui retentit
La justice s'appelle Nicky

[Refrain]

Dans la chaleur
De la nuit
Le mal est toujours puni
Aucun danger ne l'impressionne
Les coups durs il les affectionne
Et la justice le passionne
Nicky Larson ne craint personne
Lorsque les coups de feu résonnent
Comme un éclair il tourbillonne
Surtout si la fille est mignonne
Nicky Larson ne craint personne

Comme un chasseur il suit sa proie
Pour que la justice et le droit
Triomphent, il est prêt à donner
Toute sa vie sans hésiter
Quand sa silhouette apparaît
Les méchants se mettent à trembler
Ils savent qu'ils ne pourront jamais
Échapper à ce justicier

[Refrain]

Ce sujet n'a, malheureusement, rien à voir avec Nicky Larson.

Chapitre III

Exercice 00 : ft_strlen

	Exercice : 00
	ft_strlen
Dossier de rendu :	<i>ex00/</i>
Fichiers à rendre :	ft_strlen.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui compte le nombre de caractères dans une chaîne de caractères et qui retourne le nombre trouvé.
- Elle devra être prototypée de la façon suivante :

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

Chapitre IV

Exercice 01 : ft_putstr

	Exercice : 01
	ft_putstr
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	ft_putstr.c
Fonctions Autorisées :	write

- 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 V

Exercice 02 : ft_putnbr

	Exercice : 02
	ft_putnbr
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_putnbr.c
Fonctions Autorisées :	write

- Écrire une fonction qui affiche un nombre passé en paramètre. La fonction devra être capable d'afficher la totalité des valeurs possibles dans une variable de type **int**.
- Elle devra être prototypée de la façon suivante :

```
void ft_putnbr(int nb);
```

- Par exemple :
 - `ft_putnbr(42)` affiche "42".

Chapitre VI

Exercice 03 : ft_atoi

	Exercice : 03
	ft_atoi
Dossier de rendu :	<i>ex03/</i>
Fichiers à rendre :	ft_atoi.c
Fonctions Autorisées :	Aucune

- Ecrire une fonction qui convertit le début de la chaîne pointée par str en entier de type int
- str peut commencer par un nombre arbitraire de ‘white space’ (comme défini par `isspace(3)`)
- str peut ensuite être suivi par un nombre arbitraire de signe + et de signe -. Le signe - fera changer le signe de l’entier retourné en fonction du nombre de signe - et si celui ci est pair ou impair.
- Pour finir str devra être composée de chiffre de la base 10
- Votre fonction devra lire str tant que celle-ci suit les règles au dessus et elle doit retourner le nombre trouvé jusque là.
- Vous ne devriez pas prendre en compte les overflows et les underflows, le résultat est considérer comme indéfini dans ces cas.
- Vous pouvez comparer votre fonction avec la vrai fonction atoi à part la partie sur les signes ainsi que l’overflow.
- Voici l’exemple d’un programme qui affiche la valeur de retour de atoi :

```
$> ./a.out " ---+---+1234ab567"  
-1234
```

- Elle devra être prototypée de la façon suivante :

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

Chapitre VII

Exercice 04 : ft_putnbr_base

	Exercice : 04
	ft_putnbr_base
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : ft_putnbr_base.c	
Fonctions Autorisées : write	

- Écrire une fonction qui affiche un nombre dans le terminal dans une base donnée.
- Ce nombre est fourni sous la forme d'un **int** et la base sous la forme d'une **chaîne de caractères**.
- La base contient tous les symboles utilisables pour afficher le nombre :
 - 0123456789 est la base couramment utilisée pour représenter nos nombres décimaux ;
 - 01 est une base binaire ;
 - 0123456789ABCDEF est une base hexadécimale ;
 - poneyvif est une base octale.
- La fonction doit gérer les nombres négatifs.
- Si un paramètre contient une erreur la fonction n'affiche rien. Une erreur peut être :
 - base est vide ou est de taille 1 ;
 - base contient deux fois le même caractère ;
 - base contient les caractères + ou -.
- Elle devra être prototypée de la façon suivante :

```
void        ft_putnbr_base(int nbr, char *base);
```

Chapitre VIII

Exercice 05 : ft_atoi_base

	Exercice : 05
	ft_atoi_base
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : ft_atoi_base.c	
Fonctions Autorisées : Aucune	

- Ecrire une fonction qui convertit le début de la chaîne pointée par str en entier de type int.
- str est dans une base spécifique passée en second argument de la fonction.
- À part le système de base, cette fonction doit reproduire le comportement de ft_atoi.
- Si un paramètre contient une erreur la fonction renvoie 0. Une erreur peut être :
 - la base est vide ou est de taille 1 ;
 - la base contient deux fois le même caractère ;
 - la base contient les caractères + ou - ou des whitespaces ;
- Elle devra être prototypée de la façon suivante :

```
int      ft_atoi_base(char *str, char *base);
```

Chapitre IX

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 05

Résumé: ce document est le sujet du module C 05 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_iterative_factorial	6
IV	Exercice 01 : ft_recursive_factorial	7
V	Exercice 02 : ft_iterative_power	8
VI	Exercice 03 : ft_recursive_power	9
VII	Exercice 04 : ft_fibonacci	10
VIII	Exercice 05 : ft_sqrt	11
IX	Exercice 06 : ft_is_prime	12
X	Exercice 07 : ft_find_next_prime	13
XI	Exercice 08 : Les dix dames	14
XII	Rendu et peer-evaluation	15

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour cette journée, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

Voici des paroles extraite du premier livre de la saga Harry Potter :

Je n'suis pas d'une beauté suprême
Mais faut pas s'fier à ce qu'on voit
Je veux bien me manger moi-même
Si vous trouvez plus malin qu'moi.

Les hauts-d'forme, les chapeaux splendides,
Font pâl'figure auprès de moi
Car à Poudlard, quand je décide,
Chacun se soumet à mon choix.

Rien ne m'échapp' rien ne m'arrête
Le Choixpeau a toujours raison
Mettez-moi donc sur votre tête
Pour connaitre votre maison.

Si vous allez à Gryffondor
Vous rejoindrez les courageux,
Les plus hardis et les plus forts
Sont rassemblés en ce haut lieu.

Si à Poufsouffle vous allez,
Comme eux vous s'rez juste et loyal
Ceux de Poufsouffle aiment travailler
Et leur patience est proverbiale.

Si vous êtes sage et réfléchi
Serdaigle vous accueillera peut-être
Là-bas, ce sont des érudits
Qui ont envie de tout connaître.

Vous finirez à Serpentard
Si vous êtes plutôt malin,
Car ceux-là sont de vrais roublards
Qui parviennent toujours à leurs fins.

Sur ta tête pose-moi un instant
Et n'aie pas peur, reste serein
Tu seras en de bonnes mains
Car je suis un chapeau pensant !

Ce sujet n'a, malheureusement, rien à voir avec la série Harry Potter, et c'est dommage, parce que votre rendu ne sera pas fait par magie.

Chapitre III

Exercice 00 : ft_iterative_factorial

	Exercice : 00
	ft_iterative_factorial
Dossier de rendu :	ex00/
Fichiers à rendre :	ft_iterative_factorial.c
Fonctions Autorisées :	Aucune

- Écrire une fonction itérative qui renvoie un nombre. Ce nombre est le résultat de l'opération factorielle à partir du nombre passé en paramètre.
- Si l'argument n'est pas valide, la fonction doit renvoyer 0.
- Il ne faut pas gerer les "int overflow", le retour de la fonction sera indefini.
- Elle devra être prototypée de la façon suivante :

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

Chapitre IV

Exercice 01 : ft_recursive_factorial

	Exercice : 01
	ft_recursive_factorial
Dossier de rendu :	ex01/
Fichiers à rendre :	ft_recursive_factorial.c
Fonctions Autorisées :	Aucune

- Écrire une fonction récursive qui renvoie la factorielle du nombre passé en paramètre.
- Si l'argument n'est pas valide, la fonction doit renvoyer 0.
- Il ne faut pas gerer les "int overflow", le retour de la fonction sera indefini.
- Elle devra être prototypée de la façon suivante :

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

Chapitre V

Exercice 02 : ft_iterative_power

	Exercice : 02
	ft_iterative_power
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_iterative_power.c
Fonctions Autorisées :	Aucune

- Écrire une fonction itérative qui renvoie une puissance d'un nombre. Une puissance inférieure à 0 renverra 0.
- Comme il n'y a pas de consensus sur 0 puissance 0, nous considererons que le résultat sera 1.
- Il ne faut pas gérer les "int overflow", le retour de la fonction sera indéfini.
- Elle devra être prototypée de la façon suivante :

```
int ft_iterative_power(int nb, int power);
```

Chapitre VI

Exercice 03 : ft_recursive_power

	Exercice : 03
	ft_recursive_power
Dossier de rendu :	<i>ex03/</i>
Fichiers à rendre :	ft_recursive_power.c
Fonctions Autorisées :	Aucune

- Écrire une fonction récursive qui renvoie une puissance d'un nombre.
- Comme il n'y a pas de consensus sur 0 puissance 0, nous considererons que le resultat sera 1.
- Il ne faut pas gerer les "int overflow", le retour de la fonction sera indefini.
- Elle devra être prototypée de la façon suivante :

```
int ft_recursive_power(int nb, int power);
```

Chapitre VII

Exercice 04 : ft_fibonacci

	Exercice : 04
	ft_fibonacci
Dossier de rendu :	<i>ex04/</i>
Fichiers à rendre :	ft_fibonacci.c
Fonctions Autorisées :	Aucune

- Écrire une fonction **ft_fibonacci** qui renvoie le **n**-ième élément de la suite de Fibonacci, le premier élément étant à l'index 0. Nous considererons que la suite de Fibonacci commence par 0, 1, 1, 2.
- Les overflows ne devront pas être gérés.
- Elle devra être prototypée de la façon suivante :

```
int ft_fibonacci(int index);
```

- Évidemment, **ft_fibonacci** devra être récursive.
- Si **index** est inférieur à 0, la fonction renverra -1.

Chapitre VIII

Exercice 05 : ft_sqrt

	Exercice : 05
	ft_sqrt
Dossier de rendu :	<i>ex05/</i>
Fichiers à rendre :	ft_sqrt.c
Fonctions Autorisées :	Aucune

- 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);
```

Chapitre IX

Exercice 06 : ft_is_prime

	Exercice : 06
	ft_is_prime
Dossier de rendu :	<i>ex06/</i>
Fichiers à rendre :	ft_is_prime.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui renvoie 1 si le nombre est premier et 0 si le nombre ne l'est pas.
- Elle devra être prototypée de la façon suivante :

```
int ft_is_prime(int nb);
```



0 et 1 ne sont pas des nombres premiers.

Chapitre X

Exercice 07 : ft_find_next_prime

	Exercice : 07
	ft_find_next_prime
Dossier de rendu :	<i>ex07/</i>
Fichiers à rendre :	ft_find_next_prime.c
Fonctions Autorisées :	Aucune

- Écrire une fonction qui renvoie le nombre premier immédiatement supérieur ou égal au nombre passé en paramètre.
- Elle devra être prototypée de la façon suivante :

```
int ft_find_next_prime(int nb);
```

Chapitre XI

Exercice 08 : Les dix dames

	Exercice : 08
	Les dix dames
Dossier de rendu : <i>ex08/</i>	
Fichiers à rendre : <i>ft_ten_queens_puzzle.c</i>	
Fonctions Autorisées : <code>write</code>	

- Écrire une fonction qui affiche toutes les possibilités de placer dix dames sur un échiquier de 10x10 sans qu'elles ne puissent s'atteindre en un seul coup.
- La recursivité devra être utilisée.
- La valeur de retour de votre fonction devra être le nombre de solutions affichées
- Elle devra être prototypée de la façon suivante :

```
int ft_ten_queens_puzzle(void);
```

- L'affichage se fera de la façon suivante :

```
$>./a.out | cat -e
0257948136$ 
0258693147$ 
...
4605713829$ 
4609582731$ 
...
9742051863$ 
$>
```

- La suite se lit de gauche à droite. Le premier chiffre correspond à la position de la première dame dans la première colonne (l'index commençant à 0). Le énième chiffre correspond à la position de la énième dame dans la énième colonne.

Chapitre XII

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 06

Staff 42 pedago@42.fr

Résumé: Ce document est le sujet du module C 06 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_print_program_name	5
IV	Exercice 01 : ft_print_params	6
V	Exercice 02 : ft_rev_params	7
VI	Exercice 03 : ft_sort_params	8
VII	Rendu et peer-evaluation	9

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour cette journée, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

Dialogue issue du film The Big Lebowski :

Walter : Excuse-moi Smokey, t'as mordu la ligne : y'a faute.

Smokey : Mon cul ! Tu me mets huit, Duc...

Walter : Je te demande pardon, mets lui zéro, jeu suivant.

Smokey : Tu fais chier Walter, tu me mets huit Duc...

Walter : Smokey on est pas au Viet-nâm, on est au bowling, on joue selon les règles...

Le Duc : Allez, déconne pas Walter, on est là merde : son pied a légèrement mordu, il a un peu glissé... C'est qu'un sport, là !

Walter : Oui. Et il est homologué, c'est une partie qui compte pour le tournoi, j'ai pas raison ?

Smokey : Ouais, mais j'ai pas...

Walter : J'ai pas raison ?

Smokey : Ouais, mais j'ai pas mordu... Allez ! Vas-y Duc, mets moi un huit.

Walter : [Sort son flingue] Smokey mon ami, si t'as jamais souffert, tu vas comprendre...

Le Duc : Walter, fais pas le con !

Walter : [Menaçant] Vas-y, mets-toi un huit et tu vas comprendre !

Smokey : J'ai pas m...

Walter : Tu vas comprendre ta douleur, Smokey !

Smokey : Duc ? C'est ton partenaire...

Walter : Le monde est en train de DEVENIR CINGLÉ ! [Se lève] Y'A PERSONNE ICI À PART MOI QUI SE SOUCIE ENCORE DE RESPECTER LES RÈGLES ? METS-TOI ZÉRO !

Le Duc : Ils sont en train d'appeler les flics Walter remets ça dans...

Walter : [Braque son flingue sur Smokey] METS-TOI ZÉRO !

Le Duc : Range ca Walter...

Smokey : ...

Le Duc : Walter...

Walter : [Arme son flingue] TU CROIS PEUT-ÊTRE QUE JE PLAISANTE ?
METTS-TOI ZÉRO !

Smokey : Voila, je me suis mis zéro... T'es content ? Espèce de malade !

Walter : [Se calme] ... C'est un sport homologué.

Chapitre III

Exercice 00 : ft_print_program_name

	Exercice : 00
	ft_print_program_name
Dossier de rendu :	<i>ex00/</i>
Fichiers à rendre :	<code>ft_print_program_name.c</code>
Fonctions Autorisées :	<code>write</code>

- Il s'agit ici d'un programme, vous devrez donc avoir une fonction `main` dans votre `fichier.c`.
- Écrire un programme qui affiche le nom du programme.
- Exemple :

```
$>./a.out  
./a.out  
$>
```

Chapitre IV

Exercice 01 : ft_print_params

	Exercice : 01
	ft_print_params
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	ft_print_params.c
Fonctions Autorisées :	write

- 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.
- Un par ligne dans le même ordre que la ligne de commande
- Vous devez afficher tous les arguments, sauf **argv[0]**.
- Exemple :

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

Chapitre V

Exercice 02 : ft_rev_params

	Exercice : 02
	ft_rev_params
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_rev_params.c
Fonctions Autorisées :	write

- 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.
- Un par ligne dans l'ordre inverse de la ligne de commande.
- Vous devez afficher tous les arguments, sauf **argv[0]**.

Chapitre VI

Exercice 03 : ft_sort_params

	Exercice : 03
	ft_sort_params
Dossier de rendu :	<i>ex03/</i>
Fichiers à rendre :	<code>ft_sort_params.c</code>
Fonctions Autorisées :	<code>write</code>

- 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]`.
- Un argument par ligne.

Chapitre VII

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 07

Staff 42 piscine@42.fr

Résumé: Ce document est le sujet du module C 07 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_strdup	5
IV	Exercice 01 : ft_range	6
V	Exercice 02 : ft_ultimate_range	7
VI	Exercice 03 : ft_strjoin	8
VII	Exercice 04 : ft_convert_base	9
VIII	Exercice 05 : ft_split	10
IX	Rendu et peer-evaluation	11

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour cette journée, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

Voici une liste des monstres que l'on peut trouver dans le célèbre Donjon de Naheul-beuk :

- Toutes sortes de morts-vivants ;
- Des araignées géantes ;
- Des orques ;
- Des gobelins ;
- Des trolls dans les souterrains ;
- Des sorciers ;
- Des guerriers maudits ;
- Des rats mutants ;
- Une bouteille d'huile ;
- Du papier toilette ;
- Deux éponges ;
- Des raviolis.

Chapitre III

Exercice 00 : ft_strdup

	Exercice : 00
	ft_strdup
Dossier de rendu :	<i>ex00/</i>
Fichiers à rendre :	ft_strdup.c
Fonctions Autorisées :	malloc

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

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

Chapitre IV

Exercice 01 : ft_range

	Exercice : 01
	ft_range
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	ft_range.c
Fonctions Autorisées :	malloc

- 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 - 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 V

Exercice 02 : ft_ultimate_range

	Exercice : 02
	ft_ultimate_range
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_ultimate_range.c
Fonctions Autorisées :	malloc

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

```
int     ft_ultimate_range(int **range, int min, int max);
```

- La taille de **range** sera retornnée (ou -1 en cas de problème).
- Si la valeur **min** est supérieure ou égale à la valeur **max**, **range** pointera sur NULL et on renverra 0.

Chapitre VI

Exercice 03 : ft_strjoin

	Exercice : 03
	ft_strjoin
Dossier de rendu :	<i>ex03/</i>
Fichiers à rendre :	ft_strjoin.c
Fonctions Autorisées :	malloc

- Écrire une fonction qui va concatener l'ensemble des chaîne de caractères pointées par **strs** en les séparants à l'aide de **sep**.
- **size** représente la taille de **strs**.
- Si **size** vaut 0, il faut retourner une chaîne de caractères vide que l'on peut **free()**.
- Elle devra être prototypée de la façon suivante :

```
char *ft_strjoin(int size, char **strs, char *sep);
```

Chapitre VII

Exercice 04 : ft_convert_base

	Exercice : 04
	ft_convert_base
Dossier de rendu :	<i>ex04/</i>
Fichiers à rendre :	ft_convert_base.c, ft_convert_base2.c
Fonctions Autorisées :	malloc, free

- Écrire une fonction qui renvoie le résultat de la conversion de la chaîne **nbr** exprimée en une base **base_from** dans une base **base_to**.
- **nbr, base_from, base_to** ne seront pas forcement modifiable.
- **nbr** suivra les même règles que **ft_atoi_base**. Attention donc au '+', '-' et aux whitespaces.
- Le nombre représenté par **nbr** tient dans un **int**.
- Si une base est incorrecte, la fonction renverra **NULL**.
- Le nombre retourné doit être préfixé seulement par un seul et unique '-' si c'est nécessaire, pas de whitespaces ou de '+'.
- Elle devra être prototypée de la façon suivante :

```
char *ft_convert_base(char *nbr, char *base_from, char *base_to);
```

Chapitre VIII

Exercice 05 : ft_split

	Exercice : 05
	ft_split
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : ft_split.c	
Fonctions Autorisées : malloc	

- Écrire une fonction qui découpe une chaîne de caractères en fonction d'une autre chaîne de caractères.
- Il faudra utiliser chaque caractère de la chaîne **charset** comme séparateur.
- La fonction renvoie un tableau où chaque élément de celui-ci contient l'adresse d'une chaîne de caractères comprise entre deux séparateur. Le dernier élément du tableau devra être égal à 0 pour marquer la fin du tableau.
- Il ne doit pas y avoir de chaîne vide dans votre tableau. Tirez-en les conclusions qui s'imposent.
- La chaîne qui sera transmise ne sera pas modifiable.
- Elle devra être prototypée de la façon suivante :

```
char **ft_split(char *str, char *charset);
```

Chapitre IX

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 08

Résumé: Ce document est le sujet du module C 08 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft.h	5
IV	Exercice 01 : ft_boolean.h	6
V	Exercice 02 : ft_abs.h	8
VI	Exercice 03 : ft_point.h	9
VII	Exercice 04 : ft_strs_to_tab	10
VIII	Exercice 05 : ft_show_tab	12
IX	Rendu et peer-evaluation	13

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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

L'encyclopédie collaborative *Wikipédia* a ceci à dire sur l'ornithorynque :

L'ornithorynque (*Ornithorhynchus anatinus*) est une espèce de petits mammifères semi-aquatiques endémique de l'est de l'Australie, y compris la Tasmanie. C'est l'une des cinq espèces de l'ordre des monotrèmes, seul ordre de mammifères qui pond des œufs au lieu de donner naissance à des petits complètement formés (les quatre autres espèces sont des échidnés). C'est la seule espèce survivante de la famille des Ornithorhynchidae et du genre *Ornithorhynchus* bien qu'un grand nombre de fragments d'espèces fossiles de cette famille et de ce genre aient été découverts.

L'apparence bizarre de ce mammifère pondant des œufs, muni d'aiguillons venimeux, à la mâchoire cornée ressemblant au bec d'un canard, à queue évoquant un castor, qui lui sert à la fois de gouvernail dans l'eau et de réserve de graisse, et à pattes de loutre a fortement surpris les premiers explorateurs qui l'ont découvert ; bon nombre de naturalistes européens ont cru à une plaisanterie. C'est l'un des rares mammifères venimeux : le mâle porte sur les pattes postérieures un aiguillon qui peut libérer du venin capable d'infliger de vives douleurs à un être humain. Les traits originaux de l'ornithorynque en font un sujet d'études important pour mieux comprendre l'évolution des espèces animales et en ont fait un des symboles de l'Australie : il a été utilisé comme mascotte pour de nombreux évènements nationaux et il figure au verso de la pièce de 20 cents australiens.

Jusqu'au début du XXe siècle, il a été chassé pour sa fourrure mais il est protégé à l'heure actuelle. Bien que les programmes de reproduction en captivité aient eu un succès très limité et qu'il soit sensible aux effets de la pollution, l'espèce n'est pas encore considérée comme en danger.

Ce sujet ne traite pas de l'ornithorynque.

Chapitre III

Exercice 00 : ft.h

	Exercice : 00
	ft.h
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : ft.h	
Fonctions Autorisées : Aucune	

- Écrire votre fichier **ft.h**
- Il contient tous les prototypages des fonctions :

```
void    ft_putchar(char c);
void    ft_swap(int *a, int *b);
void    ft_putstr(char *str);
int     ft_strlen(char *str);
int     ft_strcmp(char *s1, char *s2);
```

Chapitre IV

Exercice 01 : ft_boolean.h

	Exercice : 01
	ft_boolean.h
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : ft_boolean.h	
Fonctions Autorisées : Aucune	

- Écrire un fichier **ft_boolean.h** qui fera compiler et fonctionner correctement le main suivant :

```
#include "ft_boolean.h"

void        ft_putstr(char *str)
{
    while (*str)
        write(1, str++, 1);
}

t_bool      ft_is_even(int nbr)
{
    return ((EVEN(nbr)) ? TRUE : FALSE);
}

int         main(int argc, char **argv)
{
    (void)argv;
    if (ft_is_even(argc - 1) == TRUE)
        ft_putstr(EVEN_MSG);
    else
        ft_putstr(ODD_MSG);
    return (SUCCESS);
}
```

- Ce programme devra afficher

```
I have an even number of arguments.
```

- ou

```
I have an odd number of arguments.
```

- suivi d'un retour à la ligne, dans le cas adéquat.



Pour cette exercice, la norminette doit être lancée avec le flag `-R CheckDefine`. La moulinette l'utilisera aussi.

Chapitre V

Exercice 02 : ft_abs.h

	Exercice : 02
	ft_abs.h
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_abs.h
Fonctions Autorisées :	Aucune

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

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



Pour cette exercice, la norminette doit être lancée avec le flag *-R CheckDefine*. La moulinette l'utilisera aussi.

Chapitre VI

Exercice 03 : ft_point.h

	Exercice : 03
	ft_point.h
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : ft_point.h	
Fonctions Autorisées : Aucune	

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

```
#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 VII

Exercice 04 : ft_strs_to_tab

	Exercice : 04
	ft_strs_to_tab
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : ft_strs_to_tab.c	
Fonctions Autorisées : malloc, free	

- Ecrire une fonction qui prend en paramètre un tableau de chaîne de caractères ainsi que la taille de ce tableau et renvoie un tableau de structure.
- Elle devra être prototypée de la façon suivante :

```
struct s_stock_str *ft_strs_to_tab(int ac, char **av);
```

- Elle doit transformer chaque élément du tableau de chaîne de caractères en structure.
- La structure sera définie dans le fichier **ft_stock_str.h** comme suit :

```
typedef struct s_stock_str
{
    int size;
    char *str;
    char *copy;
} t_stock_str;
```

- **size** étant la taille de la chaîne de caractères ;
- **str** étant la chaîne de caractères ;
- **copy** étant une copie de la chaîne de caractères ;
- Elle doit garder l'ordre des éléments de **av**.
- Le tableau de structures devra être alloué et le dernier élément aura 0 pour valeur de **str**, ceci afin de signifier la fin du tableau.

- Si une erreur d'allocation arrive elle doit renvoyer un pointeur NULL.
- Nous testons votre fonction avec notre `ft_show_tab` (exercice suivant). Prenez les mesures nécessaires pour que cela fonctionne !

Chapitre VIII

Exercice 05 : ft_show_tab

	Exercice : 05
	ft_show_tab
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : ft_show_tab.c	
Fonctions Autorisées : write	

- Écrire une fonction qui affiche le contenu d'un tableau créé par la fonction précédente.
- Elle devra être prototypée de la façon suivante :

```
void ft_show_tab(struct s_stock_str *par);
```

- La structure est la même que l'exercice précédent et sera dans le fichier **ft_stock_str.h** que nous vous fournirons, :
- Pour chaque élément du tableau :
 - la chaîne de caractères suivi d'un retour à la ligne
 - la taille suivi d'un retour à la ligne
 - la copie de la chaîne de caractères (qui aura pu être modifiée) suivi d'un retour à la ligne
- Nous testons votre fonction avec notre **ft_strs_to_tab** (exercice précédent). Prenez les mesures nécessaires pour que cela fonctionne !

Chapitre IX

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 09

Résumé: Ce document est le sujet du module C 09 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : libft	5
IV	Exercice 01 : Makefile	6
V	Exercice 02 : ft_split	8
VI	Rendu et peer-evaluation	9

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.



Pour cette journée, la norminette doit être lancée avec le flag `-R CheckForbiddenSourceHeader`. La moulinette l'utilisera aussi.

Chapitre II

Préambule

Dialogue issue du film The Big Lebowski :

Walter : Excuse-moi Smokey, t'as mordu la ligne : y'a faute.

Smokey : Mon cul ! Tu me mets huit, Duc...

Walter : Je te demande pardon, mets lui zéro, jeu suivant.

Smokey : Tu fais chier Walter, tu me mets huit Duc...

Walter : Smokey on est pas au Viet-nâm, on est au bowling, on joue selon les règles...

Le Duc : Allez, déconne pas Walter, on est là merde : son pied a légèrement mordu, il a un peu glissé... C'est qu'un sport, là !

Walter : Oui. Et il est homologué, c'est une partie qui compte pour le tournoi, j'ai pas raison ?

Smokey : Ouais, mais j'ai pas...

Walter : J'ai pas raison ?

Smokey : Ouais, mais j'ai pas mordu... Allez ! Vas-y Duc, mets moi un huit.

Walter : [Sort son flingue] Smokey mon ami, si t'as jamais souffert, tu vas comprendre...

Le Duc : Walter, fais pas le con !

Walter : [Menaçant] Vas-y, mets-toi un huit et tu vas comprendre !

Smokey : J'ai pas m...

Walter : Tu vas comprendre ta douleur, Smokey !

Smokey : Duc ? C'est ton partenaire...

Walter : Le monde est en train de DEVENIR CINGLÉ ! [Se lève] Y'A PERSONNE ICI À PART MOI QUI SE SOUCIE ENCORE DE RESPECTER LES RÈGLES ? METS-TOI ZÉRO !

Le Duc : Ils sont en train d'appeler les flics Walter remets ça dans...

Walter : [Braque son flingue sur Smokey] METS-TOI ZÉRO !

Le Duc : Range ca Walter...

Smokey : ...

Le Duc : Walter...

Walter : [Arme son flingue] TU CROIS PEUT-ÊTRE QUE JE PLAISANTE ?
METTS-TOI ZÉRO !

Smokey : Voila, je me suis mis zéro... T'es content ? Espèce de malade !

Walter : [Se calme] ... C'est un sport homologué.

Chapitre III

Exercice 00 : libft

	Exercice : 00
	libft
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : <i>libft_creator.sh</i> , <i>ft_putchar.c</i> , <i>ft_swap.c</i> , <i>ft_putstr.c</i> , <i>ft_strlen.c</i> , <i>ft_strcmp.c</i>	
Fonctions Autorisées : write	

- Créer votre bibliothèque **ft**. Elle s'appellera **libft.a**.
- Un shell script appelé **libft_creator.sh** compilera comme il le faut les fichiers sources et créera votre bibliothèque.
- Cette bibliothèque doit contenir toutes les fonctions suivantes :

```
void    ft_putchar(char c);
void    ft_swap(int *a, int *b);
void    ft_putstr(char *str);
int     ft_strlen(char *str);
int     ft_strcmp(char *s1, char *s2);
```

- Nous lancerons la commande suivante :

```
sh libft_creator.sh
```

Chapitre IV

Exercice 01 : Makefile

	Exercice : 01
	Makefile
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : Makefile	
Fonctions Autorisées : Aucune	

- Écrire le **Makefile** qui compile une librairie **libft.a**.
- Votre makefile doit afficher chaque commande qu'il effectue sans aucune fioriture.
- Votre makefile ne doit pas effectuer de commande inutile.
- Le **Makefile** ira chercher les fichiers sources dans le dossier **srcs**.
- Ces fichiers sources seront : **ft_putchar.c**, **ft_swap.c**, **ft_putstr.c**, **ft_strlen.c**, **ft_strcmp.c**
- Le **Makefile** ira chercher les fichiers headers dans le dossier **includes**.
- Ces fichiers headers seront : **ft.h**
- Il devra compiler vos fichiers **c** en utilisant **gcc** et les options **-Wall -Wextra -Werror** flags dans cet ordre.
- La lib sera à la racine de l'exercice.
- Les fichiers **.o** devront être à coté de leur fichier **.c** respectif.
- Le **Makefile** devra également implémenter des règles **clean**, **fclean**, **re**, la règle **all** et bien sur **libft.a**.
- Lancer juste **make** doit être équivalent à **make all**.
- La règle **all** devra faire comme **make libft.a**.
- la règle **clean** devra retirer tous les fichiers temporaires générés.
- La règle **fclean** fait l'équivalent d'un **make clean** et efface aussi le binaire créé

lors du `make all`.

- La règle `re` fait l'équivalent d'un `make fclean` puis un `make all`.
- Votre makefile ne doit pas recompiler les fichiers inutilement.
- Nous ne ramasserons que votre Makefile et testerons avec nos fichiers.



Attention aux wildcards!

Chapitre V

Exercice 02 : ft_split

	Exercice : 02
	ft_split
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_split.c
Fonctions Autorisées :	malloc

- Écrire une fonction qui découpe une chaîne de caractères en fonction d'une autre chaîne de caractères.
- Il faudra utiliser chaque caractère de la chaîne **charset** comme séparateur.
- La fonction renvoie un tableau où chaque case contient l'adresse d'une chaîne de caractères comprise entre deux séparateur. Le dernier élément du tableau devra être égal à 0 pour marquer la fin du tableau.
- Il ne doit pas y avoir de chaîne vide dans votre tableau. Tirez-en les conclusions qui s'imposent.
- La chaîne qui sera transmise ne sera pas modifiable.
- Elle devra être prototypée de la façon suivante :

```
char **ft_split(char *str, char *charset);
```

Chapitre VI

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 10

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

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : display_file	5
IV	Exercice 01 : cat	6
V	Exercice 02 : tail	7
VI	Exercice 03 : hexdump	8
VII	Rendu et peer-evaluation	9

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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

Pour bien commencer votre journée, voici quelques questions très simples :

- Que se passerait-il si je laissais allumé un sèche-cheveux alimenté en continu dans un cube étanche d'un mètre de côté ?
- Déverser de l'antimatière dans le réacteur de Tchernobyl quand il était en train de fondre aurait-il empêché sa fusion ?
- C'est possible de pleurer au point de se déshydrater ?
- Si tous les êtres humains disparaissaient de la surface du globe, au bout de combien de temps s'éteindrait la dernière source de lumière artificielle ?
- C'est vraiment dangereux de se baigner dans une piscine pendant un orage ?
- De quelle hauteur faudrait-il laisser tomber un steak pour qu'il soit cuit en arrivant au sol ?
- Quand la bande passante d'Internet dépassera-t-elle celle de FedEx, si elle y parvient un jour ?
- Combien de tweets différents sont possibles dans notre langue ? Et combien de temps faudrait-il à la population mondiale pour tous les lire à haute voix ?
- Quel serait le résultat si tous les candidats au code de la route répondraient au pif au questionnaire à choix multiple ? Combien répondraient juste à l'ensemble des questions ?

Questions extraites du livre ‘Et si ...?’ de Randall Munroe.

Chapitre III

Exercice 00 : display_file

	Exercice : 00
	display_file
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : Makefile , et les fichiers de votre programme	
Fonctions Autorisées : close , open , read , write	

- 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**. Le binaire s'appellera **ft_display_file**.
- La fonction **malloc** est interdite. Vous pouvez faire l'exercice uniquement en déclarant un tableau de taille fixe.
- Tous les fichiers passés en paramètre seront valides.
- Les messages d'erreurs devront être affichés sur la sortie leur étant réservée et en étant suivi d'un retour à la ligne.
- Si il n'y aucun argument, votre programme doit afficher

File name missing.

- Si il y a trop d'argument, votre programme doit afficher

Too many arguments.

- Si le fichier n'est pas lisible, votre programme doit afficher

Cannot read file.

Chapitre IV

Exercice 01 : cat

	Exercice : 01
	cat
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	Makefile, et les fichiers de votre programme
Fonctions Autorisées :	close, open, read, write, strerror, basename

- Écrire un programme appelé `ft_cat` qui réalise le même travail que la commande `cat` du système.
- Vous n'avez pas à gérer les options.
- Le répertoire de rendu aura un `Makefile` avec une règle `all`, une règle `clean`, et une règle `fclean`.
- Vous pouvez utiliser la variable `errno` (voir le `man` de `Errno`).
- Vous devriez aller lire les `man` de toutes les fonctions autorisées
- Vous pouvez faire l'exercice uniquement en déclarant un tableau de taille fixe. Ce tableau aura une taille limitée à un peu moins d'environ 30 ko. Pour que vous puissiez tester cette limitation, utilisez la commande `ulimit` dans votre shell.

Chapitre V

Exercice 02 : tail

	Exercice : 02
	tail
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : Makefile , et les fichiers de votre programme	
Fonctions Autorisées : close , open , read , write , malloc , free , strerror , basename	

- Écrire un programme appelé **ft_tail** qui réalise le même travail que la commande **tail**.
- Vous avez à gérer uniquement l'option **-c**, mais vous n'avez pas à gérer le '+' et le '-'.
- Tous les tests seront effectué avec l'option **-c**
- Le répertoire de rendu aura un **Makefile** avec une règle **all**, une règle **clean**, et une règle **fclean**.
- Vous pouvez utiliser la variable **errno**

Chapitre VI

Exercice 03 : hexdump

	Exercice : 03
	hexdump
Dossier de rendu :	<i>ex03/</i>
Fichiers à rendre :	Makefile, et les fichiers de votre programme
Fonctions Autorisées :	close, open, read, write, malloc, free, strerror, basename

- Écrire un programme appelé `ft_hexdump` qui réalise le même travail que la commande `hexdump` du système sans redirection.
- Vous n'avez à gérer que l'option `-C`.
- Le répertoire de rendu aura un `Makefile` avec une règle `all`, une règle `clean`, et une règle `fclean`.
- Vous pouvez utiliser la variable `errno` (voir le `man` de `errno`).

Chapitre VII

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



C Piscine

C 11

Summary: This document is the subject for the module C 11 of the C Piscine @ 42.

Version: 6

Contents

I	Instructions	2
II	Foreword	4
III	Exercise 00 : ft_foreach	6
IV	Exercise 01 : ft_map	7
V	Exercise 02 : ft_any	8
VI	Exercise 03 : ft_count_if	9
VII	Exercise 04 : ft_is_sort	10
VIII	Exercise 05 : do-op	11
IX	Exercise 06 : ft_sort_string_tab	13
X	Exercise 07 : ft_advanced_sort_string_tab	14
XI	Submission and peer-evaluation	15

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before submission.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for all your exercises.
- Your exercises will be checked and graded by your fellow classmates.
- On top of that, your exercises will be checked and graded by a program called Moulinette.
- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of work that doesn't pass `norminette`'s check.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We will not take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Using a forbidden function is considered cheating. Cheaters get `-42`, and this grade is non-negotiable.
- You'll only have to submit a `main()` function if we ask for a program.
- Moulinette compiles with these flags: `-Wall` `-Wextra` `-Werror`, and uses `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.

- Your reference guide is called Google / man / the Internet /
- Check out the "C Piscine" part of the forum on the intranet, or the slack Piscine.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- By Odin, by Thor ! Use your brain !!!

Chapter II

Foreword

Here's a little story :

(1982, California) Larry Walters of Los Angeles is one of the few to contend for the Darwin Awards and live to tell the tale. "I have fulfilled my 20-year dream," said Walters, a former truck driver for a company that makes TV commercials.

"I'm staying on the ground. I've proved the thing works."

Larry's boyhood dream was to fly. But fates conspired to keep him from his dream. He joined the Air Force, but his poor eyesight disqualifies him from the job of pilot. After he was discharged from the military, he sat in his backyard watching jets fly overhead.

He hatched his weather balloon scheme while sitting outside in his "extremely comfortable" Sears lawnchair. He purchased 45 weather balloons from an Army-Navy surplus store, tied them to his tethered lawnchair (dubbed the Inspiration I) and filled the four-foot diameter balloons with helium. Then, armed with some sandwiches, Miller Lite, and a pellet gun, he strapped himself into his lawnchair. He figured he would shoot to pop a few of the many balloons when it was time to descend.

Larry planned to sever the anchor and lazily float to a height of about 30 feet above the backyard, where he would enjoy a few hours of flight before coming back down. But things didn't work out quite as Larry planned.

When his friends cut the cord anchoring the lawnchair to his Jeep, he did not float lazily up to 30 feet. Instead he streaked into the LA sky as if shot from a cannon, pulled by the lift of 45 helium balloons, holding 33 cubic feet of helium each.

He didn't level off at 100 feet, nor did he level off at 1000 feet. After climbing and climbing, he leveled off at 16,000 feet.

At that height he felt he couldn't risk shooting any of the balloons, lest he unbalance the load and really find himself in trouble. So he stayed there, drifting cold and frightened with his beer and sandwiches, for more than 14 hours. He crossed the primary approach corridor of LAX, where startled Trans World Airlines and Delta Airlines pilots radioed in reports

of the strange sight.

Eventually he gathered the nerve to shoot a few balloons, and slowly descended. The hanging tethers tangled and caught in a power line, blacking out a Long Beach neighborhood for 20 minutes. Larry climbed to safety, where he was arrested by waiting members of the LAPD. As he was led away in handcuffs, a reporter dispatched to cover the daring rescue asked him why he had done it. Larry replied nonchalantly, "A man can't just sit around."

The Federal Aviation Administration was not amused. Safety Inspector Neal Savoy said, "We know he broke some part of the Federal Aviation Act, and as soon as we decide which part it is, a charge will be filed."

The moral of this story is Larry Walters should have stay on his chair and learn C....

Chapter III

Exercise 00 : ft_FOREACH

	Exercise 00
	ft_FOREACH
Turn-in directory :	<i>ex00/</i>
Files to turn in :	ft_FOREACH.c
Allowed functions :	None

- Create the function `ft_FOREACH` which, for a given ints array, applies a function on all elements of the array. This function will be applied following the array's order.
- Here's how the function should be prototyped :

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

- For example, the function `ft_FOREACH` could be called as follows in order to display all ints of the array :

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

Chapter IV

Exercise 01 : ft_map

	Exercise 01
	ft_map
Turn-in directory :	<i>ex01/</i>
Files to turn in :	ft_map.c
Allowed functions :	malloc

- Create the function **ft_map** which, for a given ints array, applies a function on all elements of the array (in order) and returns a array of all the return values.
- This function will be applied following the array's order.
- Here's how the function should be prototyped :

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

Chapter V

Exercise 02 : ft_any

	Exercise 02
	ft_any
Turn-in directory :	<i>ex02/</i>
Files to turn in :	ft_any.c
Allowed functions :	None

- Create a function **ft_any** which will return 1 if, passed to the function **f**, at least one element of the array returns something else than 0. Else, it should return 0.
- This function will be applied following the array's order.
- Here's how the function should be prototyped :

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

- The array will be delimited with a null pointer.

Chapter VI

Exercise 03 : ft_count_if

	Exercise 03
	ft_count_if
Turn-in directory :	<i>ex03/</i>
Files to turn in :	ft_count_if.c
Allowed functions :	None

- Create a function `ft_count_if` which will return the number of elements of the array that return does not return 0 when passed to the function `f`.
- This function will be applied following the array's order.
- Here's how the function should be prototyped :

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

Chapter VII

Exercise 04 : ft_is_sort

	Exercise 04
	ft_is_sort
Turn-in directory :	<i>ex04/</i>
Files to turn in :	ft_is_sort.c
Allowed functions :	None

- Create a function `ft_is_sort` which returns 1 if the array is sorted and 0 if it isn't.
- The function given as argument should return a negative integer if the first argument is lower than the second, 0 if they're equal or a positive integer for anything else.
- Here's how the function should be prototyped :

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

Chapter VIII

Exercise 05 : do-op

	Exercise 05
	do-op
Turn-in directory : <i>ex05/</i>	
Files to turn in : Your program files	
Allowed functions : write	

- Create a program called **do-op**.
- The program will be executed with three arguments: **do-op value1 operateur value2**
- Example :

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

- You should use an array of pointers to function to take care of the **operator**.
- In case of an invalid operator your program should print 0.
- If the number of arguments is invalid, **do-op** doesn't display anything.
- Your program should accept and print the result for the following operators: '+' '-' '/ '*' and '%'
- Your program should treat value as int.
- In case of a division by 0, it should print:

```
Stop : division by zero
```

- In case of a modulo by 0, it should print:

```
Stop : modulo by zero
```

- Here's an example of tests the Moulinette will run :

```
$> 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
$>
```

Chapter IX

Exercise 06 : ft_sort_string_tab

	Exercise 06
	ft_sort_string_tab
Turn-in directory :	<i>ex06/</i>
Files to turn in :	<code>ft_sort_string_tab.c</code>
Allowed functions :	None

- Create the function `ft_sort_string_tab`, by ascii order the strings in `tab`.
- `tab` will be null terminated
- The sorting will be performed by exchanging the array's pointers.
- Here's how it should be prototyped :

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

Chapter X

Exercise 07 : ft_advanced_sort_string_tab

	Exercise 07
	ft_advanced_sort_string_tab
Turn-in directory :	ex07/
Files to turn in :	ft_advanced_sort_string_tab.c
Allowed functions :	None

- Create the function `ft_advanced_sort_string_tab` which sorts, depending on the return of the function given as argument
- The sorting will be performed by exchanging the array's pointers.
- `tab` will be null terminated
- Here's how it should be prototyped :

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



Calling `ft_advanced_sort_string_tab()` with `ft_strcmp` as a second argument will return the same result as `ft_sort_string_tab()`.

Chapter XI

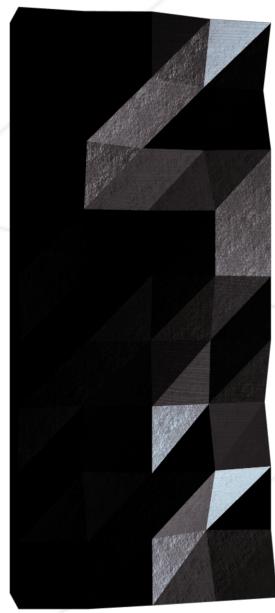
Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

As these assignments are not verified by a program, feel free to organize your files as you wish, as long as you turn in the mandatory files and comply with the requirements.



You need to return only the files requested by the subject of this project.



Piscine C

C 12

Résumé: Ce document est le sujet du module C 12 de la piscine C de 42.

Version:

Table des matières

I	Préambule	2
II	Consignes	4
III	Exercice 00 : ft_create_elem	6
IV	Exercice 01 : ft_list_push_front	7
V	Exercice 02 : ft_list_size	8
VI	Exercice 03 : ft_list_last	9
VII	Exercice 04 : ft_list_push_back	10
VIII	Exercice 05 : ft_list_push_strs	11
IX	Exercice 06 : ft_list_clear	12
X	Exercice 07 : ft_list_at	13
XI	Exercice 08 : ft_list_reverse	14
XII	Exercice 09 : ft_list_FOREACH	15
XIII	Exercice 10 : ft_list_FOREACH_if	16
XIV	Exercice 11 : ft_list_find	17
XV	Exercice 12 : ft_list_remove_if	18
XVI	Exercice 13 : ft_list_merge	19
XVII	Exercice 14 : ft_list_sort	20
XVIII	Exercice 15 : ft_list_reverse_fun	21
XIX	Exercice 16 : ft_sorted_list_insert	22
XX	Exercice 17 : ft_sorted_list_merge	23
XXI	Rendu et peer-evaluation	24

Chapitre I

Préambule

SPOILER ALERT

NE LISEZ PAS LA PAGE SUIVANTE

Vous l'aurez voulu.

- Dans **Star Wars**, Dark Vador est le père de Luke Skywalker.
- Dans **The Usual Suspects**, Verbal est Keyser Soze.
- Dans **Fight Club**, Tyler Durden et le narrateur sont la même personne.
- Dans **Sixième Sens**, Bruce Willis est mort depuis le début.
- Dans **Les Autres**, les habitants de la maison sont les fantômes et vice-versa.
- Dans **Bambi**, la mère de Bambi meurt.
- Dans **Le Village**, les monstres sont les villageois et l'action se situe, en réalité, dans notre époque.
- Dans **Harry Potter**, Dumbledore meurt.
- Dans **La Planète des Singes**, l'action se situe sur Terre.
- Dans **Le Trône de Fer**, Robb Stark et Joffrey Baratheon meurent le soir de leurs noces.
- Dans **Twilight**, les vampires brillent au soleil.
- Dans **Stargate SG-1, Saison 1, Episode 18**, O'Neill et Carter sont en Antarctique.
- Dans **The Dark Knight Rises**, Miranda Tate est Talia Al'Gul.
- Dans **Super Mario Bros**, la princesse est dans un autre château.

Chapitre II

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.
- Pour les exos sur les listes, on utilisera la structure suivante :

```
typedef struct          s_list
{
    struct s_list      *next;
    void               *data;
}                      t_list;
```

- Vous devez mettre cette structure dans un fichier `ft_list.h` et le rendre à chaque exercice.
- A partir de l'exercice 01 nous utiliserons notre `ft_create_elem`, prenez les dispositions nécessaires (il pourrait être intéressant d'avoir son prototype dans `ft_list.h`...).

Chapitre III

Exercice 00 : ft_create_elem

	Exercice : 00
	ft_create_elem
Dossier de rendu :	<i>ex00/</i>
Fichiers à rendre :	ft_create_elem.c, ft_list.h
Fonctions Autorisées :	malloc

- écrire la fonction `ft_create_elem` qui crée un nouvel élément de type `t_list`.
- Elle devra assigner `data` au paramètre fournis et `next` à `NULL`.
- Elle devra être prototypée de la façon suivante :

```
t_list *ft_create_elem(void *data);
```

Chapitre IV

Exercice 01 : ft_list_push_front

	Exercice : 01
	ft_list_push_front
Dossier de rendu :	<i>ex01/</i>
Fichiers à rendre :	<code>ft_list_push_front.c, ft_list.h</code>
Fonctions Autorisées :	<code>ft_create_elem</code>

- écrire la fonction `ft_list_push_front` qui ajoute au début de la liste un nouvel élément de type `t_list`.
- Elle devra assigner `data` au paramètre fourni.
- Elle mettra à jour, si nécessaire, le pointeur sur le début de liste.
- Elle devra être prototypée de la façon suivante :

```
void          ft_list_push_front(t_list **begin_list, void *data);
```

Chapitre V

Exercice 02 : ft_list_size

	Exercice : 02
	ft_list_size
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	ft_list_size.c, ft_list.h
Fonctions Autorisées :	Aucune

- écrire la fonction `ft_list_size` qui renvoie le nombre d'éléments dans la liste.
- Elle devra être prototypée de la façon suivante :

```
int ft_list_size(t_list *begin_list);
```

Chapitre VI

Exercice 03 : ft_list_last

	Exercice : 03
	ft_list_last
Dossier de rendu :	<i>ex03/</i>
Fichiers à rendre :	ft_list_last.c, ft_list.h
Fonctions Autorisées :	Aucune

- écrire la fonction `ft_list_last` qui renvoie le dernier élément de la liste.
- Elle devra être prototypée de la façon suivante :

```
t_list *ft_list_last(t_list *begin_list);
```

Chapitre VII

Exercice 04 : ft_list_push_back

	Exercice : 04
	ft_list_push_back
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : <i>ft_list_push_back.c, ft_list.h</i>	
Fonctions Autorisées : <i>ft_create_elem</i>	

- écrire la fonction `ft_list_push_back` qui ajoute à la fin de la liste un nouvel élément de type `t_list`.
- Elle devra assigner `data` au paramètre fourni.
- Elle mettra à jour, si nécessaire, le pointeur sur le début de liste.
- Elle devra être prototypée de la façon suivante :

```
void          ft_list_push_back(t_list **begin_list, void *data);
```

Chapitre VIII

Exercice 05 : ft_list_push_strs

	Exercice : 05
	ft_list_push_strs
Dossier de rendu :	<i>ex05/</i>
Fichiers à rendre :	ft_list_push_strs.c, ft_list.h
Fonctions Autorisées :	ft_create_elem

- écrire la fonction **ft_list_push_strs** qui crée une nouvelle liste en y mettant les chaînes de caractères pointées par les éléments tableau **strs**.
- **size** est la taille de **strs**
- Le premier élément du tableau se retrouvera à la fin de la liste.
- L'adresse du premier élément de la liste est renvoyée.
- Elle devra être prototypée de la façon suivante :

```
t_list *ft_list_push_strs(int size, char **strs);
```

Chapitre IX

Exercice 06 : ft_list_clear

	Exercice : 06
	ft_list_clear
Dossier de rendu :	<i>ex06/</i>
Fichiers à rendre :	ft_list_clear.c, ft_list.h
Fonctions Autorisées :	free

- écrire la fonction **ft_list_clear** qui retire et libère l'ensemble des éléments de la liste.
- Chaque **data** devra aussi être libéré à l'aide de **free_fct**
- Elle devra être prototypée de la façon suivante :

```
void ft_list_clear(t_list *begin_list, void (*free_fct)(void *));
```

Chapitre X

Exercice 07 : ft_list_at

	Exercice : 07
	ft_list_at
Dossier de rendu : <i>ex07/</i>	
Fichiers à rendre : ft_list_at.c, ft_list.h	
Fonctions Autorisées : Aucune	

- écrire la fonction **ft_list_at** qui renvoie le n-ième élément de la liste, sachant que le premier élément est l'élément 0.
- Elle renverra un pointeur nul en cas d'erreur.
- Elle devra être prototypée de la façon suivante :

```
t_list *ft_list_at(t_list *begin_list, unsigned int nbr);
```

Chapitre XI

Exercice 08 : ft_list_reverse

	Exercice : 08
	ft_list_reverse
Dossier de rendu :	<i>ex08/</i>
Fichiers à rendre :	ft_list_reverse.c
Fonctions Autorisées :	Aucune

- écrire la fonction **ft_list_reverse** qui inverse l'ordre des éléments de la liste.
Seuls les jeux de pointeurs sont admis.
- Attention dans cet exercice nous utiliserons notre propre **ft_list.h**
- Elle devra être prototypée de la façon suivante :

```
void ft_list_reverse(t_list **begin_list);
```

Chapitre XII

Exercice 09 : ft_list_FOREACH

	Exercice : 09
	ft_list_FOREACH
Dossier de rendu : <i>ex09/</i>	
Fichiers à rendre : ft_list_FOREACH.c, ft_list.h	
Fonctions Autorisées : Aucune	

- écrire la fonction **ft_list_FOREACH** qui applique une fonction donnée en paramètre à la valeur contenue dans chaque élément de la liste.
- **f** doit être appliquée dans l'ordre des éléments de la liste
- Elle devra être prototypée de la façon suivante :

```
void ft_list_FOREACH(t_list *begin_list, void (*f)(void *));
```

- La fonction pointée par **f** sera utilisée de la façon suivante :

```
(*f)(list_ptr->data);
```

Chapitre XIII

Exercice 10 : ft_list_FOREACH_if

	Exercice : 10
	ft_list_FOREACH_if
Dossier de rendu : <i>ex10/</i>	
Fichiers à rendre : <i>ft_list_FOREACH_if.c, ft_list.h</i>	
Fonctions Autorisées : Aucune	

- écrire la fonction `ft_list_FOREACH_if` qui applique une fonction donnée en paramètre à la valeur contenue dans certains éléments de la liste.
- `f` ne sera appliqué que sur les éléments qui passé en argument à `cmp` avec `data_ref`, `cmp` renvoie 0
- `f` doit être appliquée dans l'ordre des éléments de la liste
- Elle devra être prototypée de la façon suivante :

```
void ft_list_FOREACH_if(t_list *begin_list, void (*f)(void *), void  
*data_ref, int (*cmp)())
```

- Les fonctions pointées par `f` et par `cmp` seront utilisées de la façon suivante :

```
(*f)(list_ptr->data);  
(*cmp)(list_ptr->data, data_ref);
```



La fonction `cmp` pourrait être par exemple `ft_strcmp...`

Chapitre XIV

Exercice 11 : ft_list_find

	Exercice : 11
	ft_list_find
Dossier de rendu :	<i>ex11/</i>
Fichiers à rendre :	ft_list_find.c, ft_list.h
Fonctions Autorisées :	Aucune

- écrire la fonction **ft_list_find** qui renvoie l'adresse du premier élément dont la donnée comparée à **data_ref** à l'aide de **cmp**, fait que **cmp** renvoie 0.
- Elle devra être prototypée de la façon suivante :

```
t_list *ft_list_find(t_list *begin_list, void *data_ref, int (*cmp)());
```

- La fonctions pointée par **cmp** sera utilisée de la façon suivante :

```
(*cmp)(list_ptr->data, data_ref);
```

Chapitre XV

Exercice 12 : ft_list_remove_if

	Exercice : 12
	ft_list_remove_if
Dossier de rendu :	<i>ex12/</i>
Fichiers à rendre :	ft_list_remove_if.c, ft_list.h
Fonctions Autorisées :	free

- écrire la fonction **ft_list_remove_if** qui efface de la liste tous les éléments dont la donnée comparée à **data_ref** à l'aide de **cmp**, fait que **cmp** renvoie 0.
- La **data** d'un élément à effacer devra aussi être libérée à l'aide de **free_fct**
- Elle devra être prototypée de la façon suivante :

```
void ft_list_remove_if(t_list **begin_list, void *data_ref, int (*cmp)(), void (*free_fct)(void *));
```

- Les fonctions pointées par **free_fct** et par **cmp** seront utilisées de la façon suivante :

```
(*cmp)(list_ptr->data, data_ref);  
(*free_fct)(list_ptr->data);
```

Chapitre XVI

Exercice 13 : ft_list_merge

	Exercice : 13
	ft_list_merge
Dossier de rendu :	<i>ex13/</i>
Fichiers à rendre :	<code>ft_list_merge.c, ft_list.h</code>
Fonctions Autorisées :	Aucune

- écrire la fonction `ft_list_merge` qui met les éléments d'une liste `begin2` à la fin d'une autre liste `begin1`.
- La création d'éléments n'est pas autorisée.
- Elle devra être prototypée de la façon suivante :

```
void ft_list_merge(t_list **begin_list1, t_list *begin_list2);
```

Chapitre XVII

Exercice 14 : ft_list_sort

	Exercice : 14
	ft_list_sort
Dossier de rendu : <i>ex14/</i>	
Fichiers à rendre : ft_list_sort.c, ft_list.h	
Fonctions Autorisées : Aucune	

- écrire la fonction **ft_list_sort** qui trie par ordre croissant le contenu de la liste, en comparant deux éléments grâce à une fonction de comparaison de données des deux éléments.
- Elle devra être prototypée de la façon suivante :

```
void ft_list_sort(t_list **begin_list, int (*cmp)());
```

- La fonctions pointée par **cmp** sera utilisée de la façon suivante :

```
(*cmp)(list_ptr->data, other_list_ptr->data);
```



La fonction **cmp** pourrait être par exemple **ft_strcmp**.

Chapitre XVIII

Exercice 15 : ft_list_reverse_fun

	Exercice : 15
	ft_list_reverse_fun
Dossier de rendu :	<i>ex15/</i>
Fichiers à rendre :	ft_list_reverse_fun.c, ft_list.h
Fonctions Autorisées :	Aucune

- écrire la fonction `ft_list_reverse_fun` qui inverse l'ordre des éléments de la liste.
- Elle devra être prototypée de la façon suivante :

```
void ft_list_reverse_fun(t_list *begin_list);
```

Chapitre XIX

Exercice 16 : ft_sorted_list_insert

	Exercice : 16
	ft_sorted_list_insert
	Dossier de rendu : <i>ex16/</i>
	Fichiers à rendre : ft_sorted_list_insert.c, ft_list.h
	Fonctions Autorisées : ft_create_elem

- écrire la fonction **ft_sorted_list_insert** qui crée un nouvel élément et l'insère dans une liste triée de sorte que la liste reste triée par ordre croissant.
- Elle devra être prototypée de la façon suivante :

```
void ft_sorted_list_insert(t_list **begin_list, void *data, int (*cmp)());
```

- La fonctions pointée par **cmp** sera utilisée de la façon suivante :

```
(*cmp)(list_ptr->data, other_list_ptr->data);
```

Chapitre XX

Exercice 17 : ft_sorted_list_merge

	Exercice : 17
	ft_sorted_list_merge
Dossier de rendu : <i>ex17/</i>	
Fichiers à rendre : <i>ft_sorted_list_merge.c, ft_list.h</i>	
Fonctions Autorisées : Aucune	

- écrire la fonction `ft_sorted_list_merge` qui intègre les éléments d'une liste triée `begin2` dans une autre liste triée `begin1`, de sorte que la liste `begin1` reste triée par ordre croissant.
- Elle devra être prototypée de la façon suivante :

```
void ft_sorted_list_merge(t_list **begin_list1, t_list *begin_list2, int (*cmp)());
```

- La fonctions pointée par `cmp` sera utilisée de la façon suivante :

```
(*cmp)(list_ptr->data, other_list_ptr->data);
```

Chapitre XXI

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.



Piscine C

C 13

Résumé: Ce document est le sujet du module C 13 de la piscine C de 42.

Version:

Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : btree_create_node	5
IV	Exercice 01 : btree_apply_prefix	6
V	Exercice 02 : btree_apply_infix	7
VI	Exercice 03 : btree_apply_suffix	8
VII	Exercice 04 : btree_insert_data	9
VIII	Exercice 05 : btree_search_item	10
IX	Exercice 06 : btree_level_count	11
X	Exercice 07 : btree_apply_by_level	12
XI	Rendu et peer-evaluation	13

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Relisez bien le sujet avant de rendre vos exercices. A tout moment le sujet peut changer.
- 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. La Moulinette utilise le programme **norminette** pour vérifier la norme de vos fichiers. Comprendre par là qu'il est stupide de rendre un code qui ne passe pas la **norminette**.
- 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.
- L'utilisation d'une fonction interdite est un cas de triche. Toute triche est sanctionnée par la note de -42.
- Vous ne devrez rendre une fonction main() que si nous vous demandons un programme.
- La Moulinette compile avec les flags -Wall -Wextra -Werror, et utilise **gcc**.
- Si votre programme ne compile pas, vous aurez 0.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés 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, ainsi que sur le slack de votre Piscine !
- 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.
- Pour les exos d'aujourd'hui, on utilisera la structure suivante :

```
typedef struct          s_btreet
{
    struct s_btreet *left;
    struct s_btreet *right;
    void            *item;
}                      t_btreet;
```

- Vous devez mettre cette structure dans un fichier `ft_btreet.h` et le rendre à chaque exercice.
- A partir de l'exercice 01 nous utiliserons notre `btreet_create_node`, prenez les dispositions nécessaires (il pourrait être intéressant d'avoir son prototype dans `ft_btreet.h...`).

Chapitre II

Préambule

Voici la liste des releases de Venom :

- In League with Satan (single, 1980)
- Welcome to Hell (1981)
- Black Metal (1982)
- Bloodlust (single, 1983)
- Die Hard (single, 1983)
- Warhead (single, 1984)
- At War with Satan (1984)
- Hell at Hammersmith (EP, 1985)
- American Assault (EP, 1985)
- Canadian Assault (EP, 1985)
- French Assault (EP, 1985)
- Japanese Assault (EP, 1985)
- Scandinavian Assault (EP, 1985)
- Manitou (single, 1985)
- Nightmare (single, 1985)
- Possessed (1985)
- German Assault (EP, 1987)
- Calm Before the Storm (1987)
- Prime Evil (1989)
- Tear Your Soul Apart (EP, 1990)
- Temples of Ice (1991)
- The Waste Lands (1992)
- Venom '96 (EP, 1996)
- Cast in Stone (1997)
- Resurrection (2000)
- Anti Christ (single, 2006)
- Metal Black (2006)
- Hell (2008)
- Fallen Angels (2011)

Le sujet d'aujourd'hui est plus facile si vous travaillez en écoutant Venom.

Chapitre III

Exercice 00 : btree_create_node

	Exercice : 00
	btree_create_node
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : btree_create_node.c, ft_btree.h	
Fonctions Autorisées : malloc	

- Écrire la fonction **btree_create_node** qui alloue un nouvel élément, initialise son **item** à la valeur du paramètre et tous les autres éléments à 0.
- L'adresse de la node créée est renvoyée.
- Elle devra être prototypée de la façon suivante :

```
t_btreet *btree_create_node(void *item);
```

Chapitre IV

Exercice 01 : btree_apply_prefix

	Exercice : 01
	btree_apply_prefix
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : <i>btree_apply_prefix.c, ft_btreet.h</i>	
Fonctions Autorisées : Aucune	

- Écrire la fonction `btree_apply_prefix` qui applique la fonction passée en paramètre à l'`item` de chaque node, en parcourant l'arbre de manière `prefix`.
- Elle devra être prototypée de la façon suivante :

```
void btree_apply_prefix(t_btreet *root, void (*applyf)(void *));
```

Chapitre V

Exercice 02 : btree_apply_infix

	Exercice : 02
	btree_apply_infix
Dossier de rendu :	<i>ex02/</i>
Fichiers à rendre :	<code>btree_apply_infix.c, ft_btreet.h</code>
Fonctions Autorisées :	Aucune

- Écrire la fonction `btree_apply_infix` qui applique la fonction passée en paramètre à l'`item` de chaque node, en parcourant l'arbre de manière `infix`.
- Elle devra être prototypée de la façon suivante :

```
void btree_apply_infix(t_btreet *root, void (*applyf)(void *));
```

Chapitre VI

Exercice 03 : btree_apply_suffix

	Exercice : 03
	btree_apply_suffix
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : <i>btree_apply_suffix.c, ft_btreet.h</i>	
Fonctions Autorisées : Aucune	

- Écrire la fonction `btree_apply_suffix` qui applique la fonction passée en paramètre à l'`item` de chaque node, en parcourant l'arbre de manière `suffix`.
- Elle devra être prototypée de la façon suivante :

```
void btree_apply_suffix(t_btreet *root, void (*applyf)(void *));
```

Chapitre VII

Exercice 04 : btree_insert_data

	Exercice : 04
	btree_insert_data
Dossier de rendu :	<i>ex04/</i>
Fichiers à rendre :	btree_insert_data.c, ft_btreetree.h
Fonctions Autorisées :	btree_create_node

- Écrire la fonction **btree_insert_data** qui insère l'élément **item** dans un arbre. L'arbre passé en paramètre sera trié : pour chaque **node** tous les éléments inférieurs se situent dans la partie gauche et tous les éléments supérieurs ou égaux à droite. On enverra en paramètre une fonction de comparaison ayant le même comportement que **strcmp**.
- Le paramètre **root** pointe sur le noeud racine de l'arbre. Lors du premier appel, il pointe sur **NULL**.
- Elle devra être prototypée de la façon suivante :

```
void btree_insert_data(t_btreetree **root, void *item, int (*cmpf)(void *, void *));
```

Chapitre VIII

Exercice 05 : btree_search_item

	Exercice : 05
	btree_search_item
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : btree_search_item.c, ft_btree.h	
Fonctions Autorisées : Aucune	

- Écrire la fonction **btree_search_item** qui retourne le premier élément correspondant à la donnée de référence passée en paramètre. L'arbre devra être parcouru de manière **infix**. Si l'élément n'est pas trouvé, la fonction devra retourner **NULL**.
- Elle devra être prototypée de la façon suivante :

```
void *btree_search_item(t_btree *root, void *data_ref, int (*cmpf)(void *, void *));
```

Chapitre IX

Exercice 06 : btree_level_count

	Exercice : 06
	btree_level_count
Dossier de rendu :	<i>ex06/</i>
Fichiers à rendre :	btree_level_count.c, ft_btreet.h
Fonctions Autorisées :	Aucune

- Écrire la fonction `btree_level_count` qui retourne la taille de la plus grande branche passée en paramètre.
- Elle devra être prototypée de la façon suivante :

```
int btree_level_count(t_btreet *root);
```

Chapitre X

Exercice 07 : btree_apply_by_level

	Exercice : 07
	btree_apply_by_level
Dossier de rendu : <i>ex07/</i>	
Fichiers à rendre : <i>btree_apply_by_level.c, ft_btreet.h</i>	
Fonctions Autorisées : <i>malloc, free</i>	

- Écrire la fonction `btree_apply_by_level` qui applique la fonction passée en paramètre à chaque noeud de l'arbre. L'arbre doit être parcouru étage par étage. La fonction appelée prendra trois paramètres :
 - Le premier paramètre, de type `void *`, correspond à l'item du node ;
 - Le second paramètre, de type `int`, correspond au niveau sur lequel on se trouve : 0 pour le root, 1 pour ses enfants, 2 pour ses petits-enfants, etc. ;
 - Le troisième paramètre, de type `int`, vaut 1 s'il s'agit du premier node du niveau, 0 sinon.
- Elle devra être prototypée de la façon suivante :

```
void btree_apply_by_level(t_btreet *root, void (*applyf)(void *item, int current_level, int is_first_elem))
```

Chapitre XI

Rendu et peer-evaluation

Rendez votre travail sur votre dépôt **Git** comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.



Vous ne devez rendre uniquement les fichiers demandés par le sujet de ce projet.