



# Piscine C

Jour 11

Staff 42 [piscine@42.fr](mailto:piscine@42.fr)

*Résumé: Ce document est le sujet du jour 11 de la piscine C de 42.*

# Table des matières

I	Consignes	2
II	Préambule	4
III	Exercice 00 : ft_create_elem	6
IV	Exercice 01 : ft_list_push_back	7
V	Exercice 02 : ft_list_push_front	8
VI	Exercice 03 : ft_list_size	9
VII	Exercice 04 : ft_list_last	10
VIII	Exercice 05 : ft_list_push_params	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

# Chapitre I

## Consignes

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

### 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 nocces.
- 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 III

## Exercice 00 : ft\_create\_elem


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

- 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\_back

	Exercice : 01
ft_list_push_back	
Dossier de rendu : ex01/	
Fichiers à rendre : ft_list_push_back.c, ft_list.h	
Fonctions Autorisées : ft_create_elem	
Remarques : n/a	


- 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 V

## Exercice 02 : ft\_list\_push\_front


	Exercice : 02
ft_list_push_front	
Dossier de rendu : ex02/	
Fichiers à rendre : ft_list_push_front.c, ft_list.h	
Fonctions Autorisées : ft_create_elem	
Remarques : n/a	

- 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 VI

## Exercice 03 : ft\_list\_size


	Exercice : 03
	ft_list_size
	Dossier de rendu : <i>ex03/</i>
	Fichiers à rendre : <code>ft_list_size.c</code> , <code>ft_list.h</code>
	Fonctions Autorisées : Aucune
	Remarques : n/a

- 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 VII

## Exercice 04 : ft\_list\_last


	Exercice : 04
	ft_list_last
	Dossier de rendu : <i>ex04/</i>
	Fichiers à rendre : <b>ft_list_last.c</b> , <b>ft_list.h</b>
	Fonctions Autorisées : Aucune
	Remarques : n/a

- 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 VIII

## Exercice 05 : ft\_list\_push\_params


	Exercice : 05
ft_list_push_params	
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : <code>ft_list_push_params.c</code> , <code>ft_list.h</code>	
Fonctions Autorisées : <code>ft_create_elem</code>	
Remarques : n/a	

- Écrire la fonction `ft_list_push_params` qui crée une nouvelle liste en y mettant les paramètres de la ligne de commande.
- Le premier argument se retrouvera à la fin de la liste.
- L'adresse du premier maillon de la liste est renvoyée.
- Elle devra être prototypée de la façon suivante :

```
t_list *ft_list_push_params(int ac, char **av);
```

# Chapitre IX

## Exercice 06 : ft\_list\_clear


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

- Écrire la fonction **ft\_list\_clear** qui détruit l'ensemble des maillons de la liste.
- Elle assignera ensuite le pointeur sur la liste à nul.
- Elle devra être prototypée de la façon suivante :

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

# Chapitre X

## Exercice 07 : ft\_list\_at


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

- Écrire la fonction `ft_list_at` qui renvoie le n-ième élément de la liste.
- 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 : <code>ft_list_reverse.c</code> , <code>ft_list.h</code>
	Fonctions Autorisées : Aucune
	Remarques : n/a

- Écrire la fonction `ft_list_reverse` qui inverse l'ordre des éléments de la liste. Seuls les jeux de pointeurs sont admis.
- 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 : <code>ft_list_foreach.c</code> , <code>ft_list.h</code>
	Fonctions Autorisées : Aucune
	Remarques : n/a

- Écrire la fonction `ft_list_foreach` qui applique une fonction donnée en paramètre à l'information contenue dans chaque maillon 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 : ex10/
	Fichiers à rendre : ft_list_foreach_if.c, ft_list.h
	Fonctions Autorisées : Aucune
	Remarques : n/a

- Écrire la fonction `ft_list_foreach_if` qui applique une fonction donnée en paramètre à l'information contenue dans certains maillons de la liste. Une information de référence ainsi qu'une fonction de comparaison nous permettent de sélectionner les bons maillons de la liste : ceux qui sont "égaux" avec l'information de référence.
- 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 : <code>ft_list_find.c</code> , <code>ft_list.h</code>
	Fonctions Autorisées : Aucune
	Remarques : n/a

- Écrire la fonction `ft_list_find` qui renvoie l'adresse du premier maillon dont la donnée est "égale" à la donnée de référence.
- Elle devra être prototypée de la façon suivante :

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

# Chapitre XV

## Exercice 12 : ft\_list\_remove\_if


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

- Écrire la fonction `ft_list_remove_if` qui efface de la liste tous les éléments dont la donnée est "égale" à la donnée de référence.
- Elle devra être prototypée de la façon suivante :

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

# 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</code> , <code>ft_list.h</code>
	Fonctions Autorisées : Aucune
	Remarques : n/a

- 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 : <code>ft_list_sort.c</code> , <code>ft_list.h</code>
	Fonctions Autorisées : Aucune
	Remarques : n/a

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


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



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 : <code>ft_list_reverse_fun.c</code> , <code>ft_list.h</code>	
Fonctions Autorisées : Aucune	
Remarques : n/a	

- Écrire la fonction `ft_list_reverse_fun` qui inverse l'ordre des éléments de la liste. Seuls les jeux de pointeurs sont admis.
- 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 : ex16/	
Fichiers à rendre : ft_sorted_list_insert.c, ft_list.h	
Fonctions Autorisées : ft_create_elem	
Remarques : n/a	

- 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)());
```

# Chapitre XX

## Exercice 17 : ft\_sorted\_list\_merge

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

- 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)());
```