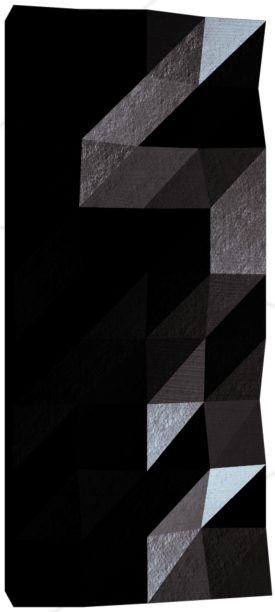

Piscine C

C 12

Staff 42 piscine@42.fr

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

Table des matières

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 specifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec

votre voisin de gauche.

- Votre manuel de référence s'appelle `Google / man / Internet / ...`.
- Pensez à discuter sur le forum Piscine de votre Intra, 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 : *ex*00/	
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 : $ex01/$	
Fichiers à rendre : `ft_list_push_front.c, ft_list.h`	
Fonctions Autorisées : `ft_create_elem`	

- 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 nécéssaire, 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 : $ex02/$	
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 : $ex03/$	
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 : *ex*04/	
Fichiers à rendre : `ft_list_push_back.c, ft_list.h`	
Fonctions Autorisées : `ft_create_elem`	

- 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 : $ex05/$	
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 chaines de caracteres 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 : *ex*06/	
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 : *ex*07/	
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 : *ex*08/	
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 : *ex*09/	
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 : *ex*10/	
Fichiers à rendre : `ft_list_foreach_if.c, ft_list.h`	
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 : $ex11/$	
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 : *ex*12/	
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 : $ex13/$	
Fichiers à rendre : `ft_list_merge.c, ft_list.h`	
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 : *ex*14/	
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 : *ex*15/	
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 : $ex16/$	
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 : *ex*17/	
Fichiers à rendre : `ft_sorted_list_merge.c, ft_list.h`	
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);
```