



Piscine

13

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

Table des matières

I	Consignes	2
II	Pré mbule	4
III	Exercice 00 : btree__cre__te__node	5
IV	Exercice 01 : btree__pply__prefix	6
V	Exercice 02 : btree__pply__infix	7
VI	Exercice 03 : btree__pply__suffix	8
VII	Exercice 04 : btree__insert__d__t	9
VIII	Exercice 05 : btree__se__rch__item	10
IX	Exercice 06 : btree__level__count	11
X	Exercice 07 : btree__pply__by__level	12

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. 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. Comprenez 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 t_s_btree
{
    struct t_s_btree *left;
    struct t_s_btree *right;
    void *item;
} t_btree;
```

- Vous devez mettre cette structure dans un fichier **ft_btree.h** et le rendre à chaque exercice.
- partir de l'exercice 01 nous utiliserons notre **btree_create_node**, prenez les dispositions nécessaires (il pourrait être intéressant d'avoir son prototype dans **ft_btree.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)
- 1st 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_btree *btree_create_node(void *item);
```

Chapitre IV

Exercice 01 : btree__ply_prefix


	Exercice : 01
btree__ply_prefix	
Dossier de rendu : ex01	
Fichiers à rendre : btree__ply_prefix.c, ft_btree.h	
Fonctions autorisées : aucune	

- Écrire la fonction `btree__ply_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__ply_prefix(t_btree *root, void (*plyf)(void *));
```

Chapitre V

Exercice 02 : btree__ pply_infix


	Exercice : 02
btree_apply_infix	
Dossier de rendu : <i>ex02</i>	
Fichiers à rendre : btree_apply_infix.c , ft_btree.h	
Fonctions autorisées : ucune	

- 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_pply_infix(t_btree *root, void (* pplyf)(void *));
```


Chapitre VI

Exercice 03 : btree__ply_suffix


	Exercice : 03
btree__ply_suffix	
Dossier de rendu : <i>ex03</i>	
Fichiers à rendre : <code>btree__ply_suffix.c</code> , <code>ft_btree.h</code>	
Fonctions autorisées : <code>ucune</code>	

- Écrire la fonction `btree__ply_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__ply_suffix(t_btree *root, void (*plyf)(void *));
```

Chapitre VII

Exercice 04 : btree_insert_data


	Exercice : 04
btree_insert_data	
Dossier de rendu : ex04	
Fichiers à rendre : btree_insert_data.c, ft_btree.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_btree **root, void *item, int (*cmpf)(void *, void *));
```

Chapitre VIII

Exercice 05 : btree_search_item


	Exercice : 05
btree_search_item	
Dossier de rendu : ex05	
Fichiers à rendre : btree_search_item.c, ft_btree.h	
Fonctions autorisées : ucune	

- 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 *d_t_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_btree.h	
Fonctions autorisées : ucune	

- 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_btree *root);
```

Chapitre X

Exercice 07 : btree__pply_by_level

	Exercice : 07
btree__pply_by_level	
Dossier de rendu : ex07	
Fichiers à rendre : btree__pply_by_level.c, ft_btree.h	
Fonctions autorisées : malloc, free	

- Écrire la fonction `btree__pply_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__pply_by_level(t_btree *root, void (* pplyf)(void *item, int current_level, int is_first_elem))
```