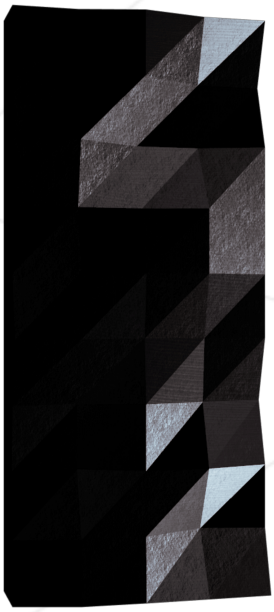

Piscine C

Jour 13

Staff 42 piscine@42.fr

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

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	Consignes intermédiaires	13
XII	Exercice 08 : rb_insert	14
XIII	Exercice 09 : rb_remove	15

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. Comprendre 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 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 !
- 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_btree
{
        struct s_btree  *left;
        struct s_btree  *right;
        void            *item;
}                       t_btree;
```

- Vous devez mettre cette structure dans un fichier `ft_btree.h` et le rendre à chaque exercice.
- A 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)
- 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 : *ex*00/	
Fichiers à rendre : `btree_create_node.c, ft_btree.h`	
Fonctions Autorisées : `malloc`	
Remarques : `n/a`	

- 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_apply_prefix

	Exercice : 01
btree_apply_prefix	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `btree_apply_prefix.c, ft_btree.h`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- 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_btree *root, void (*applyf)(void *));
```

Chapitre V

Exercice 02 : btree_apply_infix

	Exercice : 02
btree_apply_infix	
Dossier de rendu : *ex*02/	
Fichiers à rendre : `btree_apply_infix.c, ft_btree.h`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- 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_btree *root, void (*applyf)(void *));
```

Chapitre VI

Exercice 03 : btree_apply_suffix

	Exercice : 03
btree_apply_suffix	
Dossier de rendu : *ex*03/	
Fichiers à rendre : `btree_apply_suffix.c, ft_btree.h`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- 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_btree *root, void (*applyf)(void *));
```

Chapitre VII

Exercice 04 : btree_insert_data

	Exercice : 04
btree_insert_data	
Dossier de rendu : *ex*04/	
Fichiers à rendre : `btree_insert_data.c, ft_btree.h`	
Fonctions Autorisées : `btree_create_node`	
Remarques : `n/a`	Text

- 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 élements 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 : *ex*05/	
Fichiers à rendre : `btree_search_item.c, ft_btree.h`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- Écrire la fonction `btree_search_item` qui retourne le premier élèment 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 : *ex*06/	
Fichiers à rendre : `btree_level_count.c, ft_btree.h`	
Fonctions Autorisées : `Aucune`	
Remarques : `n/a`	

- 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_apply_by_level

	Exercice : 07
btree_apply_by_level	
Dossier de rendu : *ex*07/	
Fichiers à rendre : `btree_apply_by_level.c, ft_btree.h`	
Fonctions Autorisées : `malloc, free`	
Remarques : `n/a`	

- 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_btree *root, void (*applyf)(void *item, int current_level, int is_first_elem))
```

Chapitre XI

Consignes intermédiaires

- Nous allons maintenant travailler avec des arbres rouges et noirs.

```
enum    e_rb_color
{
  RB_BLACK,
  RB_RED
};

typedef struct s_rb_node
{
  struct s_rb_node *parent;
  struct s_rb_node *left;
  struct s_rb_node *right;
  void *data;
  enum e_rb_color  color;
} t_rb_node;
```

- Note : cette structure reprend à son début les mêmes champs que la structure précédente. Il est ainsi possible de réutiliser les fonctions déjà écrites avec les arbres rouges et noirs. Pour ceux qui ont un peu d'avance, il s'agit ici d'une forme rudimentaire de polymorphisme en C.
- Vous devez mettre cette structure dans un fichier `ft_btree_rb.h` et le rendre à chaque exercice.

Chapitre XII

Exercice 08 : rb_insert

	Exercice : 08
rb_insert	
Dossier de rendu : *ex*08/	
Fichiers à rendre : `rb_insert.c, ft_btree_rb.h`	
Fonctions Autorisées : `malloc`	
Remarques : `n/a`	

- Écrire la fonction `rb_insert` qui ajoute une nouvelle donnée dans l'arbre de manière à ce qu'il continue de respecter les contraintes d'un arbre rouge et noir. Le paramètre `root` pointe sur le noeud racine de l'arbre. Lors du premier appel, il pointe sur `NULL`. On enverra aussi en paramètre une fonction de comparaison ayant le même comportement que `strcmp`.
- Elle devra être prototypée de la façon suivante :

```
void rb_insert(struct s_rb_node **root, void *data, int (*cmpf)(void *, void *));
```

Chapitre XIII

Exercice 09 : rb_remove

	Exercice : 09
rb_remove	
Dossier de rendu : *ex*09/	
Fichiers à rendre : `rb_remove.c, ft_btree_rb.h`	
Fonctions Autorisées : `free`	
Remarques : `n/a`	

- Écrire la fonction `rb_remove` qui supprime une donnée dans l'arbre de manière à ce qu'il continue de respecter les contraintes d'un arbre rouge et noir. Le paramètre `root` pointe sur le noeud racine de l'arbre. On enverra aussi en paramètre une fonction de comparaison ayant le même comportement que `strcmp`, ainsi qu'une pointeur sur fonction `freef` qui sera appelée avec en paramètre l'élément de l'arbre à supprimer.
- Elle devra être prototypée de la façon suivante :

```
void rb_remove(struct s_rb_node **root, void *data, int (*cmpf)(void *, void *), void (*freef)(void *));
```