C Piscine

Day 13

Staff 42 pedago@42.fr

Summary: This document is the subject for Day13 of the C Piscine @ 42.

Contents

Chapter 1

Instructions

- The exercises are carefully laid out in order of difficulty, from easiest to hardest. An exercise is only graded if all previous ones are correct. In other words: the grading for a day stops at the first mistake.
- Be mindful of the submission procedures indicated at the start of every exercise.
- 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. Be as thorough as possible!
- Moulinette relies on a program called **norminette** to check if your files respect the Norm. An exercise containing files that do not respect the Norm will be graded 0.
- Using a forbidden function is considered cheating. Cheaters get -42, and this grade is non-negotiable.
- If **ft_putchar()** is an authorized function, we will compile your code with our **ft_putchar.c**.
- 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, it will be graded 0.
- You should not leave any additional file in your directory than those specified in the subject.

For this day, norminette is launched without any particular flag!

The forewords are entirely unrelated to the subjects and can safely be ignored.

Chapter 2

Topics

Today, you will have to learn about:

- Binary trees
- Tree traversal
- Red-black trees

For the following exercises, we'll use the following structure :

```
typedef struct            s_btree
{
        struct s_btree    *left;
        struct s_btree    *right;
        void              *item;
}                         t_btree;
```

You'll have to include this structure in a file **ft_btree.h** and submit it for each exercise.

From exercise 01 onward, we'll use our own btree_create_node, so make sure to include its prototype in **ft_btree.h**.

Chapter 3

Foreword

Here's the list of releases for **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)**

Today's subject will seem easier if you listen to **Venom**.

Chapter 4

Exercise 00 : btree_create_node

Turn-in directory : `ex00/`

Files to turn in: `btree_create_node.c, ft_btree.h`

Allowed functions: `malloc`

- Create the function **btree_create_node** which allocates a new element. It should initialise its **item** to the argument's value, and all other elements to 0.
- The created node's address is returned.
- Here's how it should be prototyped :

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

Chapter 5

Exercise 01 : btree_apply_prefix

Turn-in directory : `ex01/`

Files to turn in: `btree_apply_prefix.c, ft_btree.h`

Allowed functions: `None`

- Create a function **btree_apply_prefix** which applies the function given as argument to the **item** of each node, using **prefix traversal** to search the tree.
- Here's how it should be prototyped :

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

Chapter 6

Exercise 02 : btree_apply_infix

Turn-in directory : `ex02/`

Files to turn in: `btree_apply_infix.c, ft_btree.h`

Allowed functions: `None`

- Create a function **btree_apply_infix** which applies the function given as argument to the **item** of each node, using **infix traversal** to search the tree.
- Here's how it should be prototyped :

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

Chapter 7

Exercise 03 : btree_apply_suffix

Turn-in directory : `ex03/`

Files to turn in: `btree_apply_suffix.c, ft_btree.h`

Allowed functions: `None`

- Create a function **btree_apply_suffix** which applies the function given as argument to the **item** of each node, using **suffix traversal** to search the tree.
- Here's how it should be prototyped :

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

Chapter 8

Exercise 04 : btree_insert_data

Turn-in directory : ex04/
Files to turn in: btree_insert_data.c, ft_btree.h
Allowed functions: btree_create_node

- Create a function **btree_insert_data** which inserts the element **item** into a tree. The tree passed as argument will be sorted : for each **node** all lower elements are located on the left side and all higher or equal elements on the right. We'll also pass a comparison function similar to **strcmp** as argument.
- The **root** parameter points to the root node of the tree. First time called, it should point to **NULL**.
- Here's how it should be prototyped :

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

Chapter 9

Exercise 05 : btree_search_item

Turn-in directory : ex05/
Files to turn in: btree_search_item.c, ft_btree.h
Allowed functions: None

- Create a function **btree_search_item** which returns the first element related to the reference data given as argument. The tree should be browsed using **infix traversal** . If the element isn't found, the function should return **NULL**.
- Here's how it should be prototyped :

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

Chapter 10

Exercise 06 : btree_level_count

Turn-in directory : ex06/

Files to turn in: btree_level_count.c, ft_btree.h

Allowed functions: None

- Create a function **btree_level_count** which returns the size of the largest branch passed as argument.
- Here's how it should be prototyped :

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

Chapter 11

Exercise 07 : btree_apply_by_level

Turn-in directory : `ex07/`
Files to turn in: `btree_apply_by_level.c, ft_btree.h`
Allowed functions: `malloc, free`

- Create a function **btree_apply_by_level** which applies the function passed as argument to each node of the tree. The tree must be browsed level by level. The function called will take three arguments :
    - The first argument, of type **void ***, will correspond to the node's item ;
    - The second argument, of type **int**, corresponds to the level on which we find : 0 for root, 1 for children, 2 for grand-children, etc. ;
    - The third argument, of type **int**, is worth 1 if it's the first **node** of the level, or worth 0 otherwise.
- Here's how it should be prototyped :

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

Chapter 12

Provisional instructions

- Let's now work with red and black trees.

```
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 : this structure begins with the same fields as the previous structure. Therefore making it possible to use the already written functions with red and black trees again. For those of you that are are more experienced, this is a rudimentary form of polymorphism in C.
- You submit this structure for each exercise in a file called **ft_btree_rb.h**.

Chapter 13

Exercise 08: rb_insert

Turn-in directory : `ex08/`

Files to turn in: `rb_insert.c, ft_btree_rb.h`

Allowed functions: `malloc`

- Create a function **rb_insert** that adds a new data to the the tree so that it continues to respect a red and black tree's restrictions. The argument **root** points to the tree's root node. Upon first call, it points to **NULL**. We'll also pass a comparison function similar to **strcmp** as argument.
- Here's how it should be prototyped :

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

Chapter 14

Exercise 09: rb_remove

Turn-in directory : `ex09/`

Files to turn in: `rb_remove.c, ft_btree_rb.h`

Allowed functions: `free`

- Create a function **rb_remove** which removes a data from the the tree so that it continues to respect a red and black tree's restrictions. The argument **root** points to the tree's root node. We'll also pass a comparison function similar to **strcmp** as argument, as well as a pointer to function **freef** which will be called, with the element of the tree to be deleted, as argument.
- Here's how it should be prototyped :

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