

C Piscine C 13

Summary: This document is the subject for the module C 13 of the C Piscine @ 42.

Contents

1	mstructions	4
II	Foreword	4
III	Exercise 00 : btree_create_node	5
IV	Exercise 01 : btree_apply_prefix	6
V	Exercise 02 : btree_apply_infix	7
VI	Exercise 03 : btree_apply_suffix	8
VII	Exercise 04 : btree_insert_data	9
VIII	Exercise 05 : btree_search_item	10
IX	Exercise 06 : btree_level_count	11
\mathbf{X}	Exercise 07 : btree_apply_by_level	12

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before submission.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for all your exercises.
- 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. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called norminette to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of work that doesn't pass norminette's check.
- These exercises are carefully laid out by order of difficulty from easiest to hardest. We will not take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Using a forbidden function is considered cheating. Cheaters get -42, and this grade is non-negotiable.
- 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, you'll get 0.
- You <u>cannot</u> leave <u>any</u> additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.

C Piscine

- ullet Your reference guide is called Google / man / the Internet /
- Check out the "C Piscine" part of the forum on the intranet, or the slack Piscine.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- By Odin, by Thor! Use your brain!!!
- For the following exercises, we'll use the following structure :

- 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 btree_create_node, so make arrangements (it could be useful to have its prototype in a file ft_btree.h...).

Chapter II

Foreword

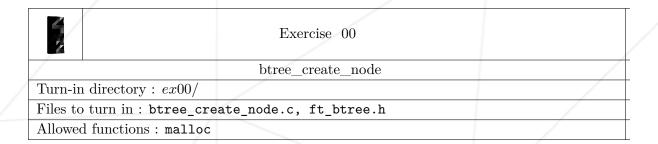
Here's the list of releases for Venom:

- 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 III

Exercise 00: btree_create_node

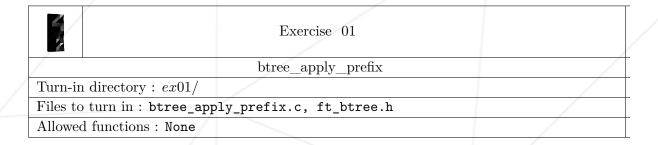


- Create the function btree_create_node which allocates a new element. It should initialise its item to the argument's value, and all other elements to 0.
- The created node's address is returned.
- Here's how it should be prototyped:

t_btree *btree_create_node(void *item);

Chapter IV

Exercise 01: btree_apply_prefix



- 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 V

Exercise 02: btree_apply_infix

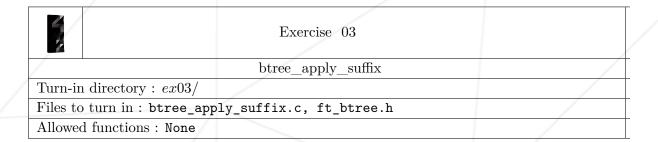
	Exercise 02	
/	btree_apply_infix	
Turn-in directory : ex		
Files to turn in: btre		
Allowed functions: N		

- 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 VI

Exercise 03: btree_apply_suffix

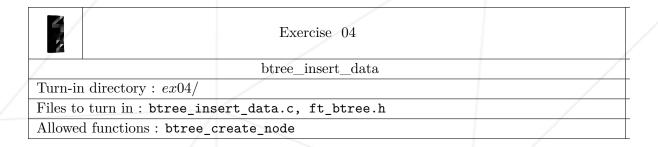


- Create a function btree_apply_suffix which applies the function given as argument to the item of each node, using suffix traversal to search the tree.
- Here's how it should be prototyped:

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

Chapter VII

Exercise 04: btree_insert_data

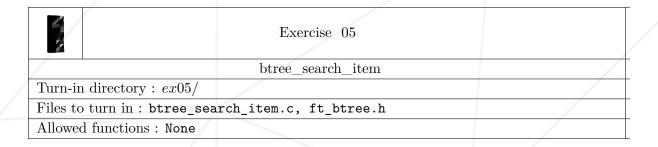


- Create a function btree_insert_data which inserts the element item into a tree. The tree passed as argument will be sorted: for each node all lower elements are located on the left side and all higher or equal elements on the right. We'll also pass a comparison function similar to strcmp as argument.
- 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 VIII

Exercise 05: btree_search_item

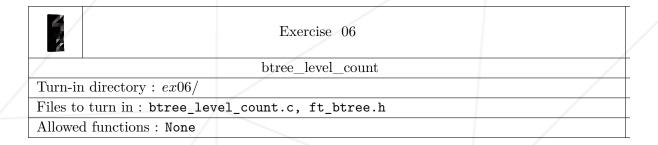


- Create a function btree_search_item which returns the first element related to the reference data given as argument. The tree should be browsed using infix traversal. If the element isn't found, the function should return NULL.
- Here's how it should be prototyped:

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

Chapter IX

Exercise 06: btree_level_count



- 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 X

Exercise 07: btree_apply_by_level

	Exercise 07	
/	btree_apply_by_level	
Turn-in directory : $ex07/$		
Files to turn in : btree_app	/	
Allowed functions: malloc,		

- 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