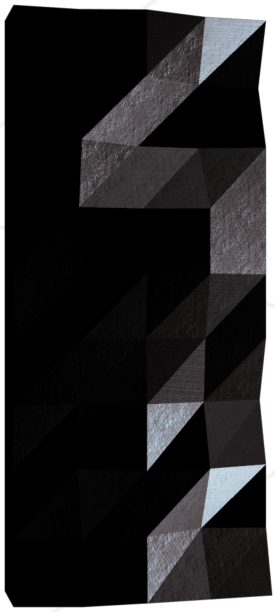

C Piscine

C 13

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

Version: 6.0

Contents

Chapter I

Instructions

- Only this page serves as your reference, do not trust rumors.
- Watch out! This document may change before submission.
- Ensure you have the appropriate permissions on your files and directories.
- You must follow the **submission procedures** for all your exercises.
- Your exercises will be checked and graded by your fellow classmates.
- Additionally, your exercises will be evaluated by a program called **Moulinette**.
- **Moulinette** is meticulous and strict in its assessment. It is fully automated, and there is no way to negotiate with it. To avoid unpleasant surprises, be as thorough as possible.
- **Moulinette** is not open-minded. If your code does not adhere to the Norm, it won't attempt to understand it. **Moulinette** relies on a program called **norminette** to check if your files comply with the Norm. TL;DR: Submitting work that doesn't pass **norminette**'s check makes no sense.
- These exercises are arranged in order of difficulty, from easiest to hardest. We **will not** consider a successfully completed harder exercise if an easier one is not fully functional.
- Using a forbidden function is considered cheating. Cheaters receive a grade of **-42**, which is non-negotiable.
- You only need to submit a **main()** function if we specifically ask for a **program**.
- **Moulinette** compiles with the following flags: **-Wall -Wextra -Werror**, using **cc**.
- If your program does not compile, you will receive a grade of **0**.
- You **cannot** leave **any** additional file in your directory beyond those specified in the assignment.
- Have a question? Ask the peer on your right. If not, try the peer on your left.

- Your reference guide is called **Google / man / the Internet / ...**
- Check the "C Piscine" section of the forum on the intranet or the Piscine on Slack.
- Carefully examine the examples. They may contain crucial details that are not explicitly stated in the assignment...
- By Odin, by Thor! Use your brain!!!
- 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 btree_create_node, so make arrangements (it could be useful to have its prototype in a file ft_btree.h...).

Chapter II

AI Instructions

Context

The C Piscine is intense. It's your first big challenge at 42 — a deep dive into problem-solving, autonomy, and community.

During this phase, your main objective is to build your foundation — through struggle, repetition, and especially **peer-learning** exchange.

In the AI era, shortcuts are easy to find. However, it's important to consider whether your AI usage is truly helping you grow — or simply getting in the way of developing real skills.

The Piscine is also a human experience — and for now, nothing can replace that. Not even AI.

For a more complete overview of our stance on AI — as a learning tool, as part of the ICT curriculum, and as a growing expectation in the job market — please refer to the dedicated FAQ available on the intranet.

Main message

- Build strong foundations without shortcuts.
- Really develop tech & power skills.
- Experience real peer-learning, start learning how to learn and solve new problems.
- The learning journey is more important than the result.
- Learn about the risks associated with AI, and develop effective control practices and countermeasures to avoid common pitfalls.

Learner rules:

- You should apply reasoning to your assigned tasks, especially before turning to AI.
- You should not ask for direct answers to the AI.
- You should learn about 42 global approach on AI.

Phase outcomes:

Within this foundational phase, you will get the following outcomes:

- Get proper tech and coding foundations.
- Know why and how AI can be dangerous during this phase.

Comments and example:

- Yes, we know AI exists — and yes, it can solve your projects. But you're here to learn, not to prove that AI has learned. Don't waste your time (or ours) just to demonstrate that AI can solve the given problem.
- Learning at 42 isn't about knowing the answer — it's about developing the ability to find one. AI gives you the answer directly, but that prevents you from building your own reasoning. And reasoning takes time, effort, and involves failure. The path to success is not supposed to be easy.
- Keep in mind that during exams, AI is not available — no internet, no smartphones, etc. You'll quickly realise if you've relied too heavily on AI in your learning process.
- Peer learning exposes you to different ideas and approaches, improving your interpersonal skills and your ability to think divergently. That's far more valuable than just chatting with a bot. So don't be shy — talk, ask questions, and learn together!
- Yes, AI will be part of the curriculum — both as a learning tool and as a topic in itself. You'll even have the chance to build your own AI software. In order to learn more about our crescendo approach you'll go through in the documentation available on the intranet.

✓ Good practice:

I'm stuck on a new concept. I ask someone nearby how they approached it. We talk for 10 minutes — and suddenly it clicks. I get it.

✗ Bad practice:

I secretly use AI, copy some code that looks right. During peer evaluation, I can't explain anything. I fail. During the exam — no AI — I'm stuck again. I fail.

Chapter III

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 IV

Exercise 00 : btree_create_node

	Exercise 00
btree_create_node	
Turn-in directory: *ex*00/	
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 value of the argument, and all other elements to 0.
- The address of the created node is returned.
- Here's how it should be prototyped :

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

Chapter V

Exercise 01 : btree_apply_prefix

	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 an argument to the `item` of each node, using `prefix traversal` to traverse the tree.
- Here's how it should be prototyped :

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

Chapter VI

Exercise 02 : btree_apply_infix

	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 an argument to the `item` of each node, using `infix traversal` to traverse the tree.
- Here's how it should be prototyped :

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

Chapter VII

Exercise 03 : btree_apply_suffix

	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 an argument to the `item` of each node, using `suffix traversal` to traverse the tree.
- Here's how it should be prototyped :

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

Chapter VIII

Exercise 04 : btree_insert_data

	Exercise 04
btree_insert_data	
Turn-in directory: *ex*04/	
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 are located on the right. We'll also pass a comparison function similar to `strcmp` as an argument.
- The `root` parameter points to the root node of the tree. When called for the first time, 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 IX

Exercise 05 : btree_search_item

	Exercise 05
btree_search_item	
Turn-in directory: *ex*05/	
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 an argument. The tree should be traversed 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 X

Exercise 06 : btree_level_count

	Exercise 06
btree_level_count	
Turn-in directory: *ex*06/	
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 an argument.
- Here's how it should be prototyped :

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

Chapter XI

Exercise 07 : btree_apply_by_level

	Exercise 07
btree_apply_by_level	
Turn-in directory: *ex*07/	
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 an 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 *`, corresponds to the node's item ;
  - The second argument, of type `int`, corresponds to the level at which we find it: 0 for the root, 1 for children, 2 for grand-children, etc. ;
  - The third argument, of type `int`, is worth 1 if it is the first `node` of the level, or 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 XII

Submission and peer-evaluation

Submit your assignment to your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Make sure to double-check the filenames to ensure they are correct.

`You must submit only the files specified in the project instructions.`