Version: 1.0

Selection

C-Binary-Trees

Summary

This project focuses on implementing and manipulating binary trees in C.

#C #DataStructures #BinaryTrees

Intellectual Property Disclaimer

All content presented in this training module, including but not limited to texts, images, graphics, and other materials, is protected by intellectual property rights held by Association 42.

Terms of Use:

- **Personal use:** You are permitted to use the contents of this module solely for personal purpose. Any commercial use, reproduction, distribution, modification, or public display is strictly prohibited without prior written permission from Association 42.
- **Respect for Integrity:** You must not alter, transform, or adapt the content in any way that could harm its integrity.

Protection of Rights:

Any violation of these terms constitutes an infringement of intellectual property rights and may result in legal action. We reserve the right to take all necessary measures to protect our rights, including but not limited to claims for damages.

For any questions regarding the use of the content or to obtain authorization, please contact: `legal@42.fr`

Contents

Chapter 1

Instructions

- Only this page will serve as a reference: do not trust rumors.
- Watch out! This document could potentially change up until 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.
- Additionally, 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, to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try to understand your code if it doesn't adhere to the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be foolish to submit 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` consider 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 `cc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional files in your directory other than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ...`.
- Check out 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!!!

Do not forget to add the *standard 42 header* in each of your .c/.h files. Norminette checks its existence anyway!

Norminette must be launched with the *-R CheckForbiddenSourceHeader* flag. Moulinette will use it too.

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 2

Foreword

Congratulations!

You've unlocked a page full of fascinating facts and trivia. Enjoy the journey through some of the world's most intriguing tidbits!

- **The Eiffel Tower:** The Eiffel Tower can be 15 cm taller during the summer due to the expansion of the iron on hot days.
- **Honey Bees:** Honey bees have hair on their eyes. It helps them collect pollen.
- **Octopuses:** Octopuses have three hearts. Two pump blood to the gills, while the third pumps it to the rest of the body.
- **Space:** A day on Venus is longer than a year on Venus. It takes about 243 Earth days to rotate once on its axis.
- **Bananas:** Bananas are slightly radioactive. They contain a small amount of potassium-40, a radioactive isotope.
- **Human Body:** The shortest war in history lasted only 38 minutes between Britain and Zanzibar in 1896.
- **Cleopatra:** Cleopatra was not actually Egyptian. She was of Macedonian Greek origin and was the first in her family to learn the Egyptian language.
- **Dogs:** Dogs can understand up to 250 words and gestures, can count up to five, and can perform simple mathematical calculations.
- **Trees:** Trees can communicate with each other through a network of soil fungi, known as the "wood wide web."
- **Ants:** Ants can lift and carry more than 10 times their body weight.

Celebrate your curiosity and keep exploring the wonders of our world!

Chapter 3

Information for this project

For the following exercises, you have to use the following structure :

ft_btree.h

```
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_breee.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 4

Exercise 0: btree_create_node

	Exercise: 0	
btree_create_node		
Directory: *ex0/*		
Files to Submit: `btree_create_node.c, ft_btree.h`		
Authorized: `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.

Prototype:

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

Chapter 5

Exercise 1: btree_apply_prefix

	Exercise: 1	
	btree_apply_prefix	
Directory: *ex1/*		
Files to Submit: `btree_apply_prefix.c, ft_btree.h`		
Authorized: `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.

Prototype:

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

Chapter 6

Exercise 2: btree_apply_infix

	Exercise: 2	
	btree_apply_infix	
Directory: *ex2/*		
Files to Submit: `btree_apply_infix.c, ft_btree.h`		
Authorized: `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.

Prototype:

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

Chapter 7

Exercise 3: btree_apply_suffix

	Exercise: 3	
	btree_apply_suffix	
Directory: *ex3/*		
Files to Submit: `btree_apply_suffix.c, ft_btree.h`		
Authorized: `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.

Prototype:

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

Chapter 8

Exercise 4: btree_insert_data

	Exercise: 4	
	btree_insert_data	
Directory: *ex4/*		
Files to Submit: `btree_insert_data.c, ft_btree.h`		
Authorized: `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`.

Prototype:

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

Chapter 9

Exercise 5: btree_search_item

	Exercise: 5	
	btree_search_item	
	Directory: *ex5/*	
	Files to Submit: `btree_search_item.c, ft_btree.h`	
	Authorized: `None`	

- Create a function `btree_search_item` which returns the first element matching 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`.

Prototype:

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

Chapter 10

Exercise 6: btree_level_count

	Exercise: 6	
	btree_level_count	
Directory: *ex6/*		
Files to Submit: `btree_level_count.c, ft_btree.h`		
Authorized: `None`		

- Create a function `btree_level_count` which returns the size of the largest branch of the tree passed as an argument.

Prototype:

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

Chapter 11

Exercise 7: btree_apply_by_level

	Exercise: 7	
btree_apply_by_level		
Directory: *ex7/*		
Files to Submit: `btree_apply_by_level.c, ft_btree.h`		
Authorized: `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 traversed 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`, represents the level of the node: 0 for the root, 1 for children, 2 for grandchildren, etc.
- The third argument, of type `int`, is set to 1 if it's the first node of the level, or 0 otherwise.

Prototype:

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

Chapter 12

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

You need to return only the files requested by the subject of this project.