(/)

Curriculum
SE Foundations
Average: **108.76%**

0x1D. C - Binary trees

C | Group project | Algorithm | Data structure

By: Alexandre Gautier

Weight: 5

Project to be done in teams of 2 people (your team: Baraka Ngaira, Abdia Elema Gababo)

Project over - took place from Jul 31, 2023 6:00 AM to Aug 4, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Contribution:** 100.0%
- **Auto QA review:** 157.0/157 mandatory & 46.0/196 optional
- **Altogether: 123.47%**
  - Mandatory: 100.0%
  - Optional: 23.47%
  - Contribution: 100.0%
  - Calculation: 100.0% * (100.0% + (100.0% * 23.47%)) == **123.47%**

Resources

Read or watch:

- Binary tree (/rltoken/1F2x42-8vUbOmU4L1C1KMg) (*note the first line:* `Not to be confused with B-tree.`)
- Data Structure and Algorithms - Tree (/rltoken/QmcTMCkQyrgMjrqoWxYdhw)
- Tree Traversal (/rltoken/z6ZaXr_RxwE5nTHAUx_dfQ)
- Binary Search Tree (/rltoken/qO5dBlMnYJzbaWG3xVpcnQ)
- Data structures: Binary Tree (/rltoken/BeyJ2gjlE7_djwRiDyeHig)

Help

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/rDjGcLNoVZsZG1Br0UbX6A), **without the help of Google**:

General

- What is a binary tree
- What is the difference between a binary tree and a Binary Search Tree
- What is the possible gain in terms of time complexity compared to linked lists
- What are the depth, the height, the size of a binary tree
- What are the different traversal methods to go through a binary tree
- What is a complete, a full, a perfect, a balanced binary tree

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi, vim, emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using gcc, using the options -Wall -Werror -Wextra -pedantic -std=gnu89
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/alx-tools/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/alx-tools/Betty/blob/master/betty-doc.pl)
- You are not allowed to use global variables
- No more than 5 functions per file
- You are allowed to use the standard library
- In the following examples, the `main.c` files are shown as examples. You can use them to test your functions, but you don't have to push them to your repo (if you do we won't take them into account). We will use our own `main.c` files at compilation. Our `main.c` files might be different from the one shown in the examples
- The prototypes of all your functions should be included in your header file called `binary_trees.h`
- Don't forget to push your header file
- All your header files should be include guarded

GitHub

There should be one project repository per group. If you clone/fork/whatever a project repository with the same name before the second deadline, you risk a 0% score.

More Info

Data structures

Please use the following data structures and types for binary trees. Don't forget to include them in your header file.

Basic Binary Tree

```
/**
 * struct binary_tree_s - Binary tree node
 *
 * @n: Integer stored in the node
 * @parent: Pointer to the parent node
 * @left: Pointer to the left child node
 * @right: Pointer to the right child node
 */
struct binary_tree_s
{
    int n;
    struct binary_tree_s *parent;
    struct binary_tree_s *left;
    struct binary_tree_s *right;
};

typedef struct binary_tree_s binary_tree_t;
```

Binary Search Tree

```
typedef struct binary_tree_s bst_t;
```

AVL Tree

```
typedef struct binary_tree_s avl_t;
```

Max Binary Heap

```
typedef struct binary_tree_s heap_t;
```

Note: For tasks 0 to 23 (included), you have to deal with simple binary trees. They are not BSTs, thus they don't follow any kind of rule.

Print function

To match the examples in the tasks, you are given this function (https://github.com/alx-tools/0x1C.c)

This function is used only for visualization purposes. You don't have to push it to your repo. It may not be used during the correction

(/)

Tasks

0. New node

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that creates a binary tree node

- Prototype: `binary_tree_t *binary_tree_node(binary_tree_t *parent, int value);`
- Where `parent` is a pointer to the parent node of the node to create
- And `value` is the value to put in the new node
- When created, a node does not have any child
- Your function must return a pointer to the new node, or `NULL` on failure

```
alex@/tmp/binary_trees$ cat 0-main.c
#include <stdlib.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);

    root->left = binary_tree_node(root, 12);
    root->left->left = binary_tree_node(root->left, 6);
    root->left->right = binary_tree_node(root->left, 16);

    root->right = binary_tree_node(root, 402);
    root->right->left = binary_tree_node(root->right, 256);
    root->right->right = binary_tree_node(root->right, 512);

    binary_tree_print(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 0-ma
in.c 0-binary_tree_node.c -o 0-node
alex@/tmp/binary_trees$ ./0-node
       .-------(098)-------.
  .--(012)--.         .--(402)--.
(006)     (016)     (256)     (512)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `0-binary_tree_node.c`

Done! | Help | Check your code | Get a sandbox | QA Review

1. Insert left

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that inserts a node as the left-child of another node

- Prototype: `binary_tree_t *binary_tree_insert_left(binary_tree_t *parent, int value);`

- Where `parent` is a pointer to the node to insert the left-child in
- And `value` is the value to store in the new node
- Your function must return a pointer to the created node, or `NULL` on failure or if `parent` is `NULL`
- If `parent` already has a left-child, the new node must take its place, and the old left-child must be set as the left-child of the new node.

```
alex@/tmp/binary_trees$ cat 1-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_print(root);
    printf("\n");
    binary_tree_insert_left(root->right, 128);
    binary_tree_insert_left(root, 54);
    binary_tree_print(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 1-ma
in.c 1-binary_tree_insert_left.c 0-binary_tree_node.c -o 1-left
alex@/tmp/binary_trees$ ./1-left
  .--(098)--.
(012)     (402)

       .--(098)-------.
  .--(054)       .--(402)
(012)          (128)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `1-binary_tree_insert_left.c`

Done! | Help | Check your code | Get a sandbox | QA Review

2. Insert right

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that inserts a node as the right-child of another node

- Prototype: `binary_tree_t *binary_tree_insert_right(binary_tree_t *parent, int value);`
- Where `parent` is a pointer to the node to insert the right-child in
- And `value` is the value to store in the new node
- Your function must return a pointer to the created node, or `NULL` on failure or if `parent` is `NULL`
- If `parent` already has a right-child, the new node must take its place, and the old right-child must be set as the right-child of the new node.

```
alex@/tmp/binary_trees$ cat 2-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_print(root);
    printf("\n");
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 2-ma
in.c 2-binary_tree_insert_right.c 0-binary_tree_node.c -o 2-right
alex@/tmp/binary_trees$ ./2-right
  .--(098)--.
(012)     (402)

  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `2-binary_tree_insert_right.c`

Done! | Help | Check your code | Get a sandbox | QA Review

3. Delete

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that deletes an entire binary tree

- Prototype: `void binary_tree_delete(binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to delete
- If `tree` is `NULL`, do nothing

```
alex@/tmp/binary_trees$ cat 3-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);
    binary_tree_delete(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 3-ma
in.c 3-binary_tree_delete.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 3-d
el
alex@/tmp/binary_trees$ valgrind ./3-del
==13264== Memcheck, a memory error detector
==13264== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==13264== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==13264== Command: ./3-del
==13264==
  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
==13264==
==13264== HEAP SUMMARY:
==13264==     in use at exit: 0 bytes in 0 blocks
==13264==   total heap usage: 9 allocs, 9 frees, 949 bytes allocated
==13264==
==13264== All heap blocks were freed -- no leaks are possible
==13264==
==13264== For counts of detected and suppressed errors, rerun with: -v
==13264== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`

- File: `3-binary_tree_delete.c`

(/)

Done! | Help | Check your code | Get a sandbox | QA Review

4. Is leaf

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that checks if a node is a leaf

- Prototype: `int binary_tree_is_leaf(const binary_tree_t *node);`
- Where `node` is a pointer to the node to check
- Your function must return `1` if `node` is a leaf, otherwise `0`
- If `node` is `NULL`, return `0`

(/)

```
alex@/tmp/binary_trees$ cat 4-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int ret;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);

    ret = binary_tree_is_leaf(root);
    printf("Is %d a leaf: %d\n", root->n, ret);
    ret = binary_tree_is_leaf(root->right);
    printf("Is %d a leaf: %d\n", root->right->n, ret);
    ret = binary_tree_is_leaf(root->right->right);
    printf("Is %d a leaf: %d\n", root->right->right->n, ret);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 4-bi
nary_tree_is_leaf.c 4-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 4-
leaf
alex@/tmp/binary_trees$ ./4-leaf
  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
Is 98 a leaf: 0
Is 128 a leaf: 0
Is 402 a leaf: 1
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `4-binary_tree_is_leaf.c`

Done! | Help | Check your code | Get a sandbox | QA Review

5. Is root

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that checks if a given node is a root

- Prototype: `int binary_tree_is_root(const binary_tree_t *node);`
- Where `node` is a pointer to the node to check
- Your function must return `1` if `node` is a root, otherwise `0`
- If `node` is `NULL`, return `0`

```
alex@/tmp/binary_trees$ cat 5-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int ret;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);

    ret = binary_tree_is_root(root);
    printf("Is %d a root: %d\n", root->n, ret);
    ret = binary_tree_is_root(root->right);
    printf("Is %d a root: %d\n", root->right->n, ret);
    ret = binary_tree_is_root(root->right->right);
    printf("Is %d a root: %d\n", root->right->right->n, ret);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 5-bi
nary_tree_is_root.c 5-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 5-
root
alex@/tmp/binary_trees$ ./5-root
  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
Is 98 a root: 1
Is 128 a root: 0
Is 402 a root: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `5-binary_tree_is_root.c`

Done! | Help | Check your code | Get a sandbox | QA Review

6. Pre-order traversal

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that goes through a binary tree using pre-order traversal

- Prototype: `void binary_tree_preorder(const binary_tree_t *tree, void (*func)(int));`
- Where `tree` is a pointer to the root node of the tree to traverse
- And `func` is a pointer to a function to call for each node. The value in the node must be passed as a parameter to this function.
- If `tree` or `func` is `NULL`, do nothing

```
alex@/tmp/binary_trees$ cat 6-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * print_num - Prints a number
 *
 * @n: Number to be printed
 */
void print_num(int n)
{
    printf("%d\n", n);
}

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    root->left->left = binary_tree_node(root->left, 6);
    root->left->right = binary_tree_node(root->left, 56);
    root->right->left = binary_tree_node(root->right, 256);
    root->right->right = binary_tree_node(root->right, 512);

    binary_tree_print(root);
    binary_tree_preorder(root, &print_num);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 6-ma
in.c 6-binary_tree_preorder.c 0-binary_tree_node.c -o 6-pre
alex@/tmp/binary_trees$ ./6-pre
       .-------(098)-------.
  .--(012)--.         .--(402)--.
(006)     (056)     (256)     (512)
98
12
6
56
402
256
512
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `6-binary_tree_preorder.c`

Done! | Help | Check your code | Get a sandbox | QA Review

7. In-order traversal

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that goes through a binary tree using in-order traversal

- Prototype: `void binary_tree_inorder(const binary_tree_t *tree, void (*func)(int));`
- Where `tree` is a pointer to the root node of the tree to traverse
- And `func` is a pointer to a function to call for each node. The value in the node must be passed as a parameter to this function.
- If `tree` or `func` is `NULL`, do nothing

```
alex@/tmp/binary_trees$ cat 7-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * print_num - Prints a number
 *
 * @n: Number to be printed
 */
void print_num(int n)
{
    printf("%d\n", n);
}

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    root->left->left = binary_tree_node(root->left, 6);
    root->left->right = binary_tree_node(root->left, 56);
    root->right->left = binary_tree_node(root->right, 256);
    root->right->right = binary_tree_node(root->right, 512);

    binary_tree_print(root);
    binary_tree_inorder(root, &print_num);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 7-ma
in.c 7-binary_tree_inorder.c 0-binary_tree_node.c -o 7-in
alex@/tmp/binary_trees$ ./7-in
       .-------(098)-------.
  .--(012)--.         .--(402)--.
(006)     (056)     (256)     (512)
6
12
56
98
256
402
512
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `7-binary_tree_inorder.c`

Done! | Help | Check your code | Get a sandbox | QA Review

8. Post-order traversal

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that goes through a binary tree using post-order traversal

- Prototype: `void binary_tree_postorder(const binary_tree_t *tree, void (*func)(int));`
- Where `tree` is a pointer to the root node of the tree to traverse
- And `func` is a pointer to a function to call for each node. The value in the node must be passed as a parameter to this function.
- If `tree` or `func` is `NULL`, do nothing

```
alex@/tmp/binary_trees$ cat 8-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * print_num - Prints a number
 *
 * @n: Number to be printed
 */
void print_num(int n)
{
    printf("%d\n", n);
}

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    root->left->left = binary_tree_node(root->left, 6);
    root->left->right = binary_tree_node(root->left, 56);
    root->right->left = binary_tree_node(root->right, 256);
    root->right->right = binary_tree_node(root->right, 512);

    binary_tree_print(root);
    binary_tree_postorder(root, &print_num);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 8-ma
in.c 8-binary_tree_postorder.c 0-binary_tree_node.c -o 8-post
alex@/tmp/binary_trees$ ./8-post
       .-------(098)-------.
  .--(012)--.         .--(402)--.
(006)     (056)     (256)     (512)
6
56
12
256
512
402
98
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `8-binary_tree_postorder.c`

Done! | Help | Check your code | Get a sandbox | QA Review

9. Height

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that measures the height of a binary tree

- Prototype: `size_t binary_tree_height(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to measure the height.
- If `tree` is `NULL`, your function must return `0`

```
alex@/tmp/binary_trees$ cat 9-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    size_t height;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);

    height = binary_tree_height(root);
    printf("Height from %d: %lu\n", root->n, height);
    height = binary_tree_height(root->right);
    printf("Height from %d: %lu\n", root->right->n, height);
    height = binary_tree_height(root->left->right);
    printf("Height from %d: %lu\n", root->left->right->n, height);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 9-bi
nary_tree_height.c 9-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 9-h
eight
alex@/tmp/binary_trees$ ./9-height
  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
Height from 98: 2
Height from 128: 1
Height from 54: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `9-binary_tree_height.c`

Done! | Help | Check your code | Get a sandbox | QA Review

10. Depth (7)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that measures the depth of a node in a binary tree

- Prototype: `size_t binary_tree_depth(const binary_tree_t *tree);`
- Where `tree` is a pointer to the node to measure the depth
- If `tree` is `NULL`, your function must return `0`

```
alex@/tmp/binary_trees$ cat 10-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    size_t depth;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);

    depth = binary_tree_depth(root);
    printf("Depth of %d: %lu\n", root->n, depth);
    depth = binary_tree_depth(root->right);
    printf("Depth of %d: %lu\n", root->right->n, depth);
    depth = binary_tree_depth(root->left->right);
    printf("Depth of %d: %lu\n", root->left->right->n, depth);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 10-b
inary_tree_depth.c 10-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 10
-depth
alex@/tmp/binary_trees$ ./10-depth
  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
Depth of 98: 0
Depth of 128: 1
Depth of 54: 2
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `10-binary_tree_depth.c`

Done! | Help | Check your code | Get a sandbox | QA Review

11. Size (7)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that measures the size of a binary tree

- Prototype: `size_t binary_tree_size(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to measure the size
- If `tree` is `NULL`, the function must return 0

(/)

```
alex@/tmp/binary_trees$ cat 11-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    size_t size;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);

    size = binary_tree_size(root);
    printf("Size of %d: %lu\n", root->n, size);
    size = binary_tree_size(root->right);
    printf("Size of %d: %lu\n", root->right->n, size);
    size = binary_tree_size(root->left->right);
    printf("Size of %d: %lu\n", root->left->right->n, size);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 11-b
inary_tree_size.c 11-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 11-
size
alex@/tmp/binary_trees$ ./11-size
  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
Size of 98: 5
Size of 128: 2
Size of 54: 1
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `11-binary_tree_size.c`

☑ Done! | Help | Check your code | >_ Get a sandbox | QA Review

12. Leaves (7)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that counts the leaves in a binary tree

- Prototype: `size_t binary_tree_leaves(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to count the number of leaves
- If `tree` is `NULL`, the function must return 0
- A `NULL` pointer is not a leaf

(/)

```
alex@/tmp/binary_trees$ cat 12-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    size_t leaves;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);

    leaves = binary_tree_leaves(root);
    printf("Leaves in %d: %lu\n", root->n, leaves);
    leaves = binary_tree_leaves(root->right);
    printf("Leaves in %d: %lu\n", root->right->n, leaves);
    leaves = binary_tree_leaves(root->left->right);
    printf("Leaves in %d: %lu\n", root->left->right->n, leaves);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 12-b
inary_tree_leaves.c 12-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 1
2-leaves
alex@/tmp/binary_trees$ ./12-leaves
  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
Leaves in 98: 2
Leaves in 128: 1
Leaves in 54: 1
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `12-binary_tree_leaves.c`

Done! | Help | Check your code | Get a sandbox | QA Review

13. Nodes (7)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that counts the nodes with at least 1 child in a binary tree

- Prototype: `size_t binary_tree_nodes(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to count the number of nodes
- If `tree` is `NULL`, the function must return 0
- A `NULL` pointer is not a node

```
alex@/tmp/binary_trees$ cat 13-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    size_t nodes;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_print(root);

    nodes = binary_tree_nodes(root);
    printf("Nodes in %d: %lu\n", root->n, nodes);
    nodes = binary_tree_nodes(root->right);
    printf("Nodes in %d: %lu\n", root->right->n, nodes);
    nodes = binary_tree_nodes(root->left->right);
    printf("Nodes in %d: %lu\n", root->left->right->n, nodes);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 13-b
inary_tree_nodes.c 13-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 13
-nodes
alex@/tmp/binary_trees$ ./13-nodes
  .-------(098)--.
(012)--.       (128)--.
     (054)          (402)
Nodes in 98: 3
Nodes in 128: 1
Nodes in 54: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `13-binary_tree_nodes.c`

Done! | Help | Check your code | Get a sandbox | QA Review

14. Balance factor (7)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that measures the balance factor of a binary tree

- Prototype: `int binary_tree_balance(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to measure the balance factor
- If `tree` is `NULL`, return `0`

```
alex@/tmp/binary_trees$ cat 14-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int balance;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    binary_tree_insert_left(root, 45);
    binary_tree_insert_right(root->left, 50);
    binary_tree_insert_left(root->left->left, 10);
    binary_tree_insert_left(root->left->left->left, 8);
    binary_tree_print(root);

    balance = binary_tree_balance(root);
    printf("Balance of %d: %+d\n", root->n, balance);
    balance = binary_tree_balance(root->right);
    printf("Balance of %d: %+d\n", root->right->n, balance);
    balance = binary_tree_balance(root->left->left->right);
    printf("Balance of %d: %+d\n", root->left->left->right->n, balance);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 14-b
inary_tree_balance.c 14-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c 1-b
inary_tree_insert_left.c -o 14-balance
alex@/tmp/binary_trees$ ./14-balance
                      .-------(098)--.
            .-------(045)--.       (128)--.
       .--(012)--.       (050)          (402)
  .--(010)     (054)
(008)
Balance of 98: +2
Balance of 128: -1
Balance of 54: +0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `14-binary_tree_balance.c`

(/)

Done! | Help | Check your code | Get a sandbox | QA Review

15. Is full

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that checks if a binary tree is full

- Prototype: `int binary_tree_is_full(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to check
- If `tree` is `NULL`, your function must return `0`

```
alex@/tmp/binary_trees$ cat 15-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int full;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    root->left->left = binary_tree_node(root->left, 10);
    binary_tree_print(root);

    full = binary_tree_is_full(root);
    printf("Is %d full: %d\n", root->n, full);
    full = binary_tree_is_full(root->left);
    printf("Is %d full: %d\n", root->left->n, full);
    full = binary_tree_is_full(root->right);
    printf("Is %d full: %d\n", root->right->n, full);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 15-b
inary_tree_is_full.c 15-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o
15-full
alex@/tmp/binary_trees$ ./15-full
       .-------(098)--.
  .--(012)--.       (128)--.
(010)     (054)          (402)
Is 98 full: 0
Is 12 full: 1
Is 128 full: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `15-binary_tree_is_full.c`

Done! | Help | Check your code | Get a sandbox | QA Review

16. Is perfect

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that checks if a binary tree is perfect

- Prototype: `int binary_tree_is_perfect(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to check
- If `tree` is `NULL`, your function must return `0`

(/)

```
alex@/tmp/binary_trees$ cat 16-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int perfect;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    binary_tree_insert_right(root->left, 54);
    binary_tree_insert_right(root, 128);
    root->left->left = binary_tree_node(root->left, 10);
    root->right->left = binary_tree_node(root->right, 10);

    binary_tree_print(root);
    perfect = binary_tree_is_perfect(root);
    printf("Perfect: %d\n\n", perfect);

    root->right->right->left = binary_tree_node(root->right->right, 10);
    binary_tree_print(root);
    perfect = binary_tree_is_perfect(root);
    printf("Perfect: %d\n\n", perfect);

    root->right->right->right = binary_tree_node(root->right->right, 10);
    binary_tree_print(root);
    perfect = binary_tree_is_perfect(root);
    printf("Perfect: %d\n", perfect);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 16-b
inary_tree_is_perfect.c 16-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c
-o 16-perfect
alex@/tmp/binary_trees$ ./16-perfect
       .-------(098)-------.
  .--(012)--.         .--(128)--.
(010)     (054)     (010)     (402)
Perfect: 1

       .-------(098)-------.
  .--(012)--.         .--(128)-------.
(010)     (054)     (010)       .--(402)
                              (010)
Perfect: 0
```

```
(/)      .-------(098)-------.
  .--(012)--.          .--(128)-------.
(010)     (054)     (010)       .--(402)--.
                               (010)     (010)
Perfect: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `16-binary_tree_is_perfect.c`

Done! | Help | Check your code | Get a sandbox | QA Review

17. Sibling

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that finds the sibling of a node

- Prototype: `binary_tree_t *binary_tree_sibling(binary_tree_t *node);`
- Where `node` is a pointer to the node to find the sibling
- Your function must return a pointer to the sibling node
- If `node` is `NULL` or the parent is `NULL`, return `NULL`
- If `node` has no sibling, return `NULL`

```
alex@/tmp/binary_trees$ cat 17-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    binary_tree_t *sibling;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 128);
    root->left->right = binary_tree_node(root->left, 54);
    root->right->right = binary_tree_node(root->right, 402);
    root->left->left = binary_tree_node(root->left, 10);
    root->right->left = binary_tree_node(root->right, 110);
    root->right->right->left = binary_tree_node(root->right->right, 200);
    root->right->right->right = binary_tree_node(root->right->right, 512);

    binary_tree_print(root);
    sibling = binary_tree_sibling(root->left);
    printf("Sibling of %d: %d\n", root->left->n, sibling->n);
    sibling = binary_tree_sibling(root->right->left);
    printf("Sibling of %d: %d\n", root->right->left->n, sibling->n);
    sibling = binary_tree_sibling(root->left->right);
    printf("Sibling of %d: %d\n", root->left->right->n, sibling->n);
    sibling = binary_tree_sibling(root);
    printf("Sibling of %d: %p\n", root->n, (void *)sibling);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 17-m
ain.c 17-binary_tree_sibling.c 0-binary_tree_node.c -o 17-sibling
alex@/tmp/binary_trees$ ./17-sibling
       .-------(098)-------.
  .--(012)--.         .--(128)-------.
(010)     (054)     (110)       .--(402)--.
                              (200)     (512)
Sibling of 12: 128
Sibling of 110: 402
Sibling of 54: 10
Sibling of 98: (nil)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `17-binary_tree_sibling.c`

(/)

Done! | Help | Check your code | Get a sandbox | QA Review

18. Uncle

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that finds the uncle of a node

- Prototype: `binary_tree_t *binary_tree_uncle(binary_tree_t *node);`
- Where `node` is a pointer to the node to find the uncle
- Your function must return a pointer to the uncle node
- If `node` is `NULL`, return `NULL`
- If `node` has no uncle, return `NULL`

(/)

```
alex@/tmp/binary_trees$ cat 18-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    binary_tree_t *uncle;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 128);
    root->left->right = binary_tree_node(root->left, 54);
    root->right->right = binary_tree_node(root->right, 402);
    root->left->left = binary_tree_node(root->left, 10);
    root->right->left = binary_tree_node(root->right, 110);
    root->right->right->left = binary_tree_node(root->right->right, 200);
    root->right->right->right = binary_tree_node(root->right->right, 512);

    binary_tree_print(root);
    uncle = binary_tree_uncle(root->right->left);
    printf("Uncle of %d: %d\n", root->right->left->n, uncle->n);
    uncle = binary_tree_uncle(root->left->right);
    printf("Uncle of %d: %d\n", root->left->right->n, uncle->n);
    uncle = binary_tree_uncle(root->left);
    printf("Uncle of %d: %p\n", root->left->n, (void *)uncle);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 18-m
ain.c 18-binary_tree_uncle.c 0-binary_tree_node.c -o 18-uncle
alex@/tmp/binary_trees$ ./18-uncle
       .-------(098)-------.
  .--(012)--.         .--(128)-------.
(010)     (054)     (110)       .--(402)--.
                              (200)     (512)
Uncle of 110: 12
Uncle of 54: 128
Uncle of 12: (nil)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`

- File: `18-binary_tree_uncle.c`

(/)

Done! | Help | Check your code | Get a sandbox | QA Review

19. Lowest common ancestor

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that finds the lowest common ancestor of two nodes

- Prototype: `binary_tree_t *binary_trees_ancestor(const binary_tree_t *first, const binary_tree_t *second);`
- Where `first` is a pointer to the first node
- And `second` is a pointer to the second node
- Your function must return a pointer to the lowest common ancestor node of the two given nodes
- If no common ancestor was found, your function must return `NULL`

```
alex@/tmp/binary_trees$ cat 100-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * launch_test - Test ancestor function and print informations
 *
 * @n1: First node
 * @n2: Second node
 */
void launch_test(binary_tree_t *n1, binary_tree_t *n2)
{
    binary_tree_t *ancestor;

    ancestor = binary_trees_ancestor(n1, n2);
    printf("Ancestor of [%d] & [%d]: ", n1->n, n2->n);
    if (!ancestor)
        printf("(nil)\n");
    else
        printf("%d\n", ancestor->n);
}

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    root->left->right = binary_tree_node(root->left, 54);
    root->right->right = binary_tree_node(root->right, 128);
    root->left->left = binary_tree_node(root->left, 10);
    root->right->left = binary_tree_node(root->right, 45);
    root->right->right->left = binary_tree_node(root->right->right, 92);
    root->right->right->right = binary_tree_node(root->right->right, 65);
    binary_tree_print(root);

    launch_test(root->left, root->right);
    launch_test(root->right->left, root->right->right->right);
    launch_test(root->right->right, root->right->right->right);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 100-
main.c 100-binary_trees_ancestor.c 0-binary_tree_node.c -o 100-ancestor
alex@/tmp/binary_trees$ ./100-ancestor
       .-------(098)-------.
```

```
  .--(012)--.         .--(402)-------.
(010)     (054)     (045)       .--(128)--.
                                (092)     (065)
Ancestor of [12] & [402]: 98
Ancestor of [45] & [65]: 402
Ancestor of [128] & [65]: 128
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `100-binary_trees_ancestor.c`

Done! | Help | Check your code | Get a sandbox | QA Review

20. Level-order traversal

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that goes through a binary tree using level-order traversal

- Prototype: `void binary_tree_levelorder(const binary_tree_t *tree, void (*func)(int));`
- Where `tree` is a pointer to the root node of the tree to traverse
- And `func` is a pointer to a function to call for each node. The value in the node must be passed as a parameter to this function.
- If `tree` or `func` is `NULL`, do nothing

```
alex@/tmp/binary_trees$ cat 101-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * print_num - Prints a number
 *
 * @n: Number to be printed
 */
void print_num(int n)
{
    printf("%d\n", n);
}

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 402);
    root->left->left = binary_tree_node(root->left, 6);
    root->left->right = binary_tree_node(root->left, 56);
    root->right->left = binary_tree_node(root->right, 256);
    root->right->right = binary_tree_node(root->right, 512);

    binary_tree_print(root);
    binary_tree_levelorder(root, &print_num);
    binary_tree_delete(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 101-
main.c 101-binary_tree_levelorder.c 0-binary_tree_node.c 3-binary_tree_delete.c -o 1
01-lvl
alex@/tmp/binary_trees$ valgrind ./101-lvl
==23445== Memcheck, a memory error detector
==23445== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==23445== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==23445== Command: ./101-lvl
==23445==
       .-------(098)-------.
  .--(012)--.         .--(402)--.
(006)     (056)     (256)     (512)
98
12
402
```

```
6
(/)
(6)
256
512
==23445==
==23445== HEAP SUMMARY:
==23445==     in use at exit: 0 bytes in 0 blocks
==23445==   total heap usage: 19 allocs, 19 frees, 1,197 bytes allocated
==23445==
==23445== All heap blocks were freed -- no leaks are possible
==23445==
==23445== For counts of detected and suppressed errors, rerun with: -v
==23445== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `101-binary_tree_levelorder.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

21. Is complete

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that checks if a binary tree is complete

- Prototype: `int binary_tree_is_complete(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to check
- If `tree` is `NULL`, your function must return `0`

```
alex@/tmp/binary_trees$ cat 102-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int complete;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 128);
    root->left->right = binary_tree_node(root->left, 54);
    root->right->right = binary_tree_node(root, 402);
    root->left->left = binary_tree_node(root->left, 10);

    binary_tree_print(root);
    complete = binary_tree_is_complete(root);
    printf("Is %d complete: %d\n", root->n, complete);
    complete = binary_tree_is_complete(root->left);
    printf("Is %d complete: %d\n", root->left->n, complete);

    root->right->left = binary_tree_node(root->right, 112);
    binary_tree_print(root);
    complete = binary_tree_is_complete(root);
    printf("Is %d complete: %d\n", root->n, complete);

    root->left->left->left = binary_tree_node(root->left->left, 8);
    binary_tree_print(root);
    complete = binary_tree_is_complete(root);
    printf("Is %d complete: %d\n", root->n, complete);

    root->left->right->left = binary_tree_node(root->left->right, 23);
    binary_tree_print(root);
    complete = binary_tree_is_complete(root);
    printf("Is %d complete: %d\n", root->n, complete);

    binary_tree_delete(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 102-
main.c 102-binary_tree_is_complete.c 0-binary_tree_node.c 3-binary_tree_delete.c -o
102-complete
alex@/tmp/binary_trees$ ./102-complete
       .-------(098)--.
  .--(012)--.       (128)--.
```

```
(010)     (054)          (402)
(/)
Is 98 complete: 0
Is 12 complete: 1
       .-------(098)-------.
  .--(012)--.         .--(128)--.
(010)     (054)     (112)     (402)
Is 98 complete: 1
            .-------(098)-------.
       .--(012)--.         .--(128)--.
  .--(010)     (054)     (112)     (402)
(008)
Is 98 complete: 1
            .------------(098)-------.
       .--(012)-------.         .--(128)--.
  .--(010)       .--(054)     (112)     (402)
(008)          (023)
Is 98 complete: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `102-binary_tree_is_complete.c`

Done! | Help | Check your code | Get a sandbox | QA Review

22. Rotate left

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that performs a left-rotation on a binary tree

- Prototype: `binary_tree_t *binary_tree_rotate_left(binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to rotate
- Your function must return a pointer to the new root node of the tree once rotated

```
alex@/tmp/binary_trees$ cat 103-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->right = binary_tree_node(root, 128);
    root->right->right = binary_tree_node(root->right, 402);
    binary_tree_print(root);
    printf("Rotate-left %d\n", root->n);
    root = binary_tree_rotate_left(root);
    binary_tree_print(root);
    printf("\n");

    root->right->right = binary_tree_node(root->right, 450);
    root->right->left = binary_tree_node(root->right, 420);
    binary_tree_print(root);
    printf("Rotate-left %d\n", root->n);
    root = binary_tree_rotate_left(root);
    binary_tree_print(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 103-
binary_tree_rotate_left.c 103-main.c 0-binary_tree_node.c -o 103-rotl
alex@/tmp/binary_trees$ ./103-rotl
(098)--.
     (128)--.
          (402)
Rotate-left 98
  .--(128)--.
(098)     (402)

  .--(128)-------.
(098)       .--(402)--.
          (420)     (450)
Rotate-left 128
       .-------(402)--.
  .--(128)--.       (450)
(098)     (420)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `103-binary_tree_rotate_left.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

23. Rotate right

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that performs a right-rotation on a binary tree

- Prototype: `binary_tree_t *binary_tree_rotate_right(binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to rotate
- Your function must return a pointer to the new root node of the tree once rotated

```
alex@/tmp/binary_trees$ cat 104-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 64);
    root->left->left = binary_tree_node(root->left, 32);
    binary_tree_print(root);
    printf("Rotate-right %d\n", root->n);
    root = binary_tree_rotate_right(root);
    binary_tree_print(root);
    printf("\n");

    root->left->left = binary_tree_node(root->left, 20);
    root->left->right = binary_tree_node(root->left, 56);
    binary_tree_print(root);
    printf("Rotate-right %d\n", root->n);
    root = binary_tree_rotate_right(root);
    binary_tree_print(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 104-
binary_tree_rotate_right.c 104-main.c 0-binary_tree_node.c -o 104-rotr
alex@/tmp/binary_trees$ ./104-rotr
       .--(098)
  .--(064)
(032)
Rotate-right 98
  .--(064)--.
(032)     (098)

       .-------(064)--.
  .--(032)--.       (098)
(020)     (056)
Rotate-right 64
  .--(032)-------.
(020)       .--(064)--.
          (056)     (098)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `104-binary_tree_rotate_right.c`

Done! | Help | Check your code | Get a sandbox | QA Review

24. Is BST

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that checks if a binary tree is a valid Binary Search Tree (/rltoken/qO5dBlMnYJzbaWG3xVpcnQ)

- Prototype: `int binary_tree_is_bst(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to check
- Your function must return `1` if `tree` is a valid BST, and `0` otherwise
- If `tree` is `NULL`, return `0`

Properties of a Binary Search Tree:

- The left subtree of a node contains only nodes with values less than the node's value
- The right subtree of a node contains only nodes with values greater than the node's value
- The left and right subtree each must also be a binary search tree
- There must be no duplicate values

```
alex@/tmp/binary_trees$ cat 110-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int bst;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 128);
    root->left->right = binary_tree_node(root->left, 54);
    root->right->right = binary_tree_node(root, 402);
    root->left->left = binary_tree_node(root->left, 10);

    binary_tree_print(root);
    bst = binary_tree_is_bst(root);
    printf("Is %d bst: %d\n", root->n, bst);
    bst = binary_tree_is_bst(root->left);
    printf("Is %d bst: %d\n", root->left->n, bst);

    root->right->left = binary_tree_node(root->right, 97);
    binary_tree_print(root);
    bst = binary_tree_is_bst(root);
    printf("Is %d bst: %d\n", root->n, bst);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 110-
main.c 110-binary_tree_is_bst.c 0-binary_tree_node.c -o 110-is_bst
alex@/tmp/binary_trees$ ./110-is_bst
       .-------(098)--.
  .--(012)--.       (128)--.
(010)     (054)          (402)
Is 98 bst: 1
Is 12 bst: 1
       .-------(098)-------.
  .--(012)--.         .--(128)--.
(010)     (054)     (097)     (402)
Is 98 bst: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `110-binary_tree_is_bst.c`

(/)

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

25. BST - Insert

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that inserts a value in a Binary Search Tree

- Prototype: `bst_t *bst_insert(bst_t **tree, int value);`
- Where `tree` is a double pointer to the root node of the BST to insert the value
- And `value` is the value to store in the node to be inserted
- Your function must return a pointer to the created node, or `NULL` on failure
- If the address stored in `tree` is `NULL`, the created node must become the root node.
- If the value is already present in the tree, it must be ignored

Your file `0-binary_tree_node.c` will be compile during the correction

(/)

```
alex@/tmp/binary_trees$ cat 111-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    bst_t *root;
    bst_t *node;

    root = NULL;
    node = bst_insert(&root, 98);
    printf("Inserted: %d\n", node->n);
    node = bst_insert(&root, 402);
    printf("Inserted: %d\n", node->n);
    node = bst_insert(&root, 12);
    printf("Inserted: %d\n", node->n);
    node = bst_insert(&root, 46);
    printf("Inserted: %d\n", node->n);
    node = bst_insert(&root, 128);
    printf("Inserted: %d\n", node->n);
    node = bst_insert(&root, 256);
    printf("Inserted: %d\n", node->n);
    node = bst_insert(&root, 512);
    printf("Inserted: %d\n", node->n);
    node = bst_insert(&root, 1);
    printf("Inserted: %d\n", node->n);
    node = bst_insert(&root, 128);
    printf("Node should be nil -> %p\n", (void *)node);
    binary_tree_print(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 111-
bst_insert.c 111-main.c 0-binary_tree_node.c -o 111-bst_insert
alex@/tmp/binary_trees$ ./111-bst_insert
Inserted: 98
Inserted: 402
Inserted: 12
Inserted: 46
Inserted: 128
Inserted: 256
Inserted: 512
Inserted: 1
Node should be nil -> (nil)
       .-------(098)------------.
  .--(012)--.         .-------(402)--.
(001)     (046)     (128)--.       (512)
```

```
                                (256)
(a)lex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `111-bst_insert.c, 0-binary_tree_node.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

26. BST - Array to BST

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that builds a Binary Search Tree from an array

- Prototype: `bst_t *array_to_bst(int *array, size_t size);`
- Where `array` is a pointer to the first element of the array to be converted
- And `size` is the number of element in the array
- Your function must return a pointer to the root node of the created BST, or `NULL` on failure
- If a value of the array is already present in the tree, this value must be ignored

Your files `111-bst_insert.c` and `0-binary_tree_node.c` will be compiled during the correction

```
alex@/tmp/binary_trees$ cat 112-main.c
#include <stdlib.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    bst_t *tree;
    int array[] = {
        79, 47, 68, 87, 84, 91, 21, 32, 34, 2,
        20, 22, 98, 1, 62, 95
    };
    size_t n = sizeof(array) / sizeof(array[0]);

    tree = array_to_bst(array, n);
    if (!tree)
        return (1);
    binary_tree_print(tree);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 112-
array_to_bst.c 112-main.c 111-bst_insert.c 0-binary_tree_node.c -o 112-bst_array
alex@/tmp/binary_trees$ ./112-bst_array
                                .------------(079)-------.
                .-----------------(047)-------.         .--(087)--.
      .-------(021)-------.             .--(068)     (084)     (091)-------.
  .--(002)--.         .--(032)--.       (062)                           .--(098)
(001)     (020)     (022)     (034)                                 (095)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `112-array_to_bst.c, 111-bst_insert.c, 0-binary_tree_node.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

27. BST - Search

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that searches for a value in a Binary Search Tree

- Prototype: `bst_t *bst_search(const bst_t *tree, int value);`

- Where `tree` is a pointer to the root node of the BST to search
- And `value` is the value to search in the tree
- Your function must return a pointer to the node containing a value equals to `value`
- If `tree` is `NULL` or if nothing is found, your function must return `NULL`

```
alex@/tmp/binary_trees$ cat 113-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    bst_t *tree;
    int array[] = {
        79, 47, 68, 87, 84, 91, 21, 32, 34, 2,
        20, 22, 98, 1, 62, 95
    };
    size_t n = sizeof(array) / sizeof(array[0]);
    bst_t *node;

    tree = array_to_bst(array, n);
    if (!tree)
        return (1);
    binary_tree_print(tree);
    node = bst_search(tree, 32);
    printf("Found: %d\n", node->n);
    binary_tree_print(node);
    node = bst_search(tree, 512);
    printf("Node should be nil -> %p\n", (void *)node);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 113-
bst_search.c 113-main.c 112-array_to_bst.c 111-bst_insert.c 0-binary_tree_node.c -o
113-bst_search
alex@/tmp/binary_trees$ ./113-bst_search
                                  .------------(079)-------.
                .-----------------(047)-------.         .--(087)--.
      .-------(021)-------.             .--(068)     (084)     (091)-------.
  .--(002)--.         .--(032)--.       (062)                           .--(098)
(001)     (020)     (022)     (034)                                 (095)
Found: 32
  .--(032)--.
(022)     (034)
Node should be nil -> (nil)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `113-bst_search.c`

Done! | Help | Check your code | Get a sandbox | QA Review

28. BST - Remove

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that removes a node from a Binary Search Tree

- Prototype: `bst_t *bst_remove(bst_t *root, int value);`
- Where `root` is a pointer to the root node of the tree where you will remove a node
- And `value` is the value to remove in the tree
- Once located, the node containing a value equals to `value` must be removed and freed
- If the node to be deleted has two children, it must be replaced with its first `in-order successor` (not predecessor)
- Your function must return a pointer to the new root node of the tree after removing the desired value

(/)

```
alex@/tmp/binary_trees$ cat 114-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    bst_t *tree;
    int array[] = {
        79, 47, 68, 87, 84, 91, 21, 32, 34, 2,
        20, 22, 98, 1, 62, 95
    };
    size_t n = sizeof(array) / sizeof(array[0]);

    tree = array_to_bst(array, n);
    if (!tree)
        return (1);
    binary_tree_print(tree);

    tree = bst_remove(tree, 79);
    printf("Removed 79...\n");
    binary_tree_print(tree);

    tree = bst_remove(tree, 21);
    printf("Removed 21...\n");
    binary_tree_print(tree);

    tree = bst_remove(tree, 68);
    printf("Removed 68...\n");
    binary_tree_print(tree);
    binary_tree_delete(tree);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 114-
bst_remove.c 114-main.c 112-array_to_bst.c 111-bst_insert.c 0-binary_tree_node.c 3-b
inary_tree_delete.c -o 114-bst_rm
alex@/tmp/binary_trees$ valgrind ./114-bst_rm
==14720== Memcheck, a memory error detector
==14720== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==14720== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==14720== Command: ./114-bst_rm
==14720==
                                .------------(079)-------.
                .-----------------(047)-------.         .--(087)--.
       .--------(021)-------.              .--(068)     (084)     (091)-------.
  .--(002)--.         .--(032)--.       (062)                           .--(098)
(001)     (020)     (022)     (034)                                   (095)
```

```
Removed 79...
(/)                                   .------------(084)--.
                 .-----------------(047)-------.       (087)--.
       .-------(021)-------.              .--(068)          (091)-------.
  .--(002)--.         .--(032)--.       (062)                    .--(098)
(001)     (020)     (022)     (034)                           (095)
Removed 21...
                                .------------(084)--.
                 .------------(047)-------.       (087)--.
       .-------(022)--.              .--(068)          (091)-------.
  .--(002)--.       (032)--.       (062)                    .--(098)
(001)     (020)          (034)                           (095)
Removed 68...
                                .-------(084)--.
                 .------------(047)--.       (087)--.
       .-------(022)--.            (062)          (091)-------.
  .--(002)--.       (032)--.                              .--(098)
(001)     (020)          (034)                          (095)
==14720==
==14720== HEAP SUMMARY:
==14720==     in use at exit: 0 bytes in 0 blocks
==14720==   total heap usage: 40 allocs, 40 frees, 5,772 bytes allocated
==14720==
==14720== All heap blocks were freed -- no leaks are possible
==14720==
==14720== For counts of detected and suppressed errors, rerun with: -v
==14720== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `114-bst_remove.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

29. Big O #BST

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

What are the average time complexities of those operations on a Binary Search Tree (one answer per line):

- Inserting the value `n`
- Removing the node with the value `n`
- Searching for a node in a BST of size n

Repo:

- GitHub repository: `binary_trees`
- File: `115-0`

(/)

Done! | Help | Check your code | Get a sandbox | QA Review

30. Is AVL

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that checks if a binary tree is a valid AVL Tree (/rltoken/fMAZ9aBS-rDWgeIAvdTKWw)

- Prototype: `int binary_tree_is_avl(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to check
- Your function must return `1` if `tree` is a valid AVL Tree, and `0` otherwise
- If `tree` is `NULL`, return `0`

Properties of an AVL Tree:

- An AVL Tree is a BST
- The difference between heights of left and right subtrees cannot be more than one
- The left and right subtrees must also be AVL trees

```
alex@/tmp/binary_trees$ cat 120-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * basic_tree - Build a basic binary tree
 *
 * Return: A pointer to the created tree
 */
binary_tree_t *basic_tree(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 12);
    root->right = binary_tree_node(root, 128);
    root->left->right = binary_tree_node(root->left, 54);
    root->right->right = binary_tree_node(root, 402);
    root->left->left = binary_tree_node(root->left, 10);
    return (root);
}

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int avl;

    root = basic_tree();

    binary_tree_print(root);
    avl = binary_tree_is_avl(root);
    printf("Is %d avl: %d\n", root->n, avl);
    avl = binary_tree_is_avl(root->left);
    printf("Is %d avl: %d\n", root->left->n, avl);

    root->right->left = binary_tree_node(root->right, 97);
    binary_tree_print(root);
    avl = binary_tree_is_avl(root);
    printf("Is %d avl: %d\n", root->n, avl);

    root = basic_tree();
    root->right->right->right = binary_tree_node(root->right->right, 430);
    binary_tree_print(root);
    avl = binary_tree_is_avl(root);
    printf("Is %d avl: %d\n", root->n, avl);
```

```
    root->right->right->right->left = binary_tree_node(root->right->right->right, 42
(6));
    binary_tree_print(root);
    avl = binary_tree_is_avl(root);
    printf("Is %d avl: %d\n", root->n, avl);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 120-
main.c 120-binary_tree_is_avl.c 0-binary_tree_node.c -o 120-is_avl
alex@/tmp/binary_trees$ ./120-is_avl
       .-------(098)--.
  .--(012)--.       (128)--.
(010)     (054)          (402)
Is 98 avl: 1
Is 12 avl: 1
       .-------(098)-------.
  .--(012)--.         .--(128)--.
(010)     (054)     (097)     (402)
Is 98 avl: 0
       .-------(098)--.
  .--(012)--.       (128)--.
(010)     (054)          (402)--.
                              (430)
Is 98 avl: 0
       .-------(098)--.
  .--(012)--.       (128)--.
(010)     (054)          (402)-------.
                                .--(430)
                              (420)
Is 98 avl: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `120-binary_tree_is_avl.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

31. AVL - Insert

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that inserts a value in an AVL Tree

- Prototype: `avl_t *avl_insert(avl_t **tree, int value);`
- Where `tree` is a double pointer to the root node of the AVL tree for inserting the value
- And `value` is the value to store in the node to be inserted

- Your function must return a pointer to the created node, or `NULL` on failure
- If the address stored in `tree` is `NULL`, the created node must become the root node.
- The resulting tree after insertion, must be a balanced AVL Tree

Your files `14-binary_tree_balance.c`, `103-binary_tree_rotate_left.c`, `104-binary_tree_rotate_right.c` and `0-binary_tree_node.c` will be compiled during the correction

```
alex@/tmp/binary_trees$ cat 121-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    avl_t *root;
    avl_t *node;

    root = NULL;
    node = avl_insert(&root, 98);
    printf("Inserted: %d\n", node->n);
    binary_tree_print(root);
    node = avl_insert(&root, 402);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = avl_insert(&root, 12);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = avl_insert(&root, 46);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = avl_insert(&root, 128);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = avl_insert(&root, 256);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = avl_insert(&root, 512);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = avl_insert(&root, 50);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 121-
avl_insert.c 121-main.c 14-binary_tree_balance.c 103-binary_tree_rotate_left.c 104-b
inary_tree_rotate_right.c 0-binary_tree_node.c -o 121-avl_insert
alex@/tmp/binary_trees$ ./121-avl_insert
Inserted: 98
(098)

Inserted: 402
(098)--.
     (402)
```

```
(/)nserted: 12
  .--(098)--.
(012)     (402)

Inserted: 46
  .-------(098)--.
(012)--.       (402)
     (046)

Inserted: 128
  .-------(098)-------.
(012)--.         .--(402)
     (046)     (128)

Inserted: 256
  .-------(098)-------.
(012)--.         .--(256)--.
     (046)     (128)     (402)

Inserted: 512
  .-------(098)-------.
(012)--.         .--(256)--.
     (046)     (128)     (402)--.
                              (512)

Inserted: 50
       .-------(098)-------.
  .--(046)--.         .--(256)--.
(012)     (050)     (128)     (402)--.
                                    (512)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `121-avl_insert.c, 14-binary_tree_balance.c, 103-binary_tree_rotate_left.c, 104-binary_tree_rotate_right.c, 0-binary_tree_node.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

32. AVL - Array to AVL

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that builds an AVL tree from an array

- Prototype: `avl_t *array_to_avl(int *array, size_t size);`

- Where `array` is a pointer to the first element of the array to be converted
- And `size` is the number of element in the array
- Your function must return a pointer to the root node of the created AVL tree, or `NULL` on failure
- If a value of the array is already present in the tree, this value must be ignored

Your files `121-avl_insert.c`, `0-binary_tree_node.c`, `14-binary_tree_balance.c`, `103-binary_tree_rotate_left.c` and `104-binary_tree_rotate_right.c` will be compiled during the correction

```
alex@/tmp/binary_trees$ cat 122-main.c
#include <stdlib.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    avl_t *tree;
    int array[] = {
        79, 47, 68, 87, 84, 91, 21, 32, 34, 2,
        20, 22, 98, 1, 62, 95
    };
    size_t n = sizeof(array) / sizeof(array[0]);

    tree = array_to_avl(array, n);
    if (!tree)
        return (1);
    binary_tree_print(tree);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 122-
array_to_avl.c 122-main.c 121-avl_insert.c 0-binary_tree_node.c 14-binary_tree_balan
ce.c 103-binary_tree_rotate_left.c 104-binary_tree_rotate_right.c -o 122-avl_array
alex@/tmp/binary_trees$ ./122-avl_array
                .-----------------(047)-----------------.
       .-------(021)-------.                   .-------(084)-------.
  .--(002)--.         .--(032)--.         .--(068)--.         .--(091)-------.
(001)     (020)     (022)     (034)     (062)     (079)     (087)       .--(098)
                                                                       (095)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `122-array_to_avl.c, 121-avl_insert.c, 0-binary_tree_node.c, 103-binary_tree_rotate_left.c, 104-binary_tree_rotate_right.c, 14-binary_tree_balance.c`

(/) Done? | Help | Check your code | Ask for a new correction | >_ Get a sandbox | QA Review

33. AVL - Remove

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that removes a node from an AVL tree

- Prototype: `avl_t *avl_remove(avl_t *root, int value);`
- Where `root` is a pointer to the root node of the tree for removing a node
- And `value` is the value to remove in the tree
- Once located, the node containing a value equals to `value` must be removed and freed
- If the node to be deleted has two children, it must be replaced with its first `in-order successor` (not predecessor)
- After deletion of the desired node, the tree must be rebalanced if necessary
- Your function must return a pointer to the new root node of the tree after removing the desired value, and after rebalancing

Your files `14-binary_tree_balance.c`, `103-binary_tree_rotate_left.c` and `104-binary_tree_rotate_right.c` will be compiled during the correction

```
alex@/tmp/binary_trees$ cat 123-main.c
#include <stdio.h>
#include <stdlib.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    avl_t *tree;
    int array[] = {
        79, 47, 68, 87, 84, 91, 21, 32, 34, 2,
        20, 22, 98, 1, 62, 95
    };
    size_t n = sizeof(array) / sizeof(array[0]);

    tree = array_to_avl(array, n);
    if (!tree)
        return (1);
    binary_tree_print(tree);

    tree = avl_remove(tree, 47);
    printf("Removed 47...\n");
    binary_tree_print(tree);

    tree = avl_remove(tree, 79);
    printf("Removed 79...\n");
    binary_tree_print(tree);

    tree = avl_remove(tree, 32);
    printf("Removed 32...\n");
    binary_tree_print(tree);

    tree = avl_remove(tree, 34);
    printf("Removed 34...\n");
    binary_tree_print(tree);

    tree = avl_remove(tree, 22);
    printf("Removed 22...\n");
    binary_tree_print(tree);
    binary_tree_delete(tree);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 123-
avl_remove.c 123-main.c 103-binary_tree_rotate_left.c 104-binary_tree_rotate_right.c
122-array_to_avl.c 121-avl_insert.c 14-binary_tree_balance.c 3-binary_tree_delete.c
0-binary_tree_node.c -o 123-avl_rm
alex@/tmp/binary_trees$ valgrind ./123-avl_rm
==15646== Memcheck, a memory error detector
```

```
==15646== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==15646== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==15646== Command: ./123-avl_rm
==15646==
                .-----------------(047)-----------------.
       .-------(021)-------.                   .-------(084)-------.
  .--(002)--.         .--(032)--.         .--(068)--.         .--(091)-------.
(001)     (020)     (022)     (034)     (062)     (079)     (087)       .--(098)
                                                                        (095)
Removed 47...
                .-----------------(062)-----------.
       .-------(021)-------.              .-------(084)-------.
  .--(002)--.         .--(032)--.       (068)--.         .--(091)-------.
(001)     (020)     (022)     (034)          (079)     (087)       .--(098)
                                                                  (095)
Removed 79...
                .-----------------(062)-----------------.
       .-------(021)-------.                   .-------(091)-------.
  .--(002)--.         .--(032)--.         .--(084)--.         .--(098)
(001)     (020)     (022)     (034)     (068)     (087)     (095)
Removed 32...
                .------------(062)-----------------.
       .-------(021)-------.              .-------(091)-------.
  .--(002)--.         .--(034)       .--(084)--.         .--(098)
(001)     (020)     (022)          (068)     (087)     (095)
Removed 34...
                .-------(062)-----------------.
       .-------(021)--.              .-------(091)-------.
  .--(002)--.       (022)       .--(084)--.         .--(098)
(001)     (020)               (068)     (087)     (095)
Removed 22...
       .------------(062)-----------------.
  .--(002)-------.              .-------(091)-------.
(001)       .--(021)       .--(084)--.         .--(098)
          (020)          (068)     (087)     (095)
==15646==
==15646== HEAP SUMMARY:
==15646==     in use at exit: 0 bytes in 0 blocks
==15646==   total heap usage: 48 allocs, 48 frees, 7,350 bytes allocated
==15646==
==15646== All heap blocks were freed -- no leaks are possible
==15646==
==15646== For counts of detected and suppressed errors, rerun with: -v
==15646== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`

(/)

- File: `123-avl_remove.c, 14-binary_tree_balance.c, 103-binary_tree_rotate_left.c, 104-binary_tree_rotate_right.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

34. AVL - From sorted array

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that builds an AVL tree from an array

- Prototype: `avl_t *sorted_array_to_avl(int *array, size_t size);`
- Where `array` is a pointer to the first element of the array to be converted
- And `size` is the number of element in the array
- Your function must return a pointer to the root node of the created AVL tree, or `NULL` on failure
- You can assume there will be no duplicate value in the array
- You are not allowed to rotate
- You can only have 2 functions in your file

Your file `0-binary_tree_node.c` will be compiled during the correction

```
alex@/tmp/binary_trees$ cat 124-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * print_array - Prints an array of integers
 *
 * @array: The array to be printed
 * @size: Size of the array
 */
void print_array(const int *array, size_t size)
{
    size_t i;

    for (i = 0; i < size; ++i)
        printf("(%03d)", array[i]);
    printf("\n");
}

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    avl_t *tree;
    int array[] = {
        1, 2, 20, 21, 22, 32, 34, 47, 62, 68,
        79, 84, 87, 91, 95, 98
    };
    size_t n = sizeof(array) / sizeof(array[0]);

    tree = sorted_array_to_avl(array, n);
    if (!tree)
        return (1);
    print_array(array, n);
    binary_tree_print(tree);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 124-
main.c 124-sorted_array_to_avl.c 0-binary_tree_node.c -o 124-avl_sorted
alex@/tmp/binary_trees$ ./124-avl_sorted
(001)(002)(020)(021)(022)(032)(034)(047)(062)(068)(079)(084)(087)(091)(095)(098)
                 .-----------------(047)-----------------.
       .-------(021)-------.                   .-------(084)-------.
  .--(002)--.         .--(032)--.         .--(068)--.         .--(091)--.
(001)     (020)     (022)     (034)     (062)     (079)     (087)     (095)--.
                                                                            (098)
alex@/tmp/binary_trees$
```

(/)

Repo:

- GitHub repository: `binary_trees`
- File: `124-sorted_array_to_avl.c, 0-binary_tree_node.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

35. Big O #AVL Tree

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

What are the average time complexities of those operations on an AVL Tree (one answer per line):

- Inserting the value `n`
- Removing the node with the value `n`
- Searching for a node in an AVL tree of size n

Repo:

- GitHub repository: `binary_trees`
- File: `125-O`

Done? | Help | Check your code | Get a sandbox | QA Review

36. Is Binary heap

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that checks if a binary tree is a valid Max Binary Heap (/rltoken/TU_7dyDvU6XqO_T0elQk4Q)

- Prototype: `int binary_tree_is_heap(const binary_tree_t *tree);`
- Where `tree` is a pointer to the root node of the tree to check
- Your function must return `1` if `tree` is a valid Max Binary Heap, and `0` otherwise
- If `tree` is `NULL`, return `0`

Properties of a Max Binary Heap:

- It's a complete tree
- In a Max Binary Heap, the value at root must be maximum among all values present in Binary Heap
- The last property must be recursively true for all nodes in Binary Tree

```
alex@/tmp/binary_trees$ cat 130-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * basic_tree - Build a basic binary tree
 *
 * Return: A pointer to the created tree
 */
binary_tree_t *basic_tree(void)
{
    binary_tree_t *root;

    root = binary_tree_node(NULL, 98);
    root->left = binary_tree_node(root, 90);
    root->right = binary_tree_node(root, 85);
    root->left->right = binary_tree_node(root->left, 80);
    root->left->left = binary_tree_node(root->left, 79);
    return (root);
}

/**
 * main - Entry point
 *
 * Return: Always 0 (Success)
 */
int main(void)
{
    binary_tree_t *root;
    int heap;

    root = basic_tree();

    binary_tree_print(root);
    heap = binary_tree_is_heap(root);
    printf("Is %d heap: %d\n", root->n, heap);
    heap = binary_tree_is_heap(root->left);
    printf("Is %d heap: %d\n", root->left->n, heap);

    root->right->left = binary_tree_node(root->right, 97);
    binary_tree_print(root);
    heap = binary_tree_is_heap(root);
    printf("Is %d heap: %d\n", root->n, heap);

    root = basic_tree();
    root->right->right = binary_tree_node(root->right, 79);
    binary_tree_print(root);
    heap = binary_tree_is_heap(root);
    printf("Is %d heap: %d\n", root->n, heap);
    return (0);
}
```

```
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 130-main.c 130-binary_tree_is_heap.c 0-binary_tree_node.c -o 130-is_heap
alex@/tmp/binary_trees$ ./130-is_heap
       .-------(098)--.
  .--(090)--.       (085)
(079)     (080)
Is 98 heap: 1
Is 90 heap: 1
       .-------(098)-------.
  .--(090)--.         .--(085)
(079)     (080)     (097)
Is 98 heap: 0
       .-------(098)--.
  .--(090)--.       (085)--.
(079)     (080)          (079)
Is 98 heap: 0
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `130-binary_tree_is_heap.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

37. Heap - Insert

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that inserts a value in Max Binary Heap

- Prototype: `heap_t *heap_insert(heap_t **root, int value)`
- Where `root` is a double pointer to the root node of the Heap to insert the value
- And `value` is the value to store in the node to be inserted
- Your function must return a pointer to the created node, or `NULL` on failure
- If the address stored in `root` is `NULL`, the created node must become the root node.
- You have to respect a `Max Heap` ordering
- You are allowed to have up to `6` functions in your file

Your file `0-binary_tree_node.c` will be compiled during the correction

```
alex@/tmp/binary_trees$ cat 131-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    heap_t *root;
    heap_t *node;

    root = NULL;
    node = heap_insert(&root, 98);
    printf("Inserted: %d\n", node->n);
    binary_tree_print(root);
    node = heap_insert(&root, 402);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = heap_insert(&root, 12);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = heap_insert(&root, 46);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = heap_insert(&root, 128);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = heap_insert(&root, 256);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = heap_insert(&root, 512);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    node = heap_insert(&root, 50);
    printf("\nInserted: %d\n", node->n);
    binary_tree_print(root);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 131-
main.c 131-heap_insert.c 0-binary_tree_node.c -o 131-heap_insert
alex@/tmp/binary_trees$ ./131-heap_insert
Inserted: 98
(098)

Inserted: 402
  .--(402)
(098)
```

```
Inserted: 12
(/).--(402)--.
(098)     (012)

Inserted: 46
       .--(402)--.
  .--(098)     (012)
(046)

Inserted: 128
       .--------(402)--.
  .--(128)--.       (012)
(046)     (098)

Inserted: 256
       .--------(402)-------.
  .--(128)--.         .--(256)
(046)     (098)     (012)

Inserted: 512
       .--------(512)-------.
  .--(128)--.         .--(402)--.
(046)     (098)     (012)     (256)

Inserted: 50
            .--------(512)-------.
       .--(128)--.         .--(402)--.
  .--(050)     (098)     (012)     (256)
(046)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `131-heap_insert.c, 0-binary_tree_node.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

38. Heap - Array to Binary Heap

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that builds a Max Binary Heap tree from an array

- Prototype: `heap_t *array_to_heap(int *array, size_t size);`
- Where array is a pointer to the first element of the array to be converted
- And size is the number of element in the array
- Your function must return a pointer to the root node of the created Binary Heap, or NULL on failure

Your files `131-heap_insert.c` and `0-binary_tree_node.c` will be compiled during the correction (/)

```
alex@/tmp/binary_trees$ cat 132-main.c
#include <stdlib.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    heap_t *tree;
    int array[] = {
        79, 47, 68, 87, 84, 91, 21, 32, 34, 2,
        20, 22, 98, 1, 62, 95
    };
    size_t n = sizeof(array) / sizeof(array[0]);

    tree = array_to_heap(array, n);
    if (!tree)
        return (1);
    binary_tree_print(tree);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 132-
main.c 132-array_to_heap.c 131-heap_insert.c 0-binary_tree_node.c -o 132-heap_array
alex@/tmp/binary_trees$ ./132-heap_array
                    .-----------------(098)-----------------.
           .-------(095)-------.                   .-------(091)-------.
       .--(084)--.         .--(079)--.         .--(087)--.         .--(062)--.
  .--(047)     (034)     (002)     (020)     (022)     (068)     (001)     (021)
(032)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `132-array_to_heap.c, 131-heap_insert.c, 0-binary_tree_node.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

39. Heap - Extract

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that extracts the root node of a Max Binary Heap

- Prototype: `int heap_extract(heap_t **root);`
- Where `root` is a double pointer to the root node of heap
- Your function must return the value stored in the root node
- The root node must be freed and replace with the last `level-order` node of the heap
- Once replaced, the heap must be rebuilt if necessary
- If your function fails, return `0`

(/)

```
alex@/tmp/binary_trees$ cat 133-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    heap_t *tree;
    int array[] = {
        79, 47, 68, 87, 84, 91, 21, 32, 34, 2,
        20, 22, 98, 1, 62, 95
    };
    size_t n = sizeof(array) / sizeof(array[0]);
    int extract;

    tree = array_to_heap(array, n);
    if (!tree)
        return (1);
    binary_tree_print(tree);

    extract = heap_extract(&tree);
    printf("Extracted: %d\n", extract);
    binary_tree_print(tree);

    extract = heap_extract(&tree);
    printf("Extracted: %d\n", extract);
    binary_tree_print(tree);

    extract = heap_extract(&tree);
    printf("Extracted: %d\n", extract);
    binary_tree_print(tree);
    binary_tree_delete(tree);
    return (0);
}
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 133-
main.c 133-heap_extract.c 132-array_to_heap.c 131-heap_insert.c 3-binary_tree_delet
e.c -o 133-heap_extract
alex@/tmp/binary_trees$ valgrind ./133-heap_extract
==29133== Memcheck, a memory error detector
==29133== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==29133== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==29133== Command: ./133-heap_extract
==29133==
                      .-----------------(098)-----------------.
            .-------(095)-------.                   .-------(091)-------.
       .--(084)--.         .--(079)--.         .--(087)--.         .--(062)--.
  .--(047)     (034)     (002)     (020)     (022)     (068)     (001)     (021)
```

```
(032)
(/)
Extracted: 98
                .-----------------(095)-----------------.
      .-------(084)-------.                 .-------(091)-------.
  .--(047)--.         .--(079)--.         .--(087)--.         .--(062)--.
(032)     (034)     (002)     (020)     (022)     (068)     (001)     (021)
Extracted: 95
                .-----------------(091)-----------------.
      .-------(084)-------.                 .-------(087)-------.
  .--(047)--.         .--(079)--.         .--(068)--.         .--(062)
(032)     (034)     (002)     (020)     (022)     (021)     (001)
Extracted: 91
                .-----------------(087)-----------------.
      .-------(084)-------.                 .-------(068)--.
  .--(047)--.         .--(079)--.         .--(022)--.       (062)
(032)     (034)     (002)     (020)     (001)     (021)
==29133==
==29133== HEAP SUMMARY:
==29133==     in use at exit: 0 bytes in 0 blocks
==29133==   total heap usage: 213 allocs, 213 frees, 9,063 bytes allocated
==29133==
==29133== All heap blocks were freed -- no leaks are possible
==29133==
==29133== For counts of detected and suppressed errors, rerun with: -v
==29133== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `133-heap_extract.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

40. Heap - Sort

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that converts a Binary Max Heap to a sorted array of integers

- Prototype: `int *heap_to_sorted_array(heap_t *heap, size_t *size);`
- Where `heap` is a pointer to the root node of the heap to convert
- And `size` is an address to store the size of the array
- You can assume `size` is a valid address
- Since we are using Max Heap, the returned array must be sorted in descending order

Your file `133-heap_extract.c` will be compile during the correction

```
alex@/tmp/binary_trees$ cat 134-main.c
#include <stdlib.h>
#include <stdio.h>
#include "binary_trees.h"

/**
 * print_array - Prints an array of integers
 *
 * @array: The array to be printed
 * @size: Number of elements in @array
 */
void print_array(const int *array, size_t size)
{
    size_t i;

    i = 0;
    while (array && i < size)
    {
        if (i > 0)
            printf(", ");
        printf("%d", array[i]);
        ++i;
    }
    printf("\n");
}

/**
 * main - Entry point
 *
 * Return: 0 on success, error code on failure
 */
int main(void)
{
    heap_t *tree;
    int array[] = {
        79, 47, 68, 87, 84, 91, 21, 32, 34, 2,
        20, 22, 98, 1, 62, 95
    };
    size_t n = sizeof(array) / sizeof(array[0]);
    int *sorted;
    size_t sorted_size;

    print_array(array, n);
    tree = array_to_heap(array, n);
    if (!tree)
        return (1);
    binary_tree_print(tree);
    sorted = heap_to_sorted_array(tree, &sorted_size);
    print_array(sorted, sorted_size);
    free(sorted);
    return (0);
}
```

```
alex@/tmp/binary_trees$ gcc -Wall -Wextra -Werror -pedantic binary_tree_print.c 134-main.c 134-heap_to_sorted_array.c 133-heap_extract.c 132-array_to_heap.c 131-heap_insert.c -o 134-heap_sort
alex@/tmp/binary_trees$ valgrind ./134-heap_sort
==46529== Memcheck, a memory error detector
==46529== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==46529== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==46529== Command: ./134-heap_sort
==46529==
79, 47, 68, 87, 84, 91, 21, 32, 34, 2, 20, 22, 98, 1, 62, 95
                      .-----------------(098)-----------------.
            .-------(095)-------.                   .-------(091)-------.
       .--(084)--.         .--(079)--.         .--(087)--.         .--(062)--.
  .--(047)     (034)     (002)     (020)     (022)     (068)     (001)     (021)
(032)
98, 95, 91, 87, 84, 79, 68, 62, 47, 34, 32, 22, 21, 20, 2, 1
==46529==
==46529== HEAP SUMMARY:
==46529==     in use at exit: 0 bytes in 0 blocks
==46529==   total heap usage: 301 allocs, 301 frees, 8,323 bytes allocated
==46529==
==46529== All heap blocks were freed -- no leaks are possible
==46529==
==46529== For counts of detected and suppressed errors, rerun with: -v
==46529== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
alex@/tmp/binary_trees$
```

Repo:

- GitHub repository: `binary_trees`
- File: `134-heap_to_sorted_array.c, 133-heap_extract.c`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

41. Big O #Binary Heap

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

What are the average time complexities of those operations on a Binary Heap (one answer per line):

- Inserting the value n
- Extracting the root node
- Searching for a node in a binary heap of size n

Repo:

- GitHub repository: `binary_trees`

- File: `135-0`

(/)

Done? | Help | Check your code | Get a sandbox | QA Review