0x1D. C - Binary trees

CGroup projectAlgorithmData structure

- By: Alexandre Gautier
- Weight: 5
- Project to be done in teams of 2 people (your team: Emmanuel Ahuron)
- Project over - took place from Aug 22, 2022 6:00 AM to Aug 26, 2022 6:00 AM
- An auto review will be launched at the deadline

In a nutshell…

- **Contribution:** 100.0%
- **Auto QA review:** 0.0/157 mandatory & 0.0/196 optional
- **Altogether: 0.0%**
  - Mandatory: 0.0%
  - Optional: 0.0%
  - Contribution: 100.0%
  - Calculation: 100.0% * (0.0% + (0.0% * 0.0%)) == **0.0%**

Resources

Read or watch:

- Binary tree (*note the first line:* `Not to be confused with B-tree.`)
- Data Structure and Algorithms - Tree
- Tree Traversal
- Binary Search Tree
- Data structures: Binary Tree

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone, **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 and 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

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

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-mai
n.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 Ask for a new correction Get a sandbox QA Review

1. Insert left

mandatory

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

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-mai
n.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 Ask for a new correction Get a sandbox QA Review

2. Insert right

mandatory

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

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-mai
n.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 Ask for a new correction Get a sandbox QA Review

3. Delete

mandatory

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

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-mai
n.c 3-binary_tree_delete.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 3-del
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 Ask for a new correction Get a sandbox QA Review

4. Is leaf

mandatory

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

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-bin
ary_tree_is_leaf.c 4-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 4-le
af
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 Ask for a new correction Get a sandbox QA Review

5. Is root

mandatory

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

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-bin
ary_tree_is_root.c 5-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 5-ro
ot
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 Ask for a new correction Get a sandbox QA Review

6. Pre-order traversal

mandatory

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

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-mai
n.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 Ask for a new correction Get a sandbox QA Review

7. In-order traversal

mandatory

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

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-mai
n.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 Ask for a new correction Get a sandbox QA Review

8. Post-order traversal

mandatory

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

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-mai
n.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 Ask for a new correction Get a sandbox QA Review

9. Height

mandatory

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

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-bin
ary_tree_height.c 9-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 9-hei
ght
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 Ask for a new correction Get a sandbox QA Review

10. Depth

mandatory

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

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-bi
nary_tree_depth.c 10-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 10-d
epth
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 Ask for a new correction Get a sandbox QA Review

11. Size

mandatory

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

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-bi
nary_tree_size.c 11-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 11-si
ze
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 Ask for a new correction Get a sandbox QA Review

12. Leaves

mandatory

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

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-bi
nary_tree_leaves.c 12-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 12-
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 Ask for a new correction Get a sandbox QA Review

13. Nodes

mandatory

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

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-bi
nary_tree_nodes.c 13-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c -o 13-n
odes
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 Ask for a new correction Get a sandbox QA Review

14. Balance factor

mandatory

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

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-bi
nary_tree_balance.c 14-main.c 0-binary_tree_node.c 2-binary_tree_insert_right.c 1-bin
ary_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 Ask for a new correction Get a sandbox QA Review

15. Is full

mandatory

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

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-bi
nary_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 Ask for a new correction Get a sandbox QA Review

16. Is perfect

mandatory

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

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-bi
nary_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 Ask for a new correction Get a sandbox QA Review

17. Sibling

mandatory

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

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-ma
in.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 Ask for a new correction Get a sandbox QA Review

18. Uncle

mandatory

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

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-ma
in.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 Ask for a new correction Get a sandbox QA Review

Done with the mandatory tasks? Unlock 23 advanced tasks now!