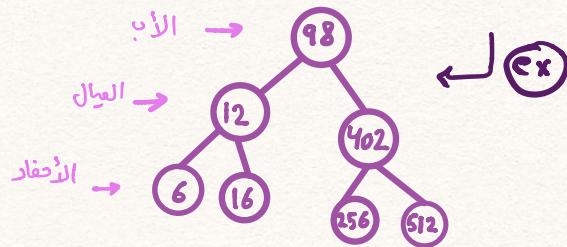


C - Binary Tree

ما هيBinary Tree؟
هي عبارة عن هيكل بيانات (Data Structure) تتكون من Node وكل Node لها مركبة تكون لها:

(Parent) أب
أبن يمين (right child)
أبن يسار (left child)



ويمكن ترجمة هذا التكمل لغة C كالتالي:

```
+typedef struct binary_tree_s
```

```
{
```

```
    int n; → رقم الاب جوانبه
```

```
    struct binary_tree_s *Parent; → مؤشر على الاب
```

```
    struct binary_tree_s *left; → مؤشر على الاب اليسار
```

```
    struct binary_tree_s *right; → مؤشر على الاب اليمين
```

```
binary_tree_t
```

Tasks

0. New node

Score: 100.00% (Checks completed: 100.00%)

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

السؤال بكل سهولة طلب منه تموي واله
تسوي في بسيط وتسو ٩٩

① تموي Node جديد في Tree

لعني هو ما طلب منك شبيه كامله، بنينا
تسوي جزء واحد منها يعني `Node` وترتبها بأبوها
لرمان لها اب

وشن فكره السؤال ٩٩

①

طيب ومن مطلوب مني ٩٩
المطلوب بترجمة والأسئلة

```
binary_tree_t *binary_tree_node(binary_tree_t *parent, int value);
```

هذا المقال تستقبل:

①: أب لـ Node (ممكن يكون NULL لو مواله أب)

②: الـ value الذي تبني تحظى في المقدمة الابدية

وتابعوا:

جدير به:

①: الـ value الذي عطيته المقال

②: تموي منه

(left=NULL, right=NULL)

أمثلة ٤
Root = binary_tree_node(NULL, 98);
Root → left = binary_tree_node(root, 12);
سوينا دل الاب وعندنا ١٢
لورضيف له ولد.

٣
كيف تشتق المقاله داخلها ٩٩

- ١: تجز مكان في المذاكرة (نستعمل malloc)
- ٢: نحط فيه رقم الـ value
- ٣: نخليه يعرف من هو أبده
- ٤: مانقطيه أولاد (نحط قيمة وليس = NULL)
- ٥: تزييج المـ Node الكبير

لعني بالـ ختصار: كأن أقول يا واده سوي شدنه جديد في الشجرة ودخله
اسم (Value) وخله يكون مدللاً (Parent)، ولا تخطه أربعة الخط

1. Insert left

Score: 100.00% (Checks completed: 100.00%)

mandatory

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

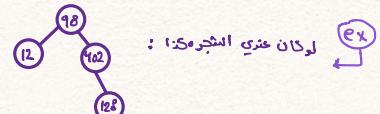
- Prototype: `binary_tree_t *binary_tree_insert_left(binary_tree_t *parent, int value);`
- Where `parent` is a pointer to the node to insert the left-child in
- And `value` is the value to store in the new node
- Your function must return a pointer to the created node, or `NULL` on failure or if `parent` 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.

وشن هکره السؤال ٩٦: يطلب منك تكتب دالة اسمها:

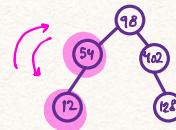
`binary_tree_t *binary_tree_insert_left (binary_tree_t *Parent, int value);`

وهذه الدالة تضيف Node جديد بجهة اليسار من Node موجوده

لكن !! فنيها حرى وشو اركه ٩٩

لو `Node` اللي بي تضيف يسارها ماعنده ولد يسار به خط وده جبناهلنه!! لو عندها ولد يسار مثلاً ٤٦ ← الدالة الكبير يعني مكانه ولد المتمييز بـ ولد لم يتم ايجاده

وليدم كتبته هذه الكود :

`binary_tree_insert_left (root, 54)`

2. Insert right

Score: 100.00% (Checks completed: 100.00%)

mandatory

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.

binary_tree_insert_right

وشن هکره السؤال ٩٧: يغا شوي داله اسمها

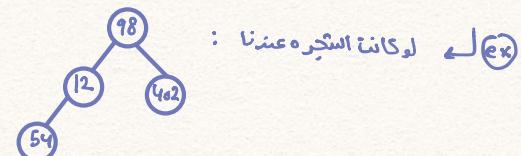
نفس هکره السؤال اللي قبل بس نطبق على جمهه اليمن

وشن تشوی الداله ٩٩

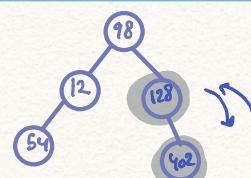
تأكد شيشين :

الـ `Node` اللي بي تضيف لها ولد يمين
← `Parent` ①
← القيمه اللي بي نحطها في الـ `Node` الكبير ②

وشن ترجع ؟ :

① موثر لـ `Node` الكبيره (إذا نجحت)② أو `NULL` لو نشلت أدا كان `Parent = NULL`③ إدا كان الـ `parent` عنده ولد يمين `Node` الكبيره تأذن كاه ولصيير ولديه `Node` الكبيره

لم سوينا هذه الكود :

`binary_tree_insert_right (root, 128);`

وشن فكره السؤال ٩٩ يعني نشيء داله اسمها :

Void binary_tree_delete (binary_tree_t *tree);

طبع وش مكتتها ٩٩

أنا تخذل سخري كامله يعني نمسح كل الـ Node الي فيها ونشيلها عن الماكرو

وشن مواهعات الداله ٩٩

١) تستقبل : tree وهو مؤشر للعنود الرئيسي الـ Node (Root) في الشجرة

٢) ترجع : ما زفج شئ ليش بـ ٩٦ نوعها (Void)

٣) إذا كانت الشجره فـ Main (للـ Main) ما نشيء شئ

طبع كيف تخذل الشجره ٩٩

بالعادة نستعمل Recursion (الاستدعاء الذاتي)

ليش بـ ٩٦ لأن لازم

٤) تخذل أول شئ أولاد العنود

٥) نبيين وختن العنود نفسه

لعني كيف المقرب بـ ٩٦

٦) روح يسار (أخذن الفرع اليسار)

٧) روح يمين (أخذن الفرع اليمين)

٨) نمسح العنود نفسه

Post-order-traversal

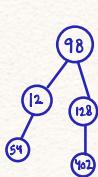
وهذه اسمونه

3. Delete

Score: 100.00% (Checks completed: 100.00%)

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



لو عندي شجره هزي : ← ← ex

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

لو سوينا :

```
binary_tree_delete(root);
```

يعني داشت نقول يا برتاج لشيكل كل العنود من الماكرو
ولاتخلي يعني نظف كل شئ

وشن فكره السؤال ٩٩ يعني نشيء داله اسمها :

Void binary_tree_delete (binary_tree_t *tree);

طبع وش مكتتها ٩٩

أنا تخذل سخري كامله يعني نمسح كل الـ Node الي فيها ونشيلها عن الماكرو

وشن مواهعات الداله ٩٩

١) تستقبل : tree وهو مؤشر للعنود الرئيسي الـ Node (Root) في الشجرة

٢) ترجع : ما زفج شئ ليش بـ ٩٦ نوعها (Void)

٣) إذا كانت الشجره فـ Main (للـ Main) ما نشيء شئ

طبع كيف تخذل الشجره ٩٩

بالعادة نستعمل Recursion (الاستدعاء الذاتي)

ليش بـ ٩٦ لأن لازم

٤) تخذل أول شئ أولاد العنود

٥) نبيين وختن العنود نفسه

لعني كيف المقرب بـ ٩٦

٦) روح يسار (أخذن الفرع اليسار)

٧) روح يمين (أخذن الفرع اليمين)

٨) نمسح العنود نفسه

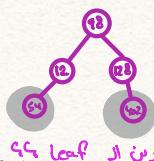
Post-order-traversal

4. Is leaf

Score: 100.00% (Checks completed: 100.00%)

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



دين اـ leaf ٩٦ ، ٥٤ ، ٤٠٢ يعني اـ leaf ٩٦

وشن فكره السؤال ٩٩ ذكيت داله اسمها :

```
int binary_tree_is_leaf (const binary_tree_t *node);
```

وشن فكره الداله ٩٩ تتحقق اذا الـ Node الي تطبيقها تعتبر اـ leaf

leaf يعني نود مالها اولاد لا يسار ولا يمين

وشن مواهعات الداله ٩٩

في كود اـ main.c

```
e = binary_tree_is_leaf(root);
ret = binary_tree_is_leaf (root->right);
ret = binary_tree_is_leaf (root->right->right);
```

١) تستقبل node وهو مؤشر لنود الي تخانثه منها

٢) ترجع 1 : اذا كانت leaf ٥٤
٠ : اذا هو leaf او NULL

كيف تتحقق اذا leaf ٩٦

```
if (node != NULL && node->left == NULL && node->right == NULL)
    return (1);
else
    return (0);
```

يعني اذا النود عديده واعنه ٥٤ او ٩٦ يعني اـ leaf ٥٤ و ٩٦

5. Is root

```
int binary_tree_is_root(const binary_tree_t *node);
```

Score: 100.00% (Checks completed: 100.00%)

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`

اللوك في main.c

1 ٩٩ Root \rightarrow $\text{Root} \neq \text{NULL}$ ← $\text{ret} = \text{binary_tree_is_root}(\text{root});$ ← (٤٩)
 ٥ ٩٩ Root \rightarrow $\text{Root} \neq \text{NULL}$ ← $\text{ret} = \text{binary_tree_is_root}(\text{root} \rightarrow \text{right});$ ← (٤٩)
 ٥ ٩٩ Root \rightarrow $\text{Root} \neq \text{NULL}$ ← $\text{ret} = \text{binary_tree_is_root}(\text{root} \rightarrow \text{right} \rightarrow \text{right});$ ← (٤٩)

وشن فكره السؤال ٤٦ : نكتب دالة تتحقق إذا التور اللى نعطيها هو اد Root في الشجرة ونرجع ١

Parent=NULL أو أول لوندى الشجرة يعني مالها أب

وشن فكره السؤال ٤٦ : Root ← داله لوندى



وشن فكره السؤال ٤٦ :
 تستقبل مؤشر لوند اللى بنى نورت هيل هي Root ←
 إذا كان Root (٤٩) ←
 إذا كان هو Root أو NULL (٤٩) ←

كين نتحقق إذا كانت Root

```
if(node != NULL && node->Parent == NULL)
    return(1);
else
    return(0);
```

يعنى إذا استود موجوده ومامعده Parent رجع ١ ومناها هي Root

6. Pre-order traversal

mandatory

Score: 100.00% (Checks completed: 100.00%)

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>
```

اللوك في main.c

Void Print_num (int n) ← (٤٩)
 ↓

Printf ("%d\n", n);

اللوك Print_num هي الـ راجع كل عنصر فى لوند تزوره الـ

النتائج او الـ output

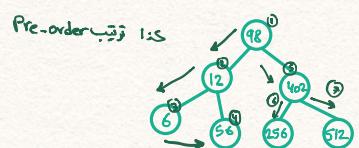
راجع كل عنصر حسب الترتيب المذكور

وشن فكره السؤال ٤٦ : نكتب دالة شنوى Pre-order traversalBinary Tree : أسم الـ

Void binary_tree_Preorder (const binary_tree_t *tree, void(*func)(int));

وشن فكره السؤال ٤٦ :
 في Pre-order traversalBinary Tree ، يعني نمشى على لوندات بالترتيب التالي:

١) فوج لوند كل فى نوع للسار فى المينى



كين بتكون على شكل كود ٤٦

```
Void binary_tree_Preorder (const binary_tree_t *tree, void(*func)(int))
{
    if (tree == NULL || func == NULL)
        return;
    func(tree->n);
    binary_tree_Preorder(tree->left, func);
    binary_tree_Preorder(tree->right, func);
}
```

7. In-order traversal

Score: 100.00% (Checks completed: 100.00%)

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

`main.c` الكود في

```
Void Print_num(int n)
{
    printf("%d\n", n);
}

الناتج ذو الـ output
راح يطبع حسب الترتيب المذكور
```

السؤال وين فكرته ٩٩ نكتب دالة نقشى فى اد ترتيب باستخدام Tree

Void binary_tree_inorder (const binary_tree_t *tree, void (*func)(int));

٩٩ In Order Traversal وشن ليني

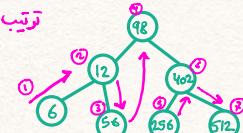
في اد ترتيب على الشجرة بالترتيب التالي

① فرج لليسار اول Left

② العدين فرج للشود اليمين Right

③ بدين فرج لليمين Right

In-order Travers ترتيب اد



كذلك يكون على شكل كود ٩٩

Void binary_tree_inorder (const binary_tree_t *tree, Void (*func)(int))

```
{
    if (tree == NULL || func == NULL)
        return;
```

```
binary_tree_inorder(tree->left, func);
func(tree->n);
binary_tree_inorder(tree->right, func);
```

ما هو الذي يتحقق `func==NULL` او `tree==NULL` معاً

وتشتمل (الكار) مثناة نقشى في الشجرة بغيره منظمه

8. Post-order traversal

Score: 100.00% (Checks completed: 100.00%)

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@tmn/hinary_trees\$ cat 8-main.c

`main.c` الكود في

```
Void Print_num(int n)
{
    printf("%d\n", n);
}

الناتج ذو الـ output
راح يطبع حسب الترتيب المذكور
```

وشن ذكره السؤال ٩٩ نكتب دالة نقشى على

Post-order traversal باستخدام

Void binary_tree_Postorder (const binary_tree_t *tree, Void (*func)(int));

٩٩ Post-order traversal وشن ليني

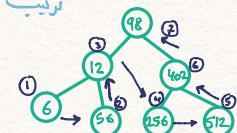
فرج لشود اليمين بعد عازف اليسار واليمين :

① اول ستي فرج للجهة اليسار Left

② بده فرج للجهة اليمين Right

وأخير فرج لشود الاليه

Post-order traversal ترتيب اد



كذلك يكون على شكل كود ٩٩

Void binary_tree_Postorder (const binary_tree_t *tree, Void (*func)(int))

```
{
    if (tree == NULL || func == NULL)
        return;
```

```
binary_tree_Postorder(tree->left, func);
binary_tree_Postorder(tree->right, func);
```

```
func(tree->n);
```

٣

9. Height

Score: 100.00% (Checks completed: 100.00%)

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`

شوي داله تقىس ارتفاع Binary Tree

و لش فتكه المسوال ٤٩

شكل الدالة: `size_t binary_tree_height(const binary_tree_t *tree);`

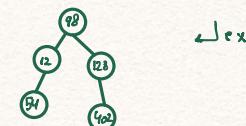
فهاد دلته ميم ٤٩
* هو مؤشر إلى المورث الروت (root node)

* إذا كانت `tree == NULL` راج ترجم لنا ٥ صفر

* لا تحسب عدد المورث (leaf node) من `Root` إلى أبعد ورقة (edges)

ولست هو ارتفاع الشجرة ٤٩

هو عدد الـ edges (أي عدد ورقة) من `Root` اللي هي أبعد ورقة
 وليس `node`



من ٩٨ إلى ٥٤: ٩٨ → ١٢ → ٩٤ → ٥٤ اكانت ٢ → الأرتفاع = ٢

من ٩٨ إلى ٤٠٢: ٩٨ → ١٢ → ١٢٨ → ٤٠٢ اكانت ٢ → الأرتفاع = ٢

كيف يتم تفخيم الدالة ٤٩

① إذا كانت `tree == NULL` أو كانت يدون أبنائ المورث = ٠

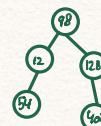
② نحسب ارتفاع الفرع الأيسر (`left_height`) والفرع اليميني (`right_height`) باستعمال أسلوب الدالة `recursion`

③ نرجع اليهم الأكبر بين المؤشر ١ +

```

size_t binary_tree_height(const binary_tree_t *tree)
{
    size_t left, right;
    if (tree == NULL || (tree->left == NULL && tree->right == NULL))
        return (0);
    left = binary_tree_height(tree->left);
    right = binary_tree_height(tree->right);
    return (left > right ? left : right) + 1;
}
  
```

程式 `main.c` في مثال



height from 98 : 2 → 2 من ٩٨ إلى أقصى ورقة :

height from 128 : 1 → 1 من ١٢٨ إلى ٤٠٢ :

height from 54:0 → 0 من ٥٤ (ورقة) لا يوجد أبناء :

• 10 - binary_tree_depth

```
#include "binary_trees.h"
header file
/**
 * binary_tree_depth - function that measures the depth of a node in,
 * a binary tree.
 * @tree: pointer to the node to measure the depth.
 *
 * Return: depth of a node in a binary tree.
 * If tree is NULL,
 * your function must return 0.
 */
size_t binary_tree_depth(const binary_tree_t *tree)
{
    size_t depth = 0;
    binary_tree_t *node = NULL;
    if (tree != NULL)
    {
        node = tree->parent;
        while (node != NULL)
        {
            depth++;
            node = node->parent;
        }
    }
    return (depth);
}
```

تعريف `binary_tree_t` اسدي نوعه `node`
حيث انه يكون `NULL` في البداية

شرط يتأكد `input node is not NULL`

في `while loop` تستمر ما دام ان `parent` هو فارغ

في النهاية يتراجع `depth`
وهو يمثل عمق الـ `node`

تعريف متغير `depth` من نوع `size_t` قيمته من 0

اذا `node` موجود ، يربط ابوها في متغير اسمه `parent node`

كل مره تتنفذ `loop` الـ `depth` يزداد بقيمة 1

كل مره تتنفذ `loop` نحدث `parent node` عشان يعبر ابو `node` الحالي

• 11 - binary_size

```
#include "binary_trees.h"

/**
 * binary_tree_size - function that measures the size of a binary tree.
 * @tree: pointer to the root node of the tree to measure the size.
 *
 * Return: size of a binary tree. If tree is NULL,
 * your function must return 0.
 */
size_t binary_tree_size(const binary_tree_t *tree)
{
    size_t left, right;
    if (tree == NULL) → تعيين متغير nodes لـ tree فاقيه ترجع 0
    return (0);
    left = binary_tree_size(tree->left); → في النهاية يتراجع عدد nodes في اليسار
    right = binary_tree_size(tree->right); → + عدد nodes في العالي
    return (left + 1 + right); → + عدد nodes في اليمين
}
```

نادي `funcion (function)` ، تعيين عدد `nodes`
على الجهة اليسار من `node` العالي ونفس الشي لليمين

• 12 - binary-tree-leaves

```
#include "binary_trees.h"

/** (nodes with no children) leaves تحسب عدد الـ leaves
 * binary_tree_leaves - function that counts the leaves in a binary tree.
 * @tree: pointer to the root tree of the tree to count the no. of leaves.
 *
 * Return: number of leaves. If tree is NULL the function must return 0.
 */
size_t binary_tree_leaves(const root node pointer binary_tree_t *tree)
{
    size_t left, right;
    if (tree == NULL) اذا كانت الـ tree فانية ترجع 0
        return (0);
    if (tree->left == NULL && tree->right == NULL) على يسار ويبين الـ right
        return (1);
    left = binary_tree_leaves(tree->left); على الجهة اليسار يدعا اليدين recursive call
    right = binary_tree_leaves(tree->right); على الجهة اليمين
    return (left + right);
}

تعريف متغيرين يخزن فهم عدد subtrees على يسار ويبين الـ right
اذا ما عندها left & right يعني تختبر leaf ا
فرجع 1
the sum of left and right leaves نرجع
```

• 13 - binary-tree-nodes

```
#include "binary_trees.h"

/** node is a leaf دالة تتأكد ان
 * binary_tree_is_leaf - function that checks if a node is a leaf.
 * @node: pointer to the node to check.
 *
 * Return: 1 if node is a leaf, and 0 otherwise. If node is NULL, return 0.
 */
int binary_tree_is_leaf(const binary_tree_t *node)
{
    if (node != NULL && node->left == NULL && node->right == NULL) اذا كانت الـ left & right children both NULL go node == 0 يعني هذى الـ node يختبر leaf
        return (1);
    return (0);
}

نحسب عدد الـ nodes الى عندها على الأقل one child
size_t binary_tree_nodes(const one child binary_tree_t *tree)
{
    size_t left, right;
    if (tree == NULL || binary_tree_is_leaf(tree)) اذا كانت الـ tree او الـ node المالي leaf ترجع 0 لأن الحقد اللي ماعندها أولاد ها نحسبها
        return (0);
    left = binary_tree_nodes(tree->left);
    right = binary_tree_nodes(tree->right);
    return (1 + left + right);
}

تعريف متغيرين يخزن فهم عدد nodes في جهة اليسار واليمين
الجهة اليسار يدعا اليدين recursive calls
نرجع 1 الى node الحالي لأن عندها واحد على الأقل children node في الجهة اليسار واليمين
+ عدد الـ nodes في الجهة اليسار واليمين
```

• |4 - binary-tree_balance

```
#include "binary_trees.h"
تحسب ارتفاع tree في node
/** tree: pointer to the root node of a binary tree.
 * node_height - function that computes the height of a binary tree node.
 * @tree: pointer to the root node of the tree to measure the height.
 * @n: accumulated height.
 * @height: pointer to the maximum height in the node's tree.
 */
int node_height(const binary_tree_t *tree,
int n, int *height)
{
    if (tree == NULL)
        return 0;
    if ((tree->left == NULL) && (tree->right == NULL))
        if (*height < n)
            *height = n;
    else
        node_height(tree->left, n + 1, height);
        node_height(tree->right, n + 1, height);
    }
}
تحسب عامل التوازن
balance factor
binary_tree_balance - function that measures the balance factor of a binary tree.
* @tree: pointer to the root node of the tree to measure the balance factor.
* @height: pointer to the maximum height in the node's tree.
* @Return: balance factor of a binary tree. If tree is NULL, return 0.
int binary_tree_balance(const binary_tree_t *tree)
{
    int balance_factor = 0, left_height = 0, right_height = 0;
    if (tree == NULL)
        return 0;
    if (node_height(tree->left, 1, &left_height) <
        node_height(tree->right, 1, &right_height))
        balance_factor = left_height - right_height;
    return balance_factor;
}
```

إذا كان الارتفاع n أكبر من أقصى ارتفاع وهل له $height$
هذا يدل أننا اكتشفنا مسار اطول في الفجوة
فنجحت قيمة $height$ عشان تساوي n

إذا الدالة leaf كانت كلها ، نستدعي الدالة مع ثانية (recursion)
على الجهة اليسرى والجهة ونزيد الارتفاع بمقدار 1

إذا الدالة كانت NULL ترجع 0 لأن ما فيه شجرة نقيعها

نستدعي الدالة الجصتين عشان نحسب ارتفاعهما

نرج العامل
الي حسنه

تعريف متغيرات:
عامل التوازن . الارتفاع في اليسار . الارتفاع في اليمين

عامل التوازن = الارتفاع الأيسر - الارتفاع الآخرين

أيضاً
نرج العامل
الي حسنه

• |5 - binary-tree_is_full

```
#include "binary_trees.h"
تحقق اذا كانت tree ممتلئة
* binary_tree_is_full - function that checks if a binary tree is full.
* @tree: pointer to the root node of the tree to check.
* @Return: 1 if tree is full. 0 otherwise.
int binary_tree_is_full(const binary_tree_t *tree)
{
    int left, right;
    if (tree == NULL)
        return 0;
    if ((tree->left == NULL) && (tree->right == NULL))
        return 1;
    if ((tree->left) && (tree->right))
    {
        left = binary_tree_is_full(tree->left);
        right = binary_tree_is_full(tree->right);
        return (left && right);
    }
    return 0;
}
```

لو عندما left and right children
عندهما 1 child إذا tree ترجع 0

إذا الجھتين ممتلئتين ترجع 1

إذا الجھتين ممتلئتين نرج 1

إذا الجھتين ممتلئتين نرج 0

• |6 - binary-tree-is-perfect

```
#include "binary_trees.h"
/** تحسب معلومات عن الـ tree
 * @tree: pointer to the root node of the tree to check.
 * @leaves: pointer to the tree's leaf count value.
 * @height: pointer to the tree's maximum height value.
 */
void tree_stats(const binary_tree_t *tree, size_t n,
size_t *leaves, size_t *height)
{
    if (tree == NULL)
        if ((tree->left == NULL) && (tree->right == NULL))
            if (*leaves == NULL)
                if ((*height == NULL) && (n > *height))
                    *height = n;
            else
                tree_stats(tree->left, n + 1, leaves, height);
                tree_stats(tree->right, n + 1, leaves, height);
        }
    }
}
/** binary_tree_is_perfect - function that checks if a binary tree is perfect.
 * @tree: pointer to the root node of the tree to check.
 */
int binary_tree_is_perfect(const binary_tree_t *tree)
{
    size_t leaves_count = 0;
    size_t tree_height = 0;
    tree_stats(tree, &leaves_count, &tree_height);
    return ((leaves_count == (1 << tree_height) ? 1 : 0));
}
```

عدد الـ leaves ارتفاعها

تحدد الـ height اذا ومتى لعن اطول

نؤكد اذا الشجرة كامله او لا

نستخدم الـ helper function عثمان بحروف

كم leaf وكم ارتفاع

نحوت اذا العدد من مفر

ترجع اذا ترجع 0

number of leaves = 2^{height}

• |7 - binary-tree-Sibling

```
#include "binary_trees.h"
/** ترجع الأخ ( sibling ) لـ node الذي يعطيها
 * binary_tree_sibling - function that finds the sibling of a node.
 * @node: pointer to the node to find the sibling.
 */
binary_tree_t *binary_tree_sibling(binary_tree_t *node)
{
    binary_tree_t *left = NULL;
    binary_tree_t *right = NULL;
    if ((node != NULL) && (node->parent != NULL)) ←
        left = node->parent->left;
        right = node->parent->right; ←
    return (left == node ? right : left);
}
```

نؤكد ان الـ node موجوده وعندما نخزن

نخزن الـ children حفين الـ parent node

فيم الـ left and right children

عرفنا هذين وظيفتين وظيفات NULL وعثمان نخزن

اذا كانت الـ node يسار ، ترجع الين

واما كانت يمين ترجع اليسار ، يعني ترجع الأخ

* هنا ترجع Null

اذا الـ node ما عندها parent او ما عندها sibling

• ١٨-binary-tree-uncle

```

#include "binary_trees.h"

/**
 * binary_tree_uncle - function that finds the uncle of a node.
 * @node: pointer to the node to find the uncle.
 *
 * If node is NULL or the parent is NULL, return NULL.
 * If node has no sibling, return NULL.
 *
 * Return: pointer to the uncle node.
 */
binary_tree_t *binary_tree_uncle(binary_tree_t *node)
{
    binary_tree_t *left = NULL;
    binary_tree_t *right = NULL;
    binary_tree_t *parent = (node != NULL ? node->parent : NULL);
    if ((parent != NULL) && (parent->parent != NULL)) {
        left = parent->parent->left;
        right = parent->parent->right;
    }
    return (left == parent ? right : left);
}

```

الأب وأخوه
↑
عرفنا مؤشرين نحط فيهم أبناء الـ parent
left and right children of the grandparent

نخزن الـ children معندهم العد
أباً وآخر أباً

إذا كان الـ parent يسار نرجع العين
إذا كان الـ parent يمين نرجع اليمين

نتأكد أن عندنا 2 parent عشان تقدر تجين أخوه
إذا الـ node موجوده نخزن الـ parent حقها
إذا ما فيه نعطي الـ parent ، parent
grandparent

Danah Alshehri

C - Sorting algorithms & Big O

وين فكره السؤال ٤٩ نكتب داله تقوم بترتيب عناصر المصفوفه أعداد مسيه (int)

ترتيب تصاعدي باستخان خوارزميه

تعريف الداله:

Void bubble_Sort(int* array, size_t size);

وشي يعني Bubble Sort ٤٩ هي خوارزميه بسيطه جي تختلف كالتالي :

١ تمر على المصفوفه أكثر من مره

٢ كل مره تقارن عنصرين جب يبغ

٣ إذا كان العنصر الأول أكبر من الثاني يتم تبديل (Swap)

٤ تدور الحلقة لين ما يصير ترتيبهم صحيح

لوعندنا مصفوفه ex [4, 8, 3, 5]

العنصر الأول أكبر من الثاني شووي تبدل 3 < 5 . أخطاء جوله:

٥ 8 > 5 ماسنوي شوي

٦ 4 < 8 العنصر أصغر من الثاني شووي تبدل [8, 4, 5, 3] .

٧ 5 > 3 ماتحتاج ٨ < 5 العنصر أكبر من الثاني شووي تبدل [8, 5, 4, 3] .

٩ ثالث جوله: خالص هرتبه ما في تبديلات

0. Bubble sort

Score: 100.00% (Checks completed: 100.00%)

mandatory



Write a function that sorts an array of integers in ascending order using the **Bubble sort** algorithm

- Prototype: `void bubble_sort(int *array, size_t size);`
- You're expected to print the `array` after each time you swap two elements (See example below)

Write in the file `0-0.c`, the big O notations of the time complexity of the Bubble sort algorithm, with 1 notation per line:

- in the best case
- in the average case
- in the worst case

Best case

$O(n)$

في ملف ٠.٠ نكتب :

Average case $\rightarrow O(n^2)$

وهو يمثل او Time Complexity

Worst case $\rightarrow O(n^2)$

طيب لين ذكرت كذا بـ ٤٩

سوأ حاله (worst): لو المصفوفه مرتب بالعكس ، يحتاج n^2 خطوه

أفضل حاله (best): لو المصفوفه مرتب اصلا ، لغز عليهما مره واحدة فقط $O(n)$

لودخنا هذى الازقان

19, 48, 99, 71, 13, 52, 96, 73, 86, 7

ex

ال output يكون :

7, 13, 19, 48, 52, 71, 73, 86, 96, 99

1. Insertion sort

Score: 100.00% (checks completed: 100.00%)

Write a function that sorts a doubly linked list of integers in ascending order using the **Insertion sort** algorithm

- Prototype: `void insertion_sort_list(listint_t **list);`
- You are not allowed to modify the integer `n` of a node. You have to swap the nodes themselves.
- You're expected to print the `list` after each time you swap two elements (See example below)

Write in the file `1_0.c`, the big O notations of the time complexity of the Insertion sort algorithm, with 1 notation per line:

- in the best case
- in the average case
- in the worst case

$$\begin{array}{l} \text{Best Case} \rightarrow O(n) : \text{القائمة موجودة} \\ \text{Average Case} \rightarrow O(n^2) : \text{متضمنة} \\ \text{Worst Case} \rightarrow O(n^2) : \text{مرتبة عكسية} \end{array}$$

في ملحوظة: O-1

نكتب دالة تقوم بترتيب عناصر `double linked list` من الأرداد الصحيحة بترتيب تصاعدي باستخدام خوارزمية `insertion sort`

وشن مذكرة السؤال ٦٩

`Void insertion_Sort_list (listint_t **list);`

وشن مذكرة الماء بـ ٦٩

هونج يسئل التود في `double linked list` (غالباً معنى في `list`) .

مايسح لله تخسر فتحة `n` داخل أي لود . بدل من ذلك يجب تبديل مواقع النود

(Swap nodes, not value)

يجب هباء القائمة بعد كل Swap يتم بين لودين

طلب وشن مبدأ عمل `insertion sort` على `Linked List` بـ ٦٩

١) بناءً من العنصر الثاني في القائمة (رئي العنصر الأول يعتبر مقروز تلقائياً)

٢) نقارنه بالعنصر الذي قبله

٣) إذا كان أصغر، نقوم بتحريكه للخلف حتى يفتح له مكان الصحيح

٤) نكرر هذه العملية لكل دسار حتى نتم إيه القائمة

٥) دايم طبعه بعد كل عملية Swap

19, 48, 99, 71, 13, 52,
 ↓
 19, 48, 71, 99, 13,
 ↓
 19, 48, 71, 13, 99,
 ↓

 7, 13, 19, 48,

ملاحظة: ← يتم تدوين النود وليس العنصر داخلها

• عليه تبديل في `Double linked list` تكون أسهل من `list` لأن كل نود يملك مؤشر في النود السابق `prev` والنود التالي `next`

• طباعة القائمة بعد كل Swap تسهل تتبع خطوات المترز .

LEEN ALSALEH

2. Selection Sort

- 2 - 0

$O(n^2)$	Worst-case
$O(n^2)$	Best-case
$O(n^2)$	Average-case

- 2 - selection - sort.c

```
#include "sort.h"                                ترتيب الأرقام داخل الـ array
/**                                                 ترتيب الأرقام داخل الـ array
 * selection_sort - function that sorts an array of integers in ascending
 * order using the Selection sort algorithm
 * @size: size of the array
 * @array: list with numbers
 */
void selection_sort(int *array, size_t size)
{
    size_t i, index;
    int tmp, swap, flag = 0;
    if (array == NULL) [ ] ← إذا كانت لا array فما يه نطلع من الـ function
        return; [ ] ←
    for (i = 0; i < size; i++) ← تمر على كل element outer loop
    {
        tmp = i; [ ] ←
        flag = 0; [ ] ←
        for (index = i + 1; index < size; index++) ← نفترض ان الـ element هو الـ max ونغير الـ flag
        {
            if (array[tmp] > array[index]) ← (ما فيه تبديل الجين)
            {
                tmp = index; ←
                flag += 1; ←
            }
        }
        if (flag != 0) ←
        {
            swap = array[i]; ←
            array[i] = array[tmp]; ←
            array[tmp] = swap; ←
            if (flag != 0) ←
                print_array(array, size); ←
        }
    }
}
```

Danah Alshehri