OLD Quiz 6 Version H

Due No due date **Points** 8 **Questions** 3 **Available** after Dec 3 at 1pm
Time Limit None **Allowed Attempts** Unlimited

Instructions

Quiz 6: Binary Search Trees

7 points required to pass

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	less than 1 minute	0 out of 8 *

* Some questions not yet graded

Score for this attempt: **0** out of 8 *
Submitted Dec 5 at 2:14pm
This attempt took less than 1 minute.

Question 1

Not yet graded / 2 pts

Suppose we have this structure definition for a node in a binary search tree (BST) of **int**:

struct bt_node { int value; bt_node *left; bt_node *right; };

The header node in a calling function is of type pointer-to **bt_node**. A header node of **nullptr** (or **0**) indicates an empty tree. The ordering rule for the BST is that all values less than the current node's value are to the left, and all values greater are to the right. Duplicate values are ***not allowed***.

Write the implementation of this function that returns the ***maximum value*** stored in a binary search tree, or **0** if the tree is empty:

```
int binary_tree_max(const bt_node *top)
{
    // this is the code you have to write
}
```

Your Answer:

fdas

```
// this is a better way
if (!top)
    return 0;
for ( ; top->right; top = top->right)
    ;
return top->value;


// this original code that I wrote works for a BST, but is needlessly complicated: it is
// what you would need to do for a linked list
if (top == nullptr)
    return 0;
int max{top->value};   // curly brace form of initialization
for (top = top->right ; top; top = top->right)
    if (top->value > max)
        max = top->value;
return max;
```

Question 2

Not yet graded / 3 pts

Suppose we have this structure definition for a node in a binary search tree (BST) of **int**:

struct bt_node { int value; bt_node *left; bt_node *right; };

The header node in a calling function is of type pointer-to **bt_node**. A header node of **nullptr** (or **0**) indicates an empty tree. The ordering rule for the BST is that all values less than the current node's value are to the left, and all values greater are to the right. Duplicate values are ***not allowed***.

Write the implementation of this function, that returns the ***depth*** of the tree, that is, the number of nodes in the longest branch downward from the top. An empty tree has a depth of **0**. A tree with two nodes must have a depth of 2. A tree with three nodes has either a depth of 2 (if the top node has one left and one right child) or a depth of 3 (if the top node has only one child, and that child has the third node as its child), and so forth.

```
size_t binary_tree_depth(const bt_node *top)
{
    // this is the code you have to write
}
```

Your Answer:

```
if (top == nullptr)
    return 0;
size_t left_depth = binary_tree_depth(top->left);
size_t right_depth = binary_tree_depth(top->right);
if (left_depth > right_depth)
    return 1 + left_depth;
else
    return 1 + right_depth;
```

Question 3

Not yet graded / 3 pts

Suppose we have this structure definition for a node in a binary search tree (BST) of **int**:

struct bt_node { int value; bt_node *left; bt_node *right };

The header node in a calling function is of type pointer-to **bt_node**. A header node of **nullptr** (or **0**) indicates an empty tree. The ordering rule for the BST is that all values less than the current node's value are to the left, and all values greater are to the right. Duplicate values are ***not allowed***.

Write the implementation of this function, that inserts **val** into a binary search tree, unless **val** already exists in the tree:

```
void binary_tree_insert(bt_node **ptop, int val)
{
    // this is the code you have to write
}
```

Your Answer:

```
if (*ptop == nullptr)
    *ptop = new bt_node{ val, nullptr, nullptr};
else {
    if (val < (*ptop)->value)
        binary_tree_insert(&(*ptop)->left, val);
    else if (val > (*ptop)->value)
        binary_tree_insert(&(*ptop)->right, val);
}
```

Quiz Score: **0** out of 8