

Data Structures and Algorithms

Assignment 3: Binary Tree

Information:

Program templates for questions 1-4 are given as separated files (Q1_template_BT.c, Q2_template_BT.c, Q3_template_BT.c, and Q4_template_BT.c). You must use them to implement your functions. The program contains a *main()* function, which includes a switch statement to execute different functions that you should implement. You need to submit your code to the NTULearn (State your name and your lab group in your submission). Deadline for program submission: **September 21st, 2021 (Tuesday) 11.59 pm.**

Assignment Questions (1-4)

1. **(identical)** Write a recursive C function `identical()` to determine whether two binary trees are structurally identical, assuming the two binary trees as `tree1` and `tree2`. This function returns 1 if two binary trees are structurally identical; otherwise, it returns 0. Note that two binary trees are structurally identical if they are both empty or if they are both non-empty and the left and the right subtrees are similar (**they are made of nodes with the same values and arranged in the same way**).

The function prototype is given as follows:

```
int identical(BTNode *tree1, BTNode *tree2);
```

For example, if the given two trees are tree 1 (1, 3, 2, 5, 4, 7, 8) and tree 2 (1, 3, 2, 5, 4, 7, 8), as shown in Figure 1, then, tree 1 and tree 2 are **structurally identical**.

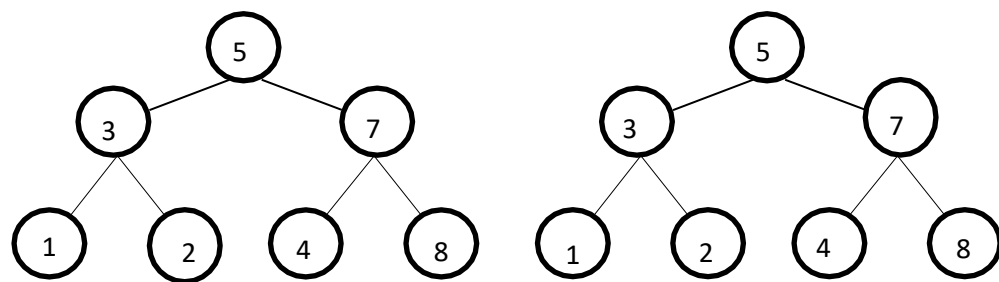


Figure 1: tree1 (left) and tree 2 (right)

A sample input and output session is given below:

```
1: Create a binary tree1.
2: Create a binary tree2.
3: Check whether two trees are structurally identical.
0: Quit;
```

```
Please input your choice(1/2/3/0): 1
```

```
Creating tree1:
```

```
Input an integer that you want to add to the binary tree. Any
Alpha value will be treated as NULL.
```

```
Enter an integer value for the root: 5
Enter an integer value for the Left child of 5: 3
Enter an integer value for the Right child of 5: 7
Enter an integer value for the Left child of 3: 1
Enter an integer value for the Right child of 3: 2
Enter an integer value for the Left child of 1: a
Enter an integer value for the Right child of 1: a
Enter an integer value for the Left child of 2: a
Enter an integer value for the Right child of 2: a
Enter an integer value for the Left child of 7: 4
Enter an integer value for the Right child of 7: 8
Enter an integer value for the Left child of 4: a
Enter an integer value for the Right child of 4: a
```

```

Enter an integer value for the Left child of 8: a
Enter an integer value for the Right child of 8: a
The resulting tree1 is: 1 3 2 5 4 7 8
Please input your choice(1/2/3/0): 2

```

Creating tree2:

Input an integer that you want to add to the binary tree. Any Alpha value will be treated as NULL.

```

Enter an integer value for the root: 5
Enter an integer value for the Left child of 5: 3
Enter an integer value for the Right child of 5: 7
Enter an integer value for the Left child of 3: 1
Enter an integer value for the Right child of 3: 2
Enter an integer value for the Left child of 1: a
Enter an integer value for the Right child of 1: a
Enter an integer value for the Left child of 2: a
Enter an integer value for the Right child of 2: a
Enter an integer value for the Left child of 7: 4
Enter an integer value for the Right child of 7: 8
Enter an integer value for the Left child of 4: a
Enter an integer value for the Right child of 4: a
Enter an integer value for the Left child of 8: a
Enter an integer value for the Right child of 8: a
The resulting tree2 is: 1 3 2 5 4 7 8

```

```

Please input your choice(1/2/3/0): 3
Both trees are structurally identical.

```

```

Please input your choice(1/2/3/0): 0

```

2. **(maxHeight)** Write a C function `maxHeight()` that accepts a pointer to the root node of a binary tree and return the number of links along the longest path from the root node down to the farthest leaf node. Note that height of a given node is equal to the number of links from that node to the deepest leaf node. (Hint: **Consider that the height of an empty tree is -1**).

The function prototype is given as follows:

```
int maxHeight(BTNode *root)
```

For example, for the given binary tree (1, 2, 3, 4, 5, 6, 7) in Figure 2, the maximum height is 2.

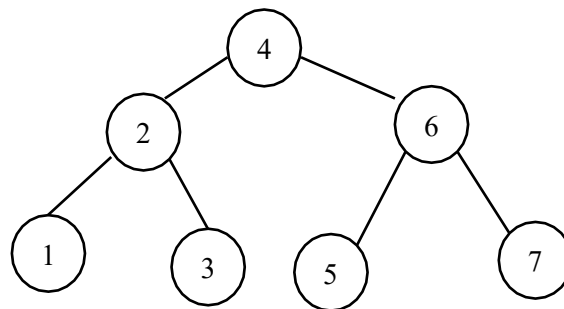


Figure 2: Binary tree

A sample input and output session is given below:

```

1: Create a binary tree.
2: Find the maximum height of the binary tree.
0: Quit;

```

```

Please input your choice(1/2/0): 1
Input an integer that you want to add to the binary tree. Any
Alpha value will be treated as NULL.

```

```

Enter an integer value for the root: 4

```

```

Enter an integer value for the Left child of 4: 2
Enter an integer value for the Right child of 4: 6
Enter an integer value for the Left child of 2: 1
Enter an integer value for the Right child of 2: 3

```

```

Enter an integer value for the Left child of 1: a
Enter an integer value for the Right child of 1: a
Enter an integer value for the Left child of 3: a
Enter an integer value for the Right child of 3: a

```

```

Enter an integer value for the Left child of 6: 5
Enter an integer value for the Right child of 6: 7

```

```

Enter an integer value for the Left child of 5: a
Enter an integer value for the Right child of 5: a
Enter an integer value for the Left child of 7: a
Enter an integer value for the Right child of 7: a

```

The resulting binary tree is: 1 2 3 4 5 6 7

```

Please input your choice(1/2/0): 2
The maximum height of the binary tree is: 2

```

```

Please input your choice(1/2/0): 0

```

3. (**countOneChildNodes**) Write a C function `countOneChildNodes()` that accepts a pointer to the root node of a binary tree and return the number of nodes that have exactly one child node.

The function prototype is given as follows:

```
int countOneChildNodes(BTNode *root)
```

For example, consider the given binary tree (10, 20, 55, 30, 50, 60, 80) in Figure 3. In the binary tree, the number of nodes that have exactly one child node is 2, and those two nodes are colored in Red.

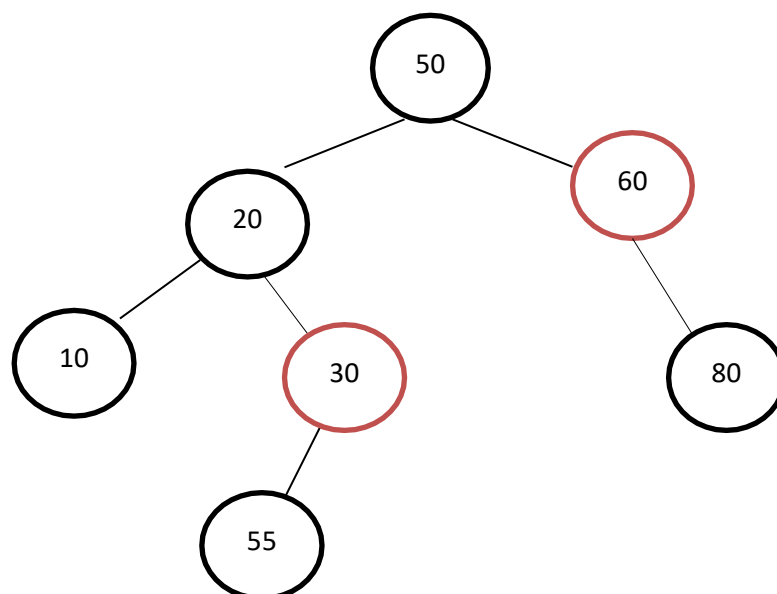


Figure 3: Binary tree

Some sample inputs and outputs sessions are given below:

```
1: Create a binary tree.
2: Count the number of nodes that have exactly one child node.
0: Quit;
```

Please input your choice(1/2/0): 1

Input an integer that you want to add to the binary tree. Any Alpha value will be treated as NULL.

Enter an integer value for the root: 50

Enter an integer value for the Left child of 50: 20

Enter an integer value for the Right child of 50: 60

Enter an integer value for the Left child of 20: 10

Enter an integer value for the Right child of 20: 30

Enter an integer value for the Left child of 10: a

Enter an integer value for the Right child of 10: a

Enter an integer value for the Left child of 30: 55

Enter an integer value for the Right child of 30: a

Enter an integer value for the Left child of 55: a

Enter an integer value for the Right child of 55: a

Enter an integer value for the Left child of 60: a

Enter an integer value for the Right child of 60: 80

Enter an integer value for the Left child of 80: a

Enter an integer value for the Right child of 80: a

The resulting binary tree is: 10 20 55 30 50 60 80

Please input your choice(1/2/0): 2

The Number of nodes that have exactly one child node is: 2

Please input your choice(1/2/0): 0

4. **(sumOfOddNodes)** Write a recursive C function `sumOfOddNodes()` that accepts a pointer to the root node of a binary tree of integers and return the sum of all odd numbers in the tree.

The function prototypes are given as follows:

```
int sumOfOddNodes(BTNode *root);
```

For example, for the given binary tree (11, 40, 35, 50, 80, 60, 85) in Figure 4, the sum of all the odd values is **131**. The odd values are colored in **Red**.

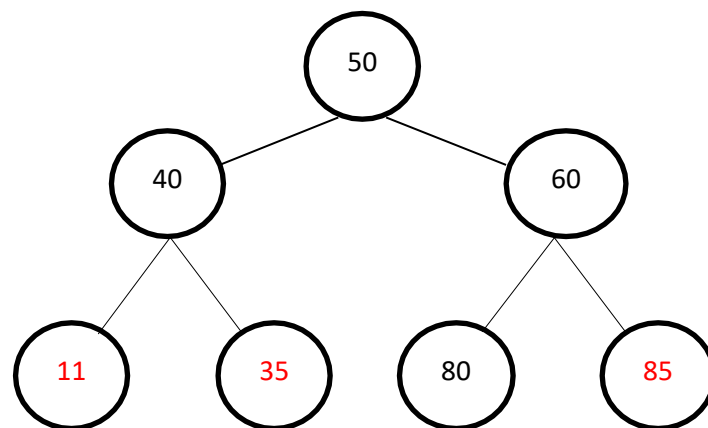


Figure 4: Binary tree

Some sample inputs and outputs sessions are given below:

1: Create a binary tree.

2: Find the sum of all odd numbers in the binary tree.

0: Quit;

Please input your choice(1/2/0): 1

Input an integer that you want to add to the binary tree. Any Alpha value will be treated as NULL.

Enter an integer value for the root: 50

Enter an integer value for the Left child of 50: 40

Enter an integer value for the Right child of 50: 60

Enter an integer value for the Left child of 40: 11

Enter an integer value for the Right child of 40: 35

Enter an integer value for the Left child of 11: a

Enter an integer value for the Right child of 11: a

Enter an integer value for the Left child of 35: a

Enter an integer value for the Right child of 35: a

Enter an integer value for the Left child of 60: 80

Enter an integer value for the Right child of 60: 85

Enter an integer value for the Left child of 80: b

Enter an integer value for the Right child of 80: b

Enter an integer value for the Left child of 85: b

Enter an integer value for the Right child of 85: b

The resulting binary tree is: 11 40 35 50 80 60 85

Please input your choice(1/2/0): 2

The sum of all odd numbers in the binary tree is: 131.

Please input your choice(1/2/0):

choice(1/2/0): 0