Write a Program to Find the Maximum Depth or Height of a Tree

Given a binary tree, find height of it. Height of empty tree is 0 and height of below tree is 3.



*Example Tree*

[**Recommended: Please solve it on “*PRACTICE*” first, before moving on to the solution.**](https://practice.geeksforgeeks.org/problems/height-of-binary-tree/1)

Recursively calculate height of left and right subtrees of a node and assign height to the node as max of the heights of two children plus 1. See below pseudo code and program for details.

**Algorithm:**

maxDepth()

1. If tree is empty then return 0

2. Else

(a) Get the max depth of left subtree recursively i.e.,

call maxDepth( tree->left-subtree)

(a) Get the max depth of right subtree recursively i.e.,

call maxDepth( tree->right-subtree)

(c) Get the max of max depths of left and right

subtrees and add 1 to it for the current node.

max\_depth = max(max dept of left subtree,

max depth of right subtree)

+ 1

(d) Return max\_depth

**See the below diagram for more clarity about execution of the recursive function maxDepth() for above example tree.**

maxDepth('1') = max(maxDepth('2'), maxDepth('3')) + 1

= 2 + 1

/ \

/ \

/ \

/ \

/ \

maxDepth('2') = 1 maxDepth('3') = 1

= max(maxDepth('4'), maxDepth('5')) + 1

= 1 + 1 = 2

/ \

/ \

/ \

/ \

/ \

maxDepth('4') = 1 maxDepth('5') = 1

**Implementation:**

* C++
* C
* Java
* Python3
* C#

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>      /\* A binary tree node has data, pointer to left child     and a pointer to right child \*/  struct node  {      int data;      struct node\* left;      struct node\* right;  };    /\* Compute the "maxDepth" of a tree -- the number of      nodes along the longest path from the root node      down to the farthest leaf node.\*/  int maxDepth(struct node\* node)  {     if (node==NULL)         return 0;     else     {         /\* compute the depth of each subtree \*/         int lDepth = maxDepth(node->left);         int rDepth = maxDepth(node->right);           /\* use the larger one \*/         if (lDepth > rDepth)             return(lDepth+1);         else return(rDepth+1);     }  }    /\* Helper function that allocates a new node with the     given data and NULL left and right pointers. \*/  struct node\* newNode(int data)  {      struct node\* node = (struct node\*)                                  malloc(sizeof(struct node));      node->data = data;      node->left = NULL;      node->right = NULL;        return(node);  }    int main()  {      struct node \*root = newNode(1);        root->left = newNode(2);      root->right = newNode(3);      root->left->left = newNode(4);      root->left->right = newNode(5);        printf("Height of tree is %d", maxDepth(root));        getchar();      return 0;  } |

**Output:**

Height of tree is 3

**Time Complexity:**O(n) (Please see our post [Tree Traversal](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/)for details)