# Copyright

The content of this book comes from [geeksforgeeks.org](http://geeksforgeeks.org) and it's licensed under [Creative Commons Attribution-NonCommercial-NoDerivs 2.5 India](http://creativecommons.org/licenses/by-nc-nd/2.5/in/deed.en_US)

Made by [Jing](http://www.jing-zhou.me/). 2015.

Head over to [this github repo](https://github.com/gnijuohz/geeksforgeeks-as-books) to report issues or contribute.

# Construct Tree from given Inorder and Preorder traversals

Let us consider the below traversals:

Inorder sequence: D B E A F C  
 Preorder sequence: A B D E C F

In a Preorder sequence, leftmost element is the root of the tree. So we know ‘A’ is root for given sequences. By searching ‘A’ in Inorder sequence, we can find out all elements on left side of ‘A’ are in left subtree and elements on right are in right subtree. So we know below structure now.

A  
 / \  
 / \  
 D B E F C

We recursively follow above steps and get the following tree.

A  
 / \  
 / \  
 B C  
 / \ /  
 / \ /  
D E F

Algorithm: buildTree()  
 1) Pick an element from Preorder. Increment a Preorder Index Variable (preIndex in below code) to pick next element in next recursive call.  
 2) Create a new tree node tNode with the data as picked element.  
 3) Find the picked element’s index in Inorder. Let the index be inIndex.  
 4) Call buildTree for elements before inIndex and make the built tree as left subtree of tNode.  
 5) Call buildTree for elements after inIndex and make the built tree as right subtree of tNode.  
 6) return tNode.

Thanks to Rohini and [Tushar](http://geeksforgeeks.org/forum/topic/given-inorder-and-postorder-traversals-construct-a-binary-tree#post-377) for suggesting the code.

/\* program to construct tree using inorder and preorder traversals \*/  
#include<stdio.h>  
#include<stdlib.h>  
  
/\* A binary tree node has data, pointer to left child  
 and a pointer to right child \*/  
struct node  
{  
 char data;  
 struct node\* left;  
 struct node\* right;  
};  
  
/\* Prototypes for utility functions \*/  
int search(char arr[], int strt, int end, char value);  
struct node\* newNode(char data);  
  
/\* Recursive function to construct binary of size len from  
 Inorder traversal in[] and Preorder traversal pre[]. Initial values  
 of inStrt and inEnd should be 0 and len -1. The function doesn't  
 do any error checking for cases where inorder and preorder  
 do not form a tree \*/  
struct node\* buildTree(char in[], char pre[], int inStrt, int inEnd)  
{  
 static int preIndex = 0;  
  
 if(inStrt > inEnd)  
 return NULL;  
  
 /\* Pick current node from Preorder traversal using preIndex  
 and increment preIndex \*/  
 struct node \*tNode = newNode(pre[preIndex++]);  
  
 /\* If this node has no children then return \*/  
 if(inStrt == inEnd)  
 return tNode;  
  
 /\* Else find the index of this node in Inorder traversal \*/  
 int inIndex = search(in, inStrt, inEnd, tNode->data);  
  
 /\* Using index in Inorder traversal, construct left and  
 right subtress \*/  
 tNode->left = buildTree(in, pre, inStrt, inIndex-1);  
 tNode->right = buildTree(in, pre, inIndex+1, inEnd);  
  
 return tNode;  
}  
  
/\* UTILITY FUNCTIONS \*/  
/\* Function to find index of value in arr[start...end]  
 The function assumes that value is present in in[] \*/  
int search(char arr[], int strt, int end, char value)  
{  
 int i;  
 for(i = strt; i <= end; i++)  
 {  
 if(arr[i] == value)  
 return i;  
 }  
}  
  
/\* Helper function that allocates a new node with the  
 given data and NULL left and right pointers. \*/  
struct node\* newNode(char data)  
{  
 struct node\* node = (struct node\*)malloc(sizeof(struct node));  
 node->data = data;  
 node->left = NULL;  
 node->right = NULL;  
  
 return(node);  
}  
  
/\* This funtcion is here just to test buildTree() \*/  
void printInorder(struct node\* node)  
{  
 if (node == NULL)  
 return;  
  
 /\* first recur on left child \*/  
 printInorder(node->left);  
  
 /\* then print the data of node \*/  
 printf("%c ", node->data);  
  
 /\* now recur on right child \*/  
 printInorder(node->right);  
}  
  
/\* Driver program to test above functions \*/  
int main()  
{  
 char in[] = {'D', 'B', 'E', 'A', 'F', 'C'};  
 char pre[] = {'A', 'B', 'D', 'E', 'C', 'F'};  
 int len = sizeof(in)/sizeof(in[0]);  
 struct node \*root = buildTree(in, pre, 0, len - 1);  
  
 /\* Let us test the built tree by printing Insorder traversal \*/  
 printf("\n Inorder traversal of the constructed tree is \n");  
 printInorder(root);  
 getchar();  
}

Time Complexity: O(n^2). Worst case occurs when tree is left skewed. Example Preorder and Inorder traversals for worst case are {A, B, C, D} and {D, C, B, A}.

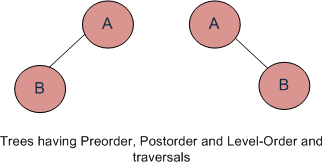
Please write comments if you find any bug in above codes/algorithms, or find other ways to solve the same problem.

Tags: [Inorder Traversal](http://www.geeksforgeeks.org/tag/inorder-traversal/), [Preorder Traversal](http://www.geeksforgeeks.org/tag/preorder-traversal/), [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/)

### Source

<http://www.geeksforgeeks.org/construct-tree-from-given-inorder-and-preorder-traversal/>

# If you are given two traversal sequences, can you construct the binary tree?

It depends on what traversals are given. If one of the traversal methods is Inorder then the tree can be constructed, otherwise not.  
   
   
 **Therefore, following combination can uniquely identify a tree.**

Inorder and Preorder.  
 Inorder and Postorder.  
 Inorder and Level-order.  
   
 **And following do not.**  
 Postorder and Preorder.  
 Preorder and Level-order.  
 Postorder and Level-order.

For example, Preorder, Level-order and Postorder traversals are same for the trees given in above diagram.

Preorder Traversal = AB  
 Postorder Traversal = BA  
 Level-Order Traversal = AB

So, even if three of them (Pre, Post and Level) are given, the tree can not be constructed.

Tags: [Binary Tree](http://www.geeksforgeeks.org/tag/binary-tree/), [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/)

### Source

<http://www.geeksforgeeks.org/if-you-are-given-two-traversal-sequences-can-you-construct-the-binary-tree/>

# Inorder Tree Traversal without recursion and without stack!

Using Morris Traversal, we can traverse the tree without using stack and recursion. The idea of Morris Traversal is based on [Threaded Binary Tree](http://en.wikipedia.org/wiki/Threaded_binary_tree). In this traversal, we first create links to Inorder successor and print the data using these links, and finally revert the changes to restore original tree.

1. Initialize current as root   
2. While current is not NULL  
 If current does not have left child  
 a) Print current’s data  
 b) Go to the right, i.e., current = current->right  
 Else  
 a) Make current as right child of the rightmost node in current's left subtree  
 b) Go to this left child, i.e., current = current->left

Although the tree is modified through the traversal, it is reverted back to its original shape after the completion. Unlike [Stack based traversal](http://geeksforgeeks.org/?p=5592), no extra space is required for this traversal.

#include<stdio.h>  
#include<stdlib.h>  
  
/\* A binary tree tNode has data, pointer to left child  
 and a pointer to right child \*/  
struct tNode  
{  
 int data;  
 struct tNode\* left;  
 struct tNode\* right;  
};  
  
/\* Function to traverse binary tree without recursion and   
 without stack \*/  
void MorrisTraversal(struct tNode \*root)  
{  
 struct tNode \*current,\*pre;  
  
 if(root == NULL)  
 return;   
  
 current = root;  
 while(current != NULL)  
 {   
 if(current->left == NULL)  
 {  
 printf(" %d ", current->data);  
 current = current->right;   
 }   
 else  
 {  
 /\* Find the inorder predecessor of current \*/   
 pre = current->left;  
 while(pre->right != NULL && pre->right != current)  
 pre = pre->right;  
  
 /\* Make current as right child of its inorder predecessor \*/  
 if(pre->right == NULL)  
 {  
 pre->right = current;  
 current = current->left;  
 }  
   
 /\* Revert the changes made in if part to restore the original   
 tree i.e., fix the right child of predecssor \*/   
 else   
 {  
 pre->right = NULL;  
 printf(" %d ",current->data);  
 current = current->right;   
 } /\* End of if condition pre->right == NULL \*/  
 } /\* End of if condition current->left == NULL\*/  
 } /\* End of while \*/  
}  
  
/\* UTILITY FUNCTIONS \*/  
/\* Helper function that allocates a new tNode with the  
 given data and NULL left and right pointers. \*/  
struct tNode\* newtNode(int data)  
{  
 struct tNode\* tNode = (struct tNode\*)  
 malloc(sizeof(struct tNode));  
 tNode->data = data;  
 tNode->left = NULL;  
 tNode->right = NULL;  
  
 return(tNode);  
}  
  
/\* Driver program to test above functions\*/  
int main()  
{  
  
 /\* Constructed binary tree is  
 1  
 / \  
 2 3  
 / \  
 4 5  
 \*/  
 struct tNode \*root = newtNode(1);  
 root->left = newtNode(2);  
 root->right = newtNode(3);  
 root->left->left = newtNode(4);  
 root->left->right = newtNode(5);   
  
 MorrisTraversal(root);  
  
 getchar();  
 return 0;  
}

References:  
 [www.liacs.nl/~deutz/DS/september28.pdf](http://www.liacs.nl/~deutz/DS/september28.pdf)  
 <http://comsci.liu.edu/~murali/algo/Morris.htm>  
 [www.scss.tcd.ie/disciplines/software\_systems/…/HughGibbonsSlides.pdf](http://www.scss.tcd.ie/disciplines/software_systems/fmg/fmg_web/IFMSIG/winter2000/HughGibbonsSlides.pdf)

Please write comments if you find any bug in above code/algorithm, or want to share more information about stack Morris Inorder Tree Traversal.

Tags: [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/)

### Source

<http://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion-and-without-stack/>

# Inorder Tree Traversal without Recursion

Using [Stack](http://en.wikipedia.org/wiki/Stack_%28data_structure%29)is the obvious way to traverse tree without recursion. Below is an algorithm for traversing binary tree using stack. See [this](http://neural.cs.nthu.edu.tw/jang/courses/cs2351/slide/animation/Iterative%20Inorder%20Traversal.pps) for step wise step execution of the algorithm.

1) Create an empty stack S.  
2) Initialize current node as root  
3) Push the current node to S and set current = current->left until current is NULL  
4) If current is NULL and stack is not empty then   
 a) Pop the top item from stack.  
 b) Print the popped item, set current = popped\_item->right   
 c) Go to step 3.  
5) If current is NULL and stack is empty then we are done.

Let us consider the below tree for example

1  
 / \  
 2 3  
 / \  
 4 5  
  
Step 1 Creates an empty stack: S = NULL  
  
Step 2 sets current as address of root: current -> 1  
  
Step 3 Pushes the current node and set current = current->left until current is NULL  
 current -> 1  
 push 1: Stack S -> 1  
 current -> 2  
 push 2: Stack S -> 2, 1  
 current -> 4  
 push 4: Stack S -> 4, 2, 1  
 current = NULL  
  
Step 4 pops from S  
 a) Pop 4: Stack S -> 2, 1  
 b) print "4"  
 c) current = NULL /\*right of 4 \*/ and go to step 3  
Since current is NULL step 3 doesn't do anything.   
  
Step 4 pops again.  
 a) Pop 2: Stack S -> 1  
 b) print "2"  
 c) current -> 5/\*right of 2 \*/ and go to step 3  
  
Step 3 pushes 5 to stack and makes current NULL  
 Stack S -> 5, 1  
 current = NULL  
  
Step 4 pops from S  
 a) Pop 5: Stack S -> 1  
 b) print "5"  
 c) current = NULL /\*right of 5 \*/ and go to step 3  
Since current is NULL step 3 doesn't do anything  
  
Step 4 pops again.  
 a) Pop 1: Stack S -> NULL  
 b) print "1"  
 c) current -> 3 /\*right of 5 \*/   
  
Step 3 pushes 3 to stack and makes current NULL  
 Stack S -> 3  
 current = NULL  
  
Step 4 pops from S  
 a) Pop 3: Stack S -> NULL  
 b) print "3"  
 c) current = NULL /\*right of 3 \*/   
  
Traversal is done now as stack S is empty and current is NULL.

Implementation:

#include<stdio.h>  
#include<stdlib.h>  
#define bool int  
  
/\* A binary tree tNode has data, pointer to left child  
 and a pointer to right child \*/  
struct tNode  
{  
 int data;  
 struct tNode\* left;  
 struct tNode\* right;  
};  
  
/\* Structure of a stack node. Linked List implementation is used for   
 stack. A stack node contains a pointer to tree node and a pointer to   
 next stack node \*/  
struct sNode  
{  
 struct tNode \*t;  
 struct sNode \*next;  
};  
  
/\* Stack related functions \*/  
void push(struct sNode\*\* top\_ref, struct tNode \*t);  
struct tNode \*pop(struct sNode\*\* top\_ref);  
bool isEmpty(struct sNode \*top);  
  
/\* Iterative function for inorder tree traversal \*/  
void inOrder(struct tNode \*root)  
{  
 /\* set current to root of binary tree \*/  
 struct tNode \*current = root;  
 struct sNode \*s = NULL; /\* Initialize stack s \*/  
 bool done = 0;  
  
 while (!done)  
 {  
 /\* Reach the left most tNode of the current tNode \*/  
 if(current != NULL)  
 {  
 /\* place pointer to a tree node on the stack before traversing   
 the node's left subtree \*/  
 push(&s, current);   
 current = current->left;   
 }  
   
 /\* backtrack from the empty subtree and visit the tNode   
 at the top of the stack; however, if the stack is empty,  
 you are done \*/  
 else   
 {  
 if (!isEmpty(s))  
 {  
 current = pop(&s);  
 printf("%d ", current->data);  
  
 /\* we have visited the node and its left subtree.  
 Now, it's right subtree's turn \*/  
 current = current->right;  
 }  
 else  
 done = 1;   
 }  
 } /\* end of while \*/   
}   
  
/\* UTILITY FUNCTIONS \*/  
/\* Function to push an item to sNode\*/  
void push(struct sNode\*\* top\_ref, struct tNode \*t)  
{  
 /\* allocate tNode \*/  
 struct sNode\* new\_tNode =  
 (struct sNode\*) malloc(sizeof(struct sNode));  
  
 if(new\_tNode == NULL)  
 {  
 printf("Stack Overflow \n");  
 getchar();  
 exit(0);  
 }   
  
 /\* put in the data \*/  
 new\_tNode->t = t;  
  
 /\* link the old list off the new tNode \*/  
 new\_tNode->next = (\*top\_ref);   
  
 /\* move the head to point to the new tNode \*/  
 (\*top\_ref) = new\_tNode;  
}  
  
/\* The function returns true if stack is empty, otherwise false \*/  
bool isEmpty(struct sNode \*top)  
{  
 return (top == NULL)? 1 : 0;  
}   
  
/\* Function to pop an item from stack\*/  
struct tNode \*pop(struct sNode\*\* top\_ref)  
{  
 struct tNode \*res;  
 struct sNode \*top;  
  
 /\*If sNode is empty then error \*/  
 if(isEmpty(\*top\_ref))  
 {  
 printf("Stack Underflow \n");  
 getchar();  
 exit(0);  
 }  
 else  
 {  
 top = \*top\_ref;  
 res = top->t;  
 \*top\_ref = top->next;  
 free(top);  
 return res;  
 }  
}  
  
/\* Helper function that allocates a new tNode with the  
 given data and NULL left and right pointers. \*/  
struct tNode\* newtNode(int data)  
{  
 struct tNode\* tNode = (struct tNode\*)  
 malloc(sizeof(struct tNode));  
 tNode->data = data;  
 tNode->left = NULL;  
 tNode->right = NULL;  
  
 return(tNode);  
}  
  
/\* Driver program to test above functions\*/  
int main()  
{  
  
 /\* Constructed binary tree is  
 1  
 / \  
 2 3  
 / \  
 4 5  
 \*/  
 struct tNode \*root = newtNode(1);  
 root->left = newtNode(2);  
 root->right = newtNode(3);  
 root->left->left = newtNode(4);  
 root->left->right = newtNode(5);   
  
 inOrder(root);  
  
 getchar();  
 return 0;  
}

Time Complexity: O(n)

References:  
 <http://web.cs.wpi.edu/~cs2005/common/iterative.inorder>  
 <http://neural.cs.nthu.edu.tw/jang/courses/cs2351/slide/animation/Iterative%20Inorder%20Traversal.pps>

See [this post](http://geeksforgeeks.org/?p=6358) for another approach of Inorder Tree Traversal without recursion and without stack!

Please write comments if you find any bug in above code/algorithm, or want to share more information about stack based Inorder Tree Traversal.

Tags: [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/)

### Source

<http://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion/>

# Write a C program to Delete a Tree.

To delete a tree we must traverse all the nodes of the tree and delete them one by one. So which traversal we should use – Inorder or Preorder or Postorder. Answer is simple – Postorder, because before deleting the parent node we should delete its children nodes first

We can delete tree with other traversals also with extra space complexity but why should we go for other traversals if we have Postorder available which does the work without storing anything in same time complexity.

For the following tree nodes are deleted in order – 4, 5, 2, 3, 1



Example Tree

**Program**

#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;  
};  
  
/\* 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);  
}  
  
/\* This function traverses tree in post order to   
 to delete each and every node of the tree \*/  
void deleteTree(struct node\* node)   
{  
 if (node == NULL) return;  
  
 /\* first delete both subtrees \*/  
 deleteTree(node->left);  
 deleteTree(node->right);  
   
 /\* then delete the node \*/  
 printf("\n Deleting node: %d", node->data);  
 free(node);  
}   
  
  
/\* Driver program to test deleteTree function\*/   
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);   
   
 deleteTree(root);   
 root = NULL;  
  
 printf("\n Tree deleted ");  
   
 getchar();  
 return 0;  
}

The above deleteTree() function deletes the tree, but doesn’t change root to NULL which may cause problems if the user of deleteTree() doesn’t change root to NULL and tires to access values using root pointer. We can modify the deleteTree() function to take reference to the root node so that this problem doesn’t occur. See the following code.

#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;  
};  
  
/\* 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);  
}  
  
/\* This function is same as deleteTree() in the previous program \*/  
void \_deleteTree(struct node\* node)  
{  
 if (node == NULL) return;  
  
 /\* first delete both subtrees \*/  
 \_deleteTree(node->left);  
 \_deleteTree(node->right);  
  
 /\* then delete the node \*/  
 printf("\n Deleting node: %d", node->data);  
 free(node);  
}  
  
/\* Deletes a tree and sets the root as NULL \*/  
void deleteTree(struct node\*\* node\_ref)  
{  
 \_deleteTree(\*node\_ref);  
 \*node\_ref = NULL;  
}  
  
/\* Driver program to test deleteTree function\*/  
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);  
  
 // Note that we pass the address of root here  
 deleteTree(&root);  
 printf("\n Tree deleted ");  
  
 getchar();  
 return 0;  
}

**Time Complexity:** O(n)  
 **Space Complexity:** If we don’t consider size of stack for function calls then O(1) otherwise O(n)

Tags: [Delete Tree](http://www.geeksforgeeks.org/tag/delete-tree/), [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/), [Trees](http://www.geeksforgeeks.org/tag/tree/)

### Source

<http://www.geeksforgeeks.org/write-a-c-program-to-delete-a-tree/>

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

Maximum depth or height of the below tree is 3.



Example Tree

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('1') maxDepth('3') = 1  
= max(maxDepth('4'), maxDepth('5')) + 1  
= 1 + 1 = 2   
 / \  
 / \  
 / \  
 / \  
 / \  
 maxDepth('4') = 1 maxDepth('5') = 1

**Implementation:**

#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("Hight of tree is %d", maxDepth(root));  
   
 getchar();  
 return 0;  
}

**Time Complexity:** O(n) (Please see our post [Tree Traversal](http://geeksforgeeks.org/?p=618)for details)

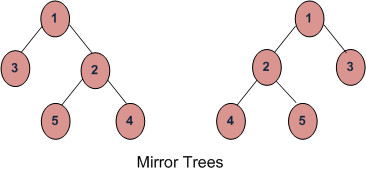
**References:**  
 <http://cslibrary.stanford.edu/110/BinaryTrees.html>

Tags: [Height of a Tree](http://www.geeksforgeeks.org/tag/height-of-a-tree/), [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/), [Trees](http://www.geeksforgeeks.org/tag/tree/)

### Source

<http://www.geeksforgeeks.org/write-a-c-program-to-find-the-maximum-depth-or-height-of-a-tree/>

# Write an Efficient C Function to Convert a Binary Tree into its Mirror Tree

Mirror of a Tree: Mirror of a Binary Tree T is another Binary Tree M(T) with left and right children of all non-leaf nodes interchanged.  
   
   
 Trees in the below figure are mirror of each other  
   
 **Algorithm** - Mirror(tree):

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)  
(2) Call Mirror for right-subtree i.e., Mirror(left-subtree)  
(3) Swap left and right subtrees.  
 temp = left-subtree  
 left-subtree = right-subtree  
 right-subtree = temp

**Program:**

#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;  
};  
  
/\* 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);  
}  
  
  
/\* Change a tree so that the roles of the left and   
 right pointers are swapped at every node.  
  
 So the tree...  
 4  
 / \  
 2 5  
 / \  
 1 3  
  
 is changed to...  
 4  
 / \  
 5 2  
 / \  
 3 1  
\*/  
void mirror(struct node\* node)   
{  
 if (node==NULL)   
 return;   
 else   
 {  
 struct node\* temp;  
   
 /\* do the subtrees \*/  
 mirror(node->left);  
 mirror(node->right);  
  
 /\* swap the pointers in this node \*/  
 temp = node->left;  
 node->left = node->right;  
 node->right = temp;  
 }  
}   
  
  
/\* Helper function to test mirror(). Given a binary  
 search tree, print out its data elements in   
 increasing sorted order.\*/  
void inOrder(struct node\* node)   
{  
 if (node == NULL)   
 return;  
   
 inOrder(node->left);  
 printf("%d ", node->data);  
  
 inOrder(node->right);  
}   
  
  
/\* Driver program to test mirror() \*/  
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);   
   
 /\* Print inorder traversal of the input tree \*/  
 printf("\n Inorder traversal of the constructed tree is \n");  
 inOrder(root);  
   
 /\* Convert tree to its mirror \*/  
 mirror(root);   
   
 /\* Print inorder traversal of the mirror tree \*/  
 printf("\n Inorder traversal of the mirror tree is \n");   
 inOrder(root);  
   
 getchar();  
 return 0;   
}

**Time & Space Complexities:** This program is similar to traversal of tree space and time complexities will be same as Tree traversal (Please see our [Tree Traversal](http://geeksforgeeks.org/?p=618) post for details)

Tags: [Convert to Mirror](http://www.geeksforgeeks.org/tag/convert-to-mirror/), [Get the Mirror](http://www.geeksforgeeks.org/tag/get-the-mirror/), [Mirror Tree](http://www.geeksforgeeks.org/tag/mirror-tree/), [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/), [Trees](http://www.geeksforgeeks.org/tag/tree/)

### Source

<http://www.geeksforgeeks.org/write-an-efficient-c-function-to-convert-a-tree-into-its-mirror-tree/>

# Write C Code to Determine if Two Trees are Identical

Two trees are identical when they have same data and arrangement of data is also same.

To identify if two trees are identical, we need to traverse both trees simultaneously, and while traversing we need to compare data and children of the trees.

**Algorithm:**

sameTree(tree1, tree2)  
1. If both trees are empty then return 1.  
2. Else If both trees are non -empty  
 (a) Check data of the root nodes (tree1->data == tree2->data)  
 (b) Check left subtrees recursively i.e., call sameTree(   
 tree1->left\_subtree, tree2->left\_subtree)  
 (c) Check right subtrees recursively i.e., call sameTree(   
 tree1->right\_subtree, tree2->right\_subtree)  
 (d) If a,b and c are true then return 1.  
3 Else return 0 (one is empty and other is not)

#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;  
};  
  
/\* 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);  
}  
  
/\* Given two trees, return true if they are  
 structurally identical \*/  
int identicalTrees(struct node\* a, struct node\* b)  
{  
 /\*1. both empty \*/  
 if (a==NULL && b==NULL)  
 return 1;  
  
 /\* 2. both non-empty -> compare them \*/  
 if (a!=NULL && b!=NULL)  
 {  
 return  
 (  
 a->data == b->data &&  
 identicalTrees(a->left, b->left) &&  
 identicalTrees(a->right, b->right)  
 );  
 }  
   
 /\* 3. one empty, one not -> false \*/  
 return 0;  
}   
  
/\* Driver program to test identicalTrees function\*/  
int main()  
{  
 struct node \*root1 = newNode(1);  
 struct node \*root2 = newNode(1);  
 root1->left = newNode(2);  
 root1->right = newNode(3);  
 root1->left->left = newNode(4);  
 root1->left->right = newNode(5);   
  
 root2->left = newNode(2);  
 root2->right = newNode(3);  
 root2->left->left = newNode(4);  
 root2->left->right = newNode(5);   
  
 if(identicalTrees(root1, root2))  
 printf("Both tree are identical.");  
 else  
 printf("Trees are not identical.");  
  
 getchar();  
 return 0;  
}

**Time Complexity:**  
 Complexity of the identicalTree() will be according to the tree with lesser number of nodes. Let number of nodes in two trees be m and n then complexity of sameTree() is O(m) where m

Tags: [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/), [Trees](http://www.geeksforgeeks.org/tag/tree/)

### Source

<http://www.geeksforgeeks.org/write-c-code-to-determine-if-two-trees-are-identical/>

# Tree Traversals

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



Example Tree

Depth First Traversals:  
 (a) Inorder  
 (b) Preorder  
 (c) Postorder

Breadth First or Level Order Traversal  
 Please see [this](http://geeksforgeeks.org/?p=2686)post for Breadth First Traversal.

**Inorder Traversal:**

Algorithm Inorder(tree)  
 1. Traverse the left subtree, i.e., call Inorder(left-subtree)  
 2. Visit the root.  
 3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Uses of Inorder  
 In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder itraversal s reversed, can be used.  
 Example: Inorder traversal for the above given figure is 4 2 5 1 3.  
   
 **Preorder Traversal:**

Algorithm Preorder(tree)  
 1. Visit the root.  
 2. Traverse the left subtree, i.e., call Preorder(left-subtree)  
 3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Uses of Preorder  
 Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree. Please see <http://en.wikipedia.org/wiki/Polish_notation> to know why prefix expressions are useful.  
 Example: Preorder traversal for the above given figure is 1 2 4 5 3.  
   
 **Postorder Traversal:**

Algorithm Postorder(tree)  
 1. Traverse the left subtree, i.e., call Postorder(left-subtree)  
 2. Traverse the right subtree, i.e., call Postorder(right-subtree)  
 3. Visit the root.

Uses of Postorder  
 Postorder traversal is used to delete the tree. Please see [the question for deletion of tree](http://geeksforgeeks.org/?p=654) for details. Postorder traversal is also useful to get the postfix expression of an expression tree. Please see <http://en.wikipedia.org/wiki/Reverse_Polish_notation> to for the usage of postfix expression.

Example: Postorder traversal for the above given figure is 4 5 2 3 1.

#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;  
};  
  
/\* 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);  
}  
  
/\* Given a binary tree, print its nodes according to the  
 "bottom-up" postorder traversal. \*/  
void printPostorder(struct node\* node)  
{  
 if (node == NULL)  
 return;  
  
 // first recur on left subtree  
 printPostorder(node->left);  
  
 // then recur on right subtree  
 printPostorder(node->right);  
  
 // now deal with the node  
 printf("%d ", node->data);  
}  
  
/\* Given a binary tree, print its nodes in inorder\*/  
void printInorder(struct node\* node)  
{  
 if (node == NULL)  
 return;  
  
 /\* first recur on left child \*/  
 printInorder(node->left);  
  
 /\* then print the data of node \*/  
 printf("%d ", node->data);   
  
 /\* now recur on right child \*/  
 printInorder(node->right);  
}  
  
/\* Given a binary tree, print its nodes in inorder\*/  
void printPreorder(struct node\* node)  
{  
 if (node == NULL)  
 return;  
  
 /\* first print data of node \*/  
 printf("%d ", node->data);   
  
 /\* then recur on left sutree \*/  
 printPreorder(node->left);   
  
 /\* now recur on right subtree \*/  
 printPreorder(node->right);  
}   
  
/\* Driver program to test above functions\*/  
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("\n Preorder traversal of binary tree is \n");  
 printPreorder(root);  
  
 printf("\n Inorder traversal of binary tree is \n");  
 printInorder(root);   
  
 printf("\n Postorder traversal of binary tree is \n");  
 printPostorder(root);  
  
 getchar();  
 return 0;  
}

**Time Complexity:** O(n)  
 Let us prove it:

Complexity function T(n) — for all problem where tree traversal is involved — can be defined as:

T(n) = T(k) + T(n – k – 1) + c

Where k is the number of nodes on one side of root and n-k-1 on the other side.

Let’s do analysis of boundary conditions

Case 1: Skewed tree (One of the subtrees is empty and other subtree is non-empty )

k is 0 in this case.  
 T(n) = T(0) + T(n-1) + c  
 T(n) = 2T(0) + T(n-2) + 2c  
 T(n) = 3T(0) + T(n-3) + 3c  
 T(n) = 4T(0) + T(n-4) + 4c

…………………………………………  
 ………………………………………….  
 T(n) = (n-1)T(0) + T(1) + (n-1)c  
 T(n) = nT(0) + (n)c

Value of T(0) will be some constant say d. (traversing a empty tree will take some constants time)

T(n) = n(c+d)  
 T(n) = (-)(n) (Theta of n)

Case 2: Both left and right subtrees have equal number of nodes.

T(n) = 2T(|\_n/2\_|) + c

This recursive function is in the standard form (T(n) = aT(n/b) + (-)(n) ) for master method <http://en.wikipedia.org/wiki/Master_theorem>. If we solve it by master method we get (-)(n)

**Auxiliary Space :** If we don’t consider size of stack for function calls then O(1) otherwise O(n).

Tags: [Inorder Traversal](http://www.geeksforgeeks.org/tag/inorder-traversal/), [PostOrder Traversal](http://www.geeksforgeeks.org/tag/postorder-traversal/), [Preorder Traversal](http://www.geeksforgeeks.org/tag/preorder-traversal/), [Tree Traveral](http://www.geeksforgeeks.org/tag/tree-traveral/), [Trees](http://www.geeksforgeeks.org/tag/tree/), [Tutorial](http://www.geeksforgeeks.org/tag/tutorial/)

### Source

<http://www.geeksforgeeks.org/618/>