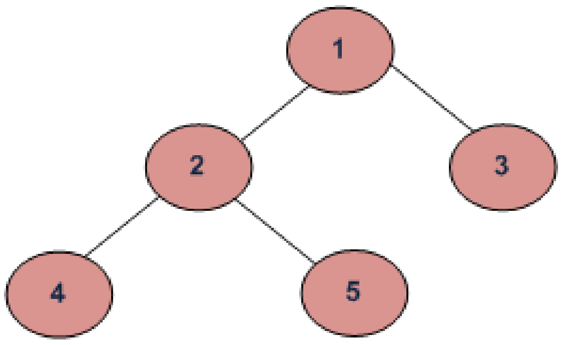
Write a C program to Delete a Tree.

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

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;
};
```

```
/* 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;
}
```

Run on IDE

Java

```
// Java program to delete a tree

// A binary tree node
class Node {

    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {

    static Node root;

    /*  This function traverses tree in post order to
     to delete each and every node of the tree */
    void deleteTree(Node node) {
        if (node == null) {
            return;
        }

        /* first delete both subtrees */
        deleteTree(node.left);
        deleteTree(node.right);

        /* then delete the node */
        System.out.println("The deleted node is " + node.data);
        node = null;
    }

    /* Driver program to test mirror() */
    public static void main(String[] args) {

        BinaryTree tree = new BinaryTree();

        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        /* Print all root-to-leaf paths of the input tree */
        tree.deleteTree(root);
        root = null;
        System.out.println("Tree deleted");

    }
}
```

Run on IDE

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;
};
```

```c
/* 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;
}
```

Run on IDE

Time Complexity: O(n)

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

Spend less time debugging, more time coding.
Get Visual Studio Code FREE
Microsoft

75 Comments Category: Trees Tags: Delete Tree , Tree Traveral , Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?

(Login to Rate and Mark)

1.5 Average Difficulty : **1.5/5.0**
Based on **22** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Like Share 7 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.