# Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology, Pune-37

*(An Autonomous Institute of Savitribai Phule Pune University)*



**Department of Artificial Intelligence and Data Science**

|  |  |
| --- | --- |
| **Division** | A |
| **Batch** | 1 |
| **GR-no** | 12211542 |
| **Roll No.** | 07 |
| **Name** | Anish Naphade |
| **Subject** | Advance Data Structures |

**Assignment-2**

**Problem Statement:**

**Tree Functions**

1. **Count nodes, 2. Find height, 3. Mirror Image**

**Code:**

#include<stdio.h>#include<stdlib.h>struct tree\_node { int data; struct tree\_node \*left; struct tree\_node \*right;};struct tree\_node \*newNode(int data) { struct tree\_node \*newNode = (struct tree\_node \*)malloc(sizeof(struct tree\_node)); if (newNode == NULL) { printf("Memory allocation failed\n"); exit(EXIT\_FAILURE); } newNode->data = data; newNode->left = NULL; newNode->right = NULL; return newNode;}void printInorder(struct tree\_node \*node) { if (node == NULL) return; printInorder(node->left); printf("%d ", node->data); printInorder(node->right);}struct tree\_node \*search(struct tree\_node \*root, int item) { if (root == NULL || root->data == item) return root; if (root->data > item) return search(root->left, item); return search(root->right, item);}int count(struct tree\_node \*root){ if (root == NULL) return 0; return 1 + count(root->left) + count(root->right);}int maxDepth(struct tree\_node \*node) { if (node == NULL) return 0; else { int lDepth = maxDepth(node->left); int rDepth = maxDepth(node->right); if (lDepth > rDepth) return (lDepth + 1); else return (rDepth + 1); }}void mirror(struct tree\_node\* node) { if (node == NULL) return; else { struct tree\_node\* temp; mirror(node->left); mirror(node->right); temp = node->left; node->left = node->right; node->right = temp; }}struct tree\_node \*findMin(struct tree\_node \*root) { if (root == NULL) return NULL; else if (root->left != NULL) return findMin(root->left); return root;}struct tree\_node \*insert(int item, struct tree\_node \*root) { if (root == NULL) return newNode(item); if (item < root->data) root->left = insert(item, root->left); else if (item > root->data) root->right = insert(item, root->right); return root;}struct tree\_node \*deleteNode(struct tree\_node \*root, int x) { if (root == NULL) return NULL; if (x > root->data) root->right = deleteNode(root->right, x); else if (x < root->data) root->left = deleteNode(root->left, x); else { if (root->left == NULL && root->right == NULL) { free(root); return NULL; } else if (root->left == NULL || root->right == NULL) { struct tree\_node \*temp; if (root->left == NULL) temp = root->right; else temp = root->left; free(root); return temp; } else { struct tree\_node \*temp = findMin(root->right); root->data = temp->data; root->right = deleteNode(root->right, temp->data); } } return root;}int main() { int choice, item; struct tree\_node \*root = NULL; do { printf("\n1) Inorder Traversal\n"); printf("2) Insertion\n"); printf("3) Delete\n"); printf("4) Count Nodes\n"); printf("5) Height/Depth\n"); printf("6) Mirror Tree\n"); printf("7) EXIT\n"); printf("Choose one from Above: "); scanf("%d", &choice); switch (choice) { case 1: printf("\nPrinting Inorder: "); printInorder(root); break; case 2: printf("Enter data to insert: "); scanf("%d", &item); root = insert(item, root); break; case 3: printf("Enter data to delete: "); scanf("%d", &item); root = deleteNode(root, item); break; case 4: printf("Total number of nodes: %d\n", count(root)); break; case 5: printf("Total height/depth of tree: %d\n", maxDepth(root)); break; case 6: mirror(root); printf("Tree mirrored successfully.\n"); break; case 7: // Free memory before exiting // Not implemented here, but recommended in a real-world scenario exit(EXIT\_SUCCESS); default: printf("Invalid choice\n"); break; } } while (1); return 0;}

O/P:



