1. **Write a program to create a Binary Search Tree (BST) and traverse it using:**
   * Inorder traversal
   * Preorder traversal
   * Postorder traversal

Solution

#include <stdio.h>

#include <stdlib.h>

// Define a node structure

struct Node {

int key;

struct Node \*left, \*right;

};

// Function to create a new node

struct Node\* newNode(int item) {

struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

// Function to insert a node in the BST

struct Node\* insert(struct Node\* node, int key) {

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

return node;

}

// Inorder traversal

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

printf("%d ", root->key);

inorder(root->right);

}

}

// Preorder traversal

void preorder(struct Node\* root) {

if (root != NULL) {

printf("%d ", root->key);

preorder(root->left);

preorder(root->right);

}

}

// Postorder traversal

void postorder(struct Node\* root) {

if (root != NULL) {

postorder(root->left);

postorder(root->right);

printf("%d ", root->key);

}

}

// Main function

int main() {

struct Node\* root = NULL;

int elements[] = {50, 30, 20, 40, 70, 60, 80};

int n = sizeof(elements)/sizeof(elements[0]);

for (int i = 0; i < n; i++) {

root = insert(root, elements[i]);

}

printf("Inorder traversal:\n");

inorder(root);

printf("\nPreorder traversal:\n");

preorder(root);

printf("\nPostorder traversal:\n");

postorder(root);

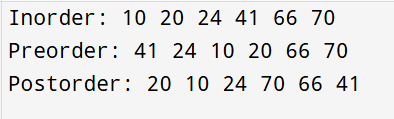
return 0;

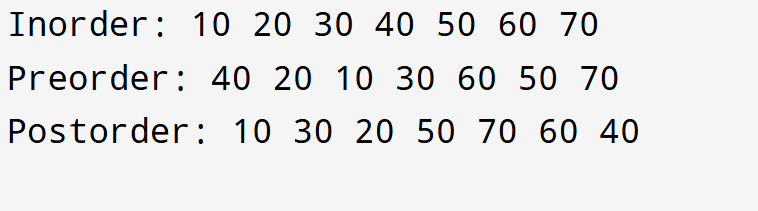
}

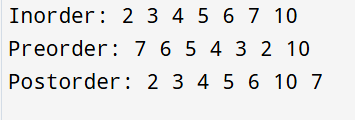
OUTPUT -:

A screenshot of a computer

AI-generated content may be incorrect.







1. **Assuming that we already have a BST with the address root, write a function to count the total number of nodes.**
   * The function should not return any value.

Solution

// Function to count total number of nodes (void function with pointer)

void countNodes(struct Node\* root, int\* count) {

if (root != NULL) {

(\*count)++;

countNodes(root->left, count);

countNodes(root->right, count);

}

}

// Main function

// Count total number of nodes

int total = 0;

countNodes(root, &total);

printf("Total number of nodes in the BST: %d\n", total);

OUTPUT





****

****

1. **Write a function to count the total number of leaf nodes.**

Solution

// Function to count total number of leaf nodes (void function)

void countLeafNodes(struct Node\* root, int\* leafCount) {

if (root != NULL) {

if (root->left == NULL && root->right == NULL) {

(\*leafCount)++; // Node is a leaf

}

countLeafNodes(root->left, leafCount);

countLeafNodes(root->right, leafCount);

}

}

//Main Function

int leafCount = 0;

countLeafNodes(root, &leafCount);

printf("Total leaf nodes: %d\n", leafCount);

OUTPUT







1. **Write a function to count the number of nodes that have only one child.**

**Solution**

// Function to count nodes with only one child

void countSingleChildNodes(struct Node\* root, int\* count) {

if (root != NULL) {

// Node has exactly one child

if ((root->left == NULL && root->right != NULL) ||

(root->left != NULL && root->right == NULL)) {

(\*count)++;

}

countSingleChildNodes(root->left, count);

countSingleChildNodes(root->right, count);

}

}

int singleChildCount = 0;

countSingleChildNodes(root, &singleChildCount);

printf("Total number of nodes with only one child: %d\n", singleChildCount);

OUTPUT









**5. Write a function to count the number of nodes that have only a left child.**

Solution

// Function to count nodes with only a left child

void countLeftChildOnly(struct Node\* root, int\* count) {

if (root != NULL) {

if (root->left != NULL && root->right == NULL) {

(\*count)++;

}

countLeftChildOnly(root->left, count);

countLeftChildOnly(root->right, count);

}

}

int leftOnlyCount = 0;

countLeftChildOnly(root, &leftOnlyCount);

printf("Total number of nodes with only a left child: %d\n", leftOnlyCount);

Output





