**Question-1: Binary Search Tree implementation using C**

#include <stdio.h>

#include <stdlib.h>

// Structure to represent a node in the BST

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Function to insert a new node into the BST

struct Node\* insert(struct Node\* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

} else if (data > root->data) {

root->right = insert(root->right, data);

}

return root;

}

// Inorder traversal (Left, Root, Right)

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

// Preorder traversal (Root, Left, Right)

void preorder(struct Node\* root) {

if (root != NULL) {

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

preorder(root->left);

preorder(root->right);

}

}

// Postorder traversal (Left, Right, Root)

void postorder(struct Node\* root) {

if (root != NULL) {

postorder(root->left);

postorder(root->right);

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

}

}

// Function to search for a key in the BST

struct Node\* search(struct Node\* root, int key) {

if (root == NULL || root->data == key) {

return root;

}

if (key < root->data) {

return search(root->left, key);

} else {

return search(root->right, key);

}

}

int main() {

struct Node\* root = NULL;

int arr[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2};

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

// Insert elements into the BST

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

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

}

printf("Value of root data = %d\n",root->data);

// Perform inorder traversal

printf("Inorder traversal: ");

inorder(root);

printf("\n");

// Perform preorder traversal

printf("Preorder traversal: ");

preorder(root);

printf("\n");

// Perform postorder traversal

printf("Postorder traversal: ");

postorder(root);

printf("\n");

// Search for a given key in the BST

int key;

printf("Enter a key to search: ");

scanf("%d", &key);

struct Node\* result = search(root, key);

if (result != NULL) {

printf("Element %d is found in the BST.\n", key);

} else {

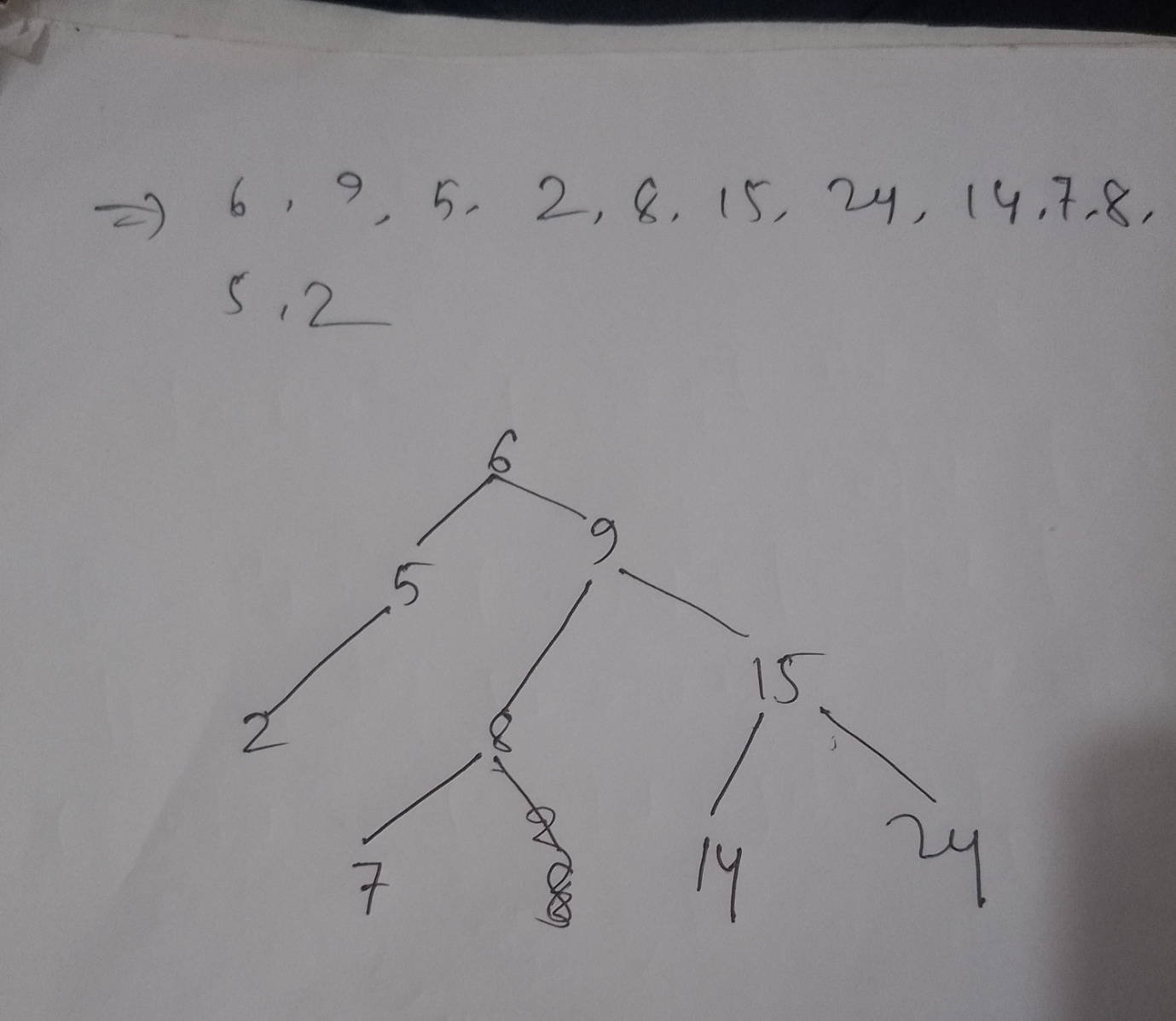
printf("Element %d is not found in the BST.\n", key);

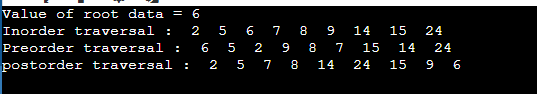
}

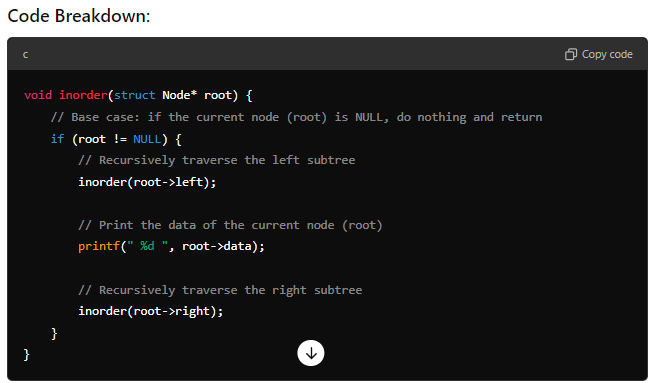
return 0;

}

A Binary search tree will constructed:







The function `void inorder(struct Node\* root)` is a recursive function that performs an \*\*in-order traversal\*\* of a binary tree. Let's break it down step by step:

### In-Order Traversal:

In in-order traversal, the nodes of a binary tree are visited in the following order:

1. Visit the left subtree.

2. Visit the root node (current node).

3. Visit the right subtree.

### How It Works:

1. \*\*Base Case (`if (root != NULL)`):\*\*

- If the `root` (current node) is `NULL`, the function returns immediately, meaning it stops recursion at this point. This is important to avoid trying to access a non-existent node.

2. \*\*Recursive Left Subtree (`inorder(root->left)`):\*\*

- The function first recursively calls itself to visit the left child of the current node (`root->left`). This ensures that all the nodes in the left subtree are visited first before moving to the current node (root).

3. \*\*Print the Root (`printf(" %d ", root->data)`):\*\*

-After the left subtree has been fully visited, the function prints the value stored in the current node (`root->data`).

4. \*\*Recursive Right Subtree (`inorder(root->right)`):\*\*

- Once the current node's data is printed, the function recursively calls itself to visit the right child of the current node (`root->right`), ensuring that all nodes in the right subtree are visited after the current node.



