//Binary Tree traversal in C:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int item;
  struct node* left;
  struct node* right;
};
// Inorder traversal
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ->", root->item);
  inorderTraversal(root->right);
}
// Preorder traversal
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}
// Postorder traversal
void postorderTraversal(struct node* root) {
```

```
    if (root == NULL) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ->", root->item);
}
// Create a new Node
struct node* createNode(value) {
    struct node* newNode = malloc(sizeof(struct node));
    newNode->item = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
    root->left = createNode(value);
    return root->left;
}
// Insert on the right of the node
struct node* insertRight(struct node* root, int value) {
    root->right = createNode(value);
    return root->right;
}
int main() {
```

```
    struct node* root = createNode(1);
    insertLeft(root, 2);
    insertRight(root, 3);
    insertLeft(root->left, 4);
    printf("Inorder traversal \n");
    inorderTraversal(root);
    printf("\nPreorder traversal \n");
    preorderTraversal(root);
    printf("\nPostorder traversal \n");
    postorderTraversal(root);
}
```

OUTPUT:

```
Inorder traversal
4 ->2 ->1 ->3 ->
Preorder traversal
1 ->2 ->4 ->3 ->
Postorder traversal
4 ->2 ->3 ->1 ->
```

//2.SEARCHING IN BINARY TREE:

```
#include <stdio.h>
#include <stdlib.h>
// Structure for a BST node
struct Node {
```

```
    int data;
    struct Node* left;
    struct Node* right;
};
// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
// Function to insert a node into the BST
struct Node* insert(struct Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else {
        root->right = insert(root->right, value);
    }
    return root;
```

```
}
// Function to search for a value in the BST
struct Node* search(struct Node* root, int value) {
    if (root == NULL || root->data == value) {
        return root;
    }
    if (value < root->data) {
        return search(root->left, value);
    } else {
        return search(root->right, value);
    }
}
int main() {
    // Input BST: [8,3,10,1,6,null,14,null,null,4,7,13,null]
    struct Node* root = NULL;
    root = insert(root, 8);
    root = insert(root, 3);
    root = insert(root, 10);
    root = insert(root, 1);
    root = insert(root, 6);
    root = insert(root, 4);
    root = insert(root, 7);
    root = insert(root, 14);
    root = insert(root, 13);
```

```
    int key = 6;
    struct Node* result = search(root, key);
    if (result != NULL) {
        printf("Key %d found in the BST.\n", key);
    } else {
        printf("Key %d not found in the BST.\n", key);
    }
    return 0;
}
```

OUTPUT:

Key 6 found in the BST

//3.BINARY TREE:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Error allocating memory!\n");
        exit(1);
```

```
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
Node* insertNode(Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}
void freeTree(Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
```

```
int main() {
    Node* root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 10);
    insertNode(root, 5);
    insertNode(root, 15);
    insertNode(root, 2);
    insertNode(root, 7);
    insertNode(root, 12);
    insertNode(root, 20);
    freeTree(root);
    return 0;
}
```

OUTPUT: