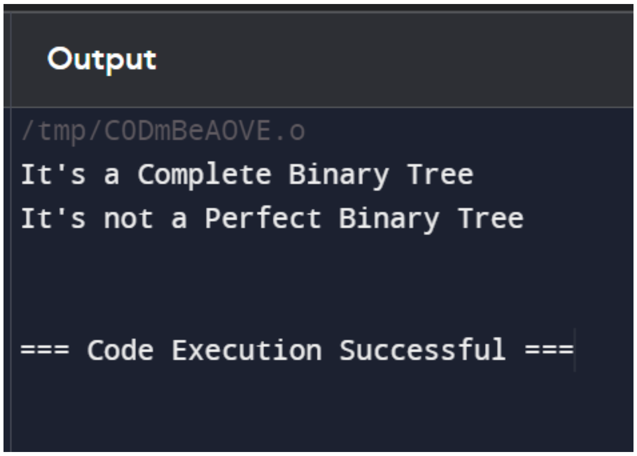
1)write a c code for binary tree using perfect,complete and full.

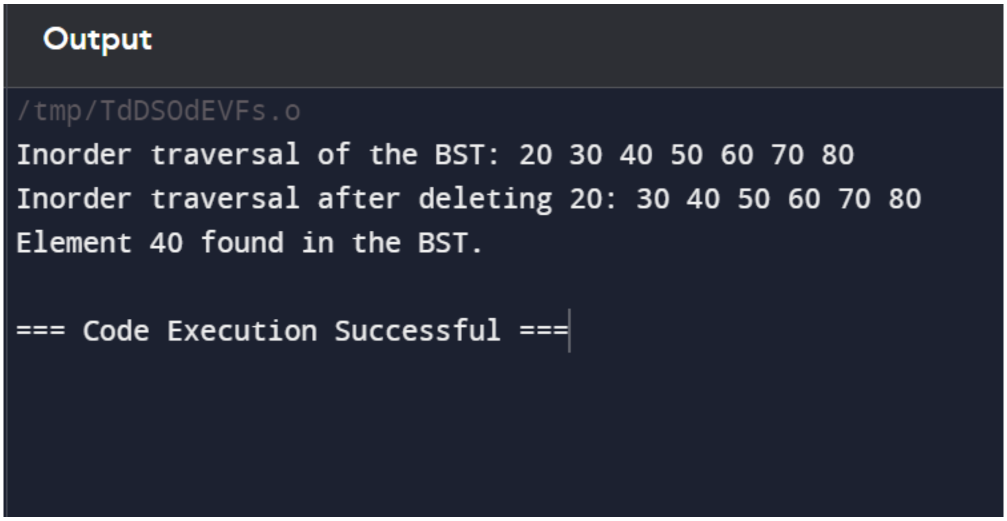
```
#include <stdio.h>
#include<stdlib.h>
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
int isComplete(struct Node* root, int index, int numNodes) {
    if (root == NULL)
        return 1;
    if (index >= numNodes)
        return 0;
    return (isComplete(root->left, 2 * index + 1, numNodes) && isComplete(root->right, 2 * index + 2, numNodes));
}
int isPerfect(struct Node* root, int depth, int level) {
    if (root == NULL)
        return 1;
    if (root->left == NULL && root->right == NULL)
        return (depth == level + 1);
    if (root->left == NULL || root->right == NULL)
        return 0;
    return isPerfect(root->left, depth, level + 1) && isPerfect(root->right, depth, level + 1);
}
int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    int numNodes = 5;
    if (isComplete(root, 0, numNodes))
        printf("It's a Complete Binary Tree\n");
    else
        printf("It's not a Complete Binary Tree\n");
    int depth = 0, level = 0;
    if (isPerfect(root, depth, level))
        printf("It's a Perfect Binary Tree\n");
    else
        printf("It's not a Perfect Binary Tree\n");
    return 0;
}
```



```
/tmp/C0DmBeAOVE.o
It's a Complete Binary Tree
It's not a Perfect Binary Tree


=== Code Execution Successful ===
```

2)write a c code for binary search using operations insert,delete,search

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node *left, *right;

};

struct Node* insert(struct Node* root, int key) {

    if (root == NULL) {

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

        newNode->data = key;

        newNode->left = newNode->right = NULL;

        return newNode;

    }

    if (key < root->data)

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

    else if (key > root->data)

        root->right = insert(root->right, key);
    return root;
}
struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
struct Node* deleteNode(struct Node* root, int key) {
    if (root == NULL) return root;
    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (root->data < key)
        return search(root->right, key);

    return search(root->left, key);
}
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);

    insert(root, 80);

    printf("Inorder traversal of the BST: ");

    inorder(root);

    root = deleteNode(root, 20);

    printf("\nInorder traversal after deleting 20: ");

    inorder(root);

    struct Node* result = search(root, 40);

    if (result != NULL)

        printf("\nElement 40 found in the BST.");

    else

        printf("\nElement 40 not found in the BST.");

return 0;

}
```



```
/tmp/TdDSOdEVFs.o
Inorder traversal of the BST: 20 30 40 50 60 70 80
Inorder traversal after deleting 20: 30 40 50 60 70 80
Element 40 found in the BST.

=== Code Execution Successful ===
```