*1.write a C program for 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));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

Node\* insert(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;

}

Node\* search(Node\* root, int data) {

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

return root;

}

if (data < root->data) {

return search(root->left, data);

} else {

return search(root->right, data);

}

}

void inOrderTraversal(Node\* root) {

if (root != NULL) {

inOrderTraversal(root->left);

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

inOrderTraversal(root->right);

}

}

void freeTree(Node\* root) {

if (root != NULL) {

freeTree(root->left);

freeTree(root->right);

free(root);

}

}

int main() {

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("In-order traversal of the BST:\n");

inOrderTraversal(root);

printf("\n");

int valueToSearch = 40;

Node\* searchResult = search(root, valueToSearch);

if (searchResult != NULL) {

printf("Node with value %d found in the BST.\n", valueToSearch);

} else {

printf("Node with value %d not found in the BST.\n", valueToSearch);

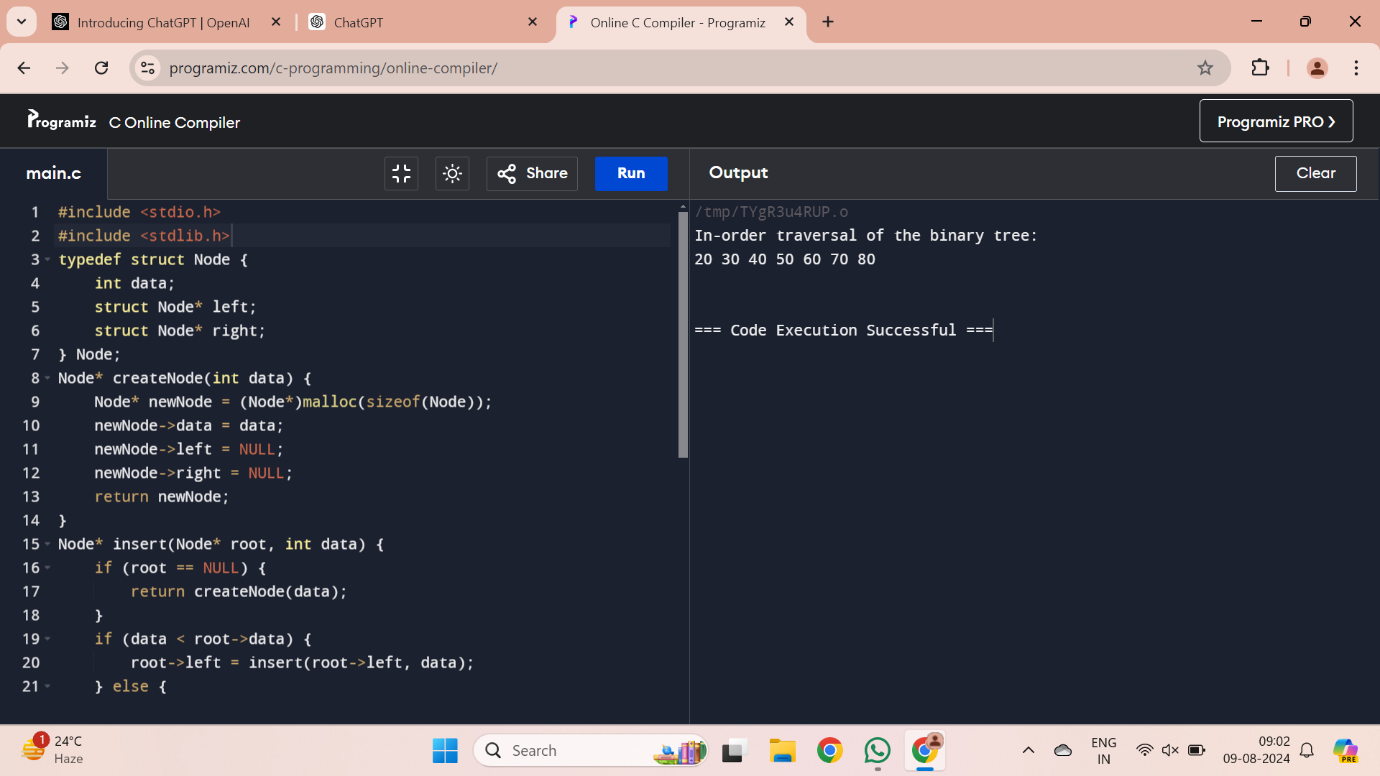
}

freeTree(root);

return 0;

}

**OUTPUT:**



*2. write a C program for binary search 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));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

Node\* insert(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;

}

Node\* search(Node\* root, int data) {

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

return root;

}

if (data < root->data) {

return search(root->left, data);

} else {

return search(root->right, data);

}

}

void inOrderTraversal(Node\* root) {

if (root != NULL) {

inOrderTraversal(root->left);

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

inOrderTraversal(root->right);

}

}

void freeTree(Node\* root) {

if (root != NULL) {

freeTree(root->left);

freeTree(root->right);

free(root);

}

}

int main() {

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("In-order traversal of the BST:\n");

inOrderTraversal(root);

printf("\n");

int valueToSearch = 40;

Node\* searchResult = search(root, valueToSearch);

if (searchResult != NULL) {

printf("Node with value %d found in the BST.\n", valueToSearch);

} else {

printf("Node with value %d not found in the BST.\n", valueToSearch);

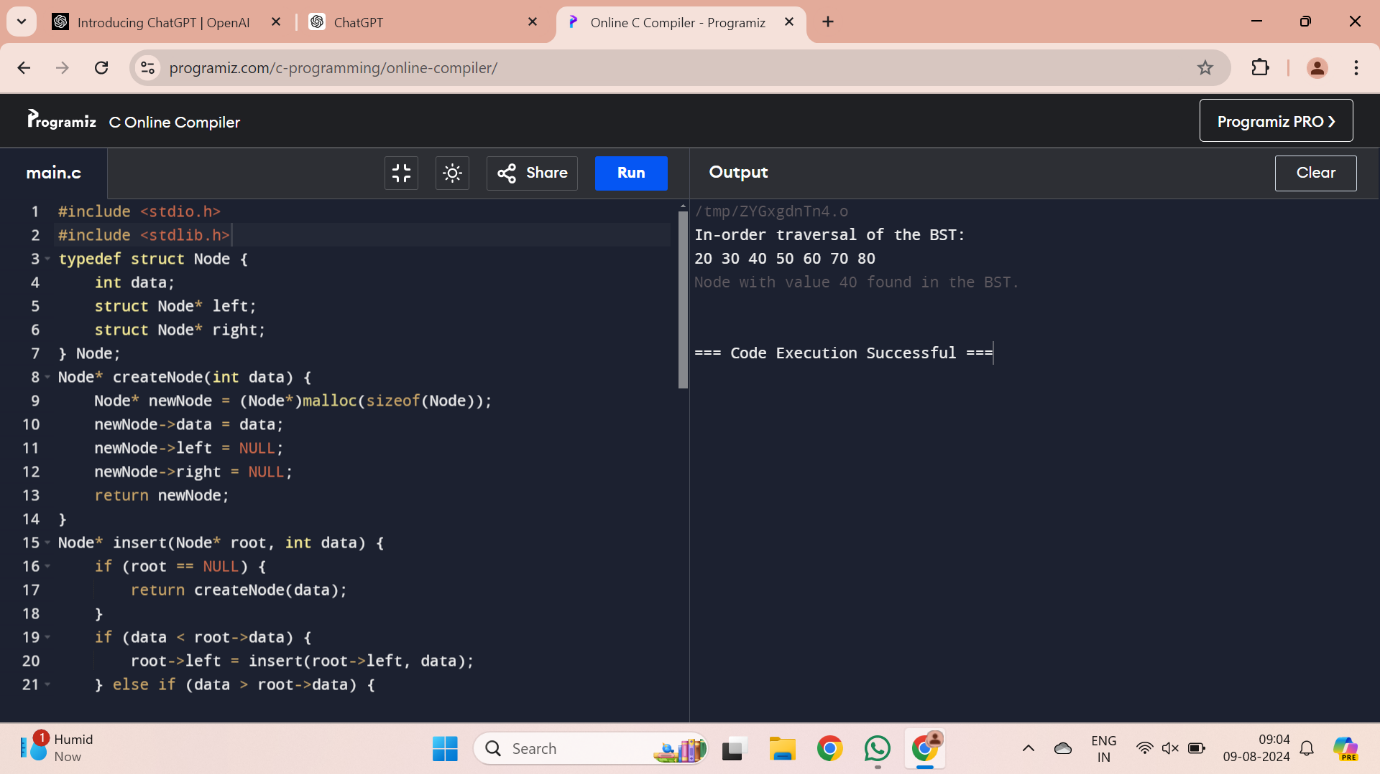
}

freeTree(root);

return 0;

}

**OUTPUT:**

****