BINARY TREE PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct BTnode
{
int keyVal;
struct BTnode *leftNode;
struct BTnode *rightNode;
};
struct BTnode *getNode(int value)
{
struct BTnode *newNode = malloc(sizeof(struct BTnode));
newNode->keyVal = value;
newNode->leftNode = NULL;
newNode->rightNode = NULL;
return newNode;
}
struct BTnode *insert(struct BTnode *rootNode, int value)
{
if(rootNode == NULL)
return getNode(value);
if(rootNode->keyVal < value)
rootNode->rightNode = insert(rootNode->rightNode,value);
else if(rootNode->keyVal > value)
rootNode->leftNode = insert(rootNode->leftNode,value);
return rootNode;
```

```
}
void insertorder(struct BTnode *rootNode)
{
if(rootNode == NULL)
return;
insertorder(rootNode->leftNode);
printf("%d ",rootNode->keyVal);
insertorder(rootNode->rightNode);
}
int main()
{
struct BTnode *rootNode = NULL;
rootNode = insert(rootNode,7);
rootNode = insert(rootNode,4);
rootNode = insert(rootNode,8);
rootNode = insert(rootNode,1);
rootNode = insert(rootNode,5);
rootNode = insert(rootNode,2);
rootNode = insert(rootNode,9);
rootNode = insert(rootNode,3);
insertorder(rootNode);
return 0;
}
```

OUTPUT

1 2 3 4 5 7 8 9

=== Code Execution Successful ===

BINARY SEARCH PROGRAM

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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;
}

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 if (value > root->data) {
                root->right = insert(root->right, value);
        }
        return root;
}

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);
        }
        return search(root->right, value);
}

struct Node* delete(struct Node* root, int value)
      {
        if (root == NULL)
        {
                return root;
        }
        if (value < root->data)
        {
                root->left = delete(root->left, value);
```

```
    } else if (value > root->data)
    {
        root->right = delete(root->right, value);
    } 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 = root->right;
        while (temp->left != NULL) {
            temp = temp->left;
        }
        root->data = temp->data;
        root->right = delete(root->right, temp->data);
    }
    return root;
}

void inorderTraversal(struct Node* root)
{
```

```
    if (root == NULL) {
        return;
    }
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

int main()
{
    struct Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);

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

    struct Node* searchResult = search(root, 60);
    if (searchResult != NULL) {
        printf("Element found: %d", searchResult->data);
    }
```

```
    else
        {
            printf("Element not found.");
        }

        root = delete(root, 20);
        root = delete(root, 30);
        root = delete(root, 50);

        printf("Inorder traversal after deletion of 20, 30, and 50: ");
        inorderTraversal(root);

        return 0;
}
```

OUTPUT

Inorder traversal of the BST: 20 30 40 50 60 70 80 Element found: 60Inorder traversal after deletion of 20, 30, and 50: 40 60 70 80

=== Code Execution Successful ===

BINARY TREE TRANSVERSAL

```
#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);
}

// preorderTraversal traversal
void preorderTraversal(struct node* root) {
    if (root == NULL) return;
    printf("%d ->", root->item);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

// postorderTraversal 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, 12);
```

```
    insertRight(root, 9);

    insertLeft(root->left, 5);
    insertRight(root->left, 6);

    printf("Inorder traversal \n");
    inorderTraversal(root);

    printf("\nPreorder traversal \n");
    preorderTraversal(root);

    printf("\nPostorder traversal \n");
    postorderTraversal(root);
}
```

OUTPUT

```
Inorder traversal
5 ->12 ->6 ->1 ->9 ->
Preorder traversal
1 ->12 ->5 ->6 ->9 ->
Postorder traversal
5 ->6 ->12 ->9 ->1 ->
```