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

// Structure for a Binary Search Tree (BST) Node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node with the given data
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a value 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 if (value > root->data) {
        root->right = insert(root->right, value);
    }

    return root;
}

// Function to perform inorder traversal of the BST
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function to perform preorder traversal of the BST
void preorderTraversal(struct Node* root) {
```

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

// Function to perform postorder traversal of the BST
void postorderTraversal(struct Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

// Function to search for a value 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);
    }

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

// Function to free the entire BST
void freeBST(struct Node* root) {
    if (root != NULL) {
        freeBST(root->left);
        freeBST(root->right);
        free(root);
    }
}

int main() {
    struct Node* root = NULL;
    int choice, key;

    // Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14,
7, 8, 5, 2
    int values[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2};
```

```
    int n = sizeof(values) / sizeof(values[0]);

    for (int i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    do {
        printf("\nMenu:\n");
        printf("1. Traverse the BST (Inorder)\n");
        printf("2. Traverse the BST (Preorder)\n");
        printf("3. Traverse the BST (Postorder)\n");
        printf("4. Search for an element\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 2:
                printf("Preorder Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Postorder Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Enter the element to search: ");
                scanf("%d", &key);
                if (search(root, key) != NULL) {
                    printf("Element %d is present in the BST.
\n", key);
                } else {
                    printf("Element %d is not present in the
BST.\n", key);
                }
                break;
            case 5:
                freeBST(root);
```

```
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid
option.\n");
        }
    } while (choice != 5);

    return 0;
}
```