1.Write a c program code for binary search

Code:

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

// Definition of the binary tree node
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Function prototypes
Node* createNode(int data);
Node* insert(Node* root, int data);
void inorderTraversal(Node* root);
void displayTree(Node* root, int level);

// Main function
int main() {
    Node* root = NULL;
    int choice, value;

    while (1) {
        printf("\nBinary Tree Operations:\n");
        printf("1. Insert Node\n");
        printf("2. In-order Traversal\n");
        printf("3. Display Tree\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Tree Structure:\n");
                displayTree(root, 0);
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}

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

// Function to insert a new node into the binary tree
Node* insert(Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

    // Insert into left or right subtree based on value
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else {
        root->right = insert(root->right, data);
    }
```

```c
    return root;
}

// Function for in-order traversal of the binary tree
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function to display the binary tree structure
void displayTree(Node* root, int level) {
    if (root == NULL) {
        return;
    }

    displayTree(root->right, level + 1);
    printf("\n");
    for (int i = 0; i < level; i++) {
        printf("    ");
    }
    printf("%d", root->data);
    displayTree(root->left, level + 1);
}
```

Output:

```
Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Display Tree
4. Exit
Enter your choice: 1
Enter value to insert: 10

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Display Tree
4. Exit
```

```
Enter your choice: 1
Enter value to insert: 5

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Display Tree
4. Exit
Enter your choice: 1
Enter value to insert: 15

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Display Tree
4. Exit
Enter your choice: 2
In-order Traversal: 5 10 15

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Display Tree
4. Exit
Enter your choice: 3
Tree Structure:

    15
  10
    5

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Display Tree
4. Exit
Enter your choice: 4
Exiting...
```

2.Write a c program code for Binary search tree(insertion, deletion and searching).

Code:

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

// Definition of the BST node
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Function prototypes
Node* createNode(int data);
Node* insert(Node* root, int data);
Node* deleteNode(Node* root, int data);
Node* search(Node* root, int data);
Node* findMin(Node* root);
void inorderTraversal(Node* root);
void displayTree(Node* root, int level);
void freeTree(Node* root);

// Main function
int main() {
    Node* root = NULL;
    int choice, value;

    while (1) {
        printf("\nBinary Search Tree Operations:\n");
        printf("1. Insert Node\n");
        printf("2. Delete Node\n");
        printf("3. Search Node\n");
        printf("4. In-order Traversal\n");
        printf("5. Display Tree\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                root = deleteNode(root, value);
                break;
            case 3:
                printf("Enter value to search: ");
                scanf("%d", &value);
                Node* result = search(root, value);
                if (result != NULL) {
                    printf("Value %d found in the tree.\n", value);
                } else {
                    printf("Value %d not found in the tree.\n", value);
                }
                break;
            case 4:
                printf("In-order Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 5:
                printf("Tree Structure:\n");
                displayTree(root, 0);
                break;
            case 6:
                printf("Exiting...\n");
                freeTree(root);
                exit(0);
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

```
// Function to create a new BST node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a new node into the BST
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;
}

// Function to find the minimum value node in the BST
Node* findMin(Node* root) {
    while (root && root->left != NULL) {
        root = root->left;
    }
    return root;
}

// Function to delete a node from the BST
Node* deleteNode(Node* root, int data) {
    if (root == NULL) {
        return root;
    }

    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        // Node with only one child or no child
        if (root->left == NULL) {
            Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            free(root);
            return temp;
        }

        // Node with two children: Get the inorder successor (smallest in the right subtree)
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}

// Function to search for a value in the BST
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);
    }
}

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

// Function to display the BST structure
void displayTree(Node* root, int level) {
    if (root == NULL) {
        return;
    }

    displayTree(root->right, level + 1);
    printf("\n");
    for (int i = 0; i < level; i++) {
        printf("   ");
    }
    printf("%d", root->data);
    displayTree(root->left, level + 1);
}

// Function to free the memory allocated for the BST
void freeTree(Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
```

Output:

```
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter value to insert: 10
```

```
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter value to insert: 5
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter value to insert: 15
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 4
In-order Traversal: 5 10 15
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 5
Tree Structure:

    15
  10
    5
```

```
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 3
Enter value to search: 10
Value 10 found in the tree.
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 2
Enter value to delete: 10
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 4
In-order Traversal: 5 15
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 5
Tree Structure:

    15
5
```

```
Binary Search Tree Operations:
1. Insert Node
2. Delete Node
3. Search Node
4. In-order Traversal
5. Display Tree
6. Exit
Enter your choice: 6
Exiting...
```

3.Write a c program code for Binary Tree Traversal (In order, Pre Order and Post order).

Code:

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

// Definition of the binary tree node
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Function prototypes
Node* createNode(int data);
void insert(Node** root, int data);
void inorderTraversal(Node* root);
void preorderTraversal(Node* root);
void postorderTraversal(Node* root);
void displayTree(Node* root, int level);
void freeTree(Node* root);

// Main function
int main() {
    Node* root = NULL;

    // Manually creating a binary tree for demonstration
    insert(&root, 10);
    insert(&root, 5);
    insert(&root, 15);
    insert(&root, 3);
    insert(&root, 7);
    insert(&root, 12);
    insert(&root, 18);

    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Pre-order Traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Post-order Traversal: ");
    postorderTraversal(root);
    printf("\n");

    printf("Tree Structure:\n");
    displayTree(root, 0);

    // Free the allocated memory
    freeTree(root);

    return 0;
}

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

// Function to insert a new node into the binary tree
```

```
void insert(Node** root, int data) {
    if (*root == NULL) {
        *root = createNode(data);
    } else {
        if (data < (*root)->data) {
            insert(&(*root)->left, data);
        } else {
            insert(&(*root)->right, data);
        }
    }
}

// Function for in-order traversal of the binary tree
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function for pre-order traversal of the binary tree
void preorderTraversal(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function for post-order traversal of the binary tree
void postorderTraversal(Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

// Function to display the binary tree structure
void displayTree(Node* root, int level) {
    if (root == NULL) {
        return;
    }

    displayTree(root->right, level + 1);
    printf("\n");
    for (int i = 0; i < level; i++) {
        printf("    ");
    }
    printf("%d", root->data);
    displayTree(root->left, level + 1);
}

// Function to free the memory allocated for the binary tree
void freeTree(Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
```

Output:

```
In-order Traversal: 3 5 7 10 12 15 18

Pre-order Traversal: 10 5 3 7 15 12 18

Post-order Traversal: 3 7 5 12 18 15 10

Tree Structure:
```

18

15

12

10

7

5

3