1)Binary tree.

Code

```
 #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* insertNode(Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
void preorderTraversal(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}
```

```
void postorderTraversal(Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
void freeTree(Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
    int main() {
    Node* root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);
    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n");
    printf("Pre-order Traversal: ");
    preorderTraversal(root);
    printf("\n");
    printf("Post-order Traversal: ");
    postorderTraversal(root);
    printf("\n");
    freeTree(root);
    return 0;
}
```

Output: In-order Traversal: 20 30 40 50 60 70 80
Pre-order Traversal: 50 30 20 40 70 60 80
Post-order Traversal: 20 40 30 60 80 70 50

2)Binary search tree(insertion,deletion and searching).

Code

```
#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);
    }
}
Node* findMin(Node* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}
```

```
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 {
        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* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
int main() {
    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:\n");
    inorderTraversal(root);
    printf("\n");
    int searchValue = 40;
```

```
    Node* searchResult = search(root, searchValue);
    if (searchResult != NULL) {
        printf("Node with value %d found.\n", searchValue);
    } else {
        printf("Node with value %d not found.\n", searchValue);
    }
    int deleteValue = 20;
    root = deleteNode(root, deleteValue);
    printf("Inorder traversal after deleting %d:\n", deleteValue);
    inorderTraversal(root);
    printf("\n");
    return 0;
}
```

Output:

```
Inorder traversal of the BST:
 20 30 40 50 60 70 80
Inorder traversal after deleting 20:
30 40 50 60 70 80
```

3)Binary tree traversal (in order, pre order and post order)

Code

```
#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;
}

void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
```

```
        inorderTraversal(root->right);
    }
}
void preorderTraversal(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}
void postorderTraversal(Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
int main() {
    Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);
    printf("In-order traversal:\n");
    inorderTraversal(root);
    printf("\n");
    printf("Pre-order traversal:\n");
    preorderTraversal(root);
    printf("\n");
    printf("Post-order traversal:\n");
    postorderTraversal(root);
    printf("\n");
    return 0;
}
```

Output:

```
In-order traversal:
 4 2 5 1 6 3 7
Pre-order traversal:
 1 2 4 5 3 6 7
 Post-order traversal:
 4 5 2 6 7 3 1
```