1.Binary Tree

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

// Define the structure for a node in the binary tree
Struct Node {
   Int data;
   Struct Node* left;
   Struct Node* right;
};

// Create a new node with given data
Struct Node* createNode(int data) {
   Struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}

// Insert a node into the binary tree
Struct Node* insertNode(struct Node* root, int data) {
   If (root == NULL) {
      Return createNode(data);
   }
```

```
    If (data < root->data) {
        Root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        Root->right = insertNode(root->right, data);
    }

    Return root;
}

// Search for a node in the binary tree
Struct Node* searchNode(struct Node* root, int data) {
    If (root == NULL || root->data == data) {
        Return root;
    }

    If (data < root->data) {
        Return searchNode(root->left, data);
    }

    Return searchNode(root->right, data);
}

// In-order traversal of the binary tree
Void inorderTraversal(struct Node* root) {
    If (root != NULL) {
        inorderTraversal(root->left);
        printf(“%d “, root->data);
        inorderTraversal(root->right);
    }
}

// Free the memory allocated for the binary tree
Void freeTree(struct Node* root) {
    If (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

Int main() {
    Struct 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”);

   int key = 40;
   if (searchNode(root, key) != NULL) {
      printf(“Node with value %d found.\n”, key);
   } else {
      Printf(“Node with value %d not found.\n”, key);
   }

   freeTree(root);

   return 0;
}
```

2.Binary search tree

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

// Define the structure for a node in the binary search tree
Struct Node {
   Int data;
   Struct Node* left;
   Struct Node* right;
};

// Create a new node with given data
```

```
Struct Node* createNode(int data) {
    Struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Insert a node into the binary search tree
Struct Node* insertNode(struct Node* root, int data) {
    If (root == NULL) {
        Return createNode(data);
    }

    If (data < root->data) {
        Root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        Root->right = insertNode(root->right, data);
    }

    Return root;
}

// Search for a node in the binary search tree
Struct Node* searchNode(struct Node* root, int data) {
    If (root == NULL || root->data == data) {
        Return root;
    }

    If (data < root->data) {
        Return searchNode(root->left, data);
    }

    Return searchNode(root->right, data);
}

// In-order traversal of the binary search tree
Void inorderTraversal(struct Node* root) {
    If (root != NULL) {
        inorderTraversal(root->left);
        printf(“%d “, root->data);
        inorderTraversal(root->right);
    }
}

// Free the memory allocated for the binary search tree
Void freeTree(struct Node* root) {
    If (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
```

```
}

Int main() {
  Struct 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”);

  int key = 40;
  if (searchNode(root, key) != NULL) {
    printf(“Node with value %d found.\n”, key);
  } else {
    Printf(“Node with value %d not found.\n”, key);
  }

  freeTree(root);

  return 0;
```

```
}
```

3. Binary Tree transversal

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

// Define the structure for a node in the binary tree
Struct Node {
   Int data;
   Struct Node* left;
   Struct Node* right;
};

// Create a new node with given data
Struct Node* createNode(int data) {
   Struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}

// Insert a node into the binary tree
Struct Node* insertNode(struct Node* root, int data) {
   If (root == NULL) {
      Return createNode(data);
```

```
    }

    If (data < root->data) {
        Root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        Root->right = insertNode(root->right, data);
    }

    Return root;
}

// In-order traversal of the binary tree
Void inorderTraversal(struct Node* root) {
    If (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Pre-order traversal of the binary tree
Void preorderTraversal(struct Node* root) {
    If (root != NULL) {
        Printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Post-order traversal of the binary tree
Void postorderTraversal(struct Node* root) {
    If (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf(“%d “, root->data);
    }
}

// Free the memory allocated for the binary tree
Void freeTree(struct Node* root) {
    If (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

Int main() {
    Struct 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;
}
```