1.BINARY TREE OF SEARCH:

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

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

struct Node* newNode(int item)
{
	struct Node* temp
		= (struct Node*)malloc(sizeof(struct Node));
	temp->key = item;
	temp->left = temp->right = NULL;
	return temp;
}


struct Node* search(struct Node* root, int key)
{


	if (root == NULL || root->key == key)
		return root;
```

```
    if (root->key < key)
        return search(root->right, key);

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

int main()
{
    struct Node* root = newNode(50);
    root->left = newNode(30);
    root->right = newNode(70);
    root->left->left = newNode(20);
    root->left->right = newNode(40);
    root->right->left = newNode(60);
    root->right->right = newNode(80);

    printf(search(root, 19) != NULL ? "Found\n"
                                    : "Not Found\n");
    printf(search(root, 80) != NULL ? "Found\n"
                                    : "Not Found\n");

    return 0;
}
```

2.BINARY TREE OF TRAVERSE:

```
// Tree traversal in C


#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);
}
```

3.BINARY TREE:

```
#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));
    if (newNode == NULL) {
        printf("Error allocating memory!\n");
        exit(1);
    }
    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 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, 70);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 60);
    insertNode(root, 80);


    freeTree(root);

    return 0;
}
```