Problem_1:

```
#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 creating a new node.\n");
exit(0);
}
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 printInOrder(Node* root) {
if (root != NULL) {
```

```
printInOrder(root->left);
printf("%d ", root->data);
printInOrder(root->right);
}
}
void printPreOrder(Node* root) {
if (root != NULL) {
printf("%d ", root->data);
printPreOrder(root->left);
printPreOrder(root->right);
}
}
void printPostOrder(Node* root) {
if (root != NULL) {
printPostOrder(root->left);
printPostOrder(root->right);
printf("%d ", root->data);
}
}
int main() {
Node* root = NULL;
root = insertNode(root, 8);
insertNode(root, 3);
insertNode(root, 10);
insertNode(root, 1);
insertNode(root, 6);
insertNode(root, 14);
insertNode(root, 4);
insertNode(root, 7);
insertNode(root, 13);
printf("In-order traversal of the binary tree is:\n");
printInOrder(root);
```

```
printf("\n");
printPostOrder(root);
printf("\n");
printPreOrder(root);
printf("\n");
return 0;
}
```

Problem 2:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
 int data;
 struct Node* left;
 struct Node* right;
};
struct Node* createNode(int value) {
 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode->data = value;
 newNode->left = NULL;
 newNode->right = NULL;
 return newNode;
}
struct Node* insertNode(struct Node* root, int value) {
 if (root == NULL)
 return createNode(value);
 if (value < root->data)
 root->left = insertNode(root->left, value);
 else if (value > root->data)
 root->right = insertNode(root->right, value);
 return root;
}
struct Node* minValueNode(struct Node* node) {
 struct Node* current = node;
 while (current && current->left != NULL)
 current = current->left;
 return current;
}
struct Node* deleteNode(struct Node* root, int value) {
 if (root == NULL)
 return root;
 if (value < root->data)
 root->left = deleteNode(root->left, value);
 else if (value > root->data)
 root->right = deleteNode(root->right, value);
 else {
 if (root->left == NULL) {
 struct Node* temp = root->right;
 free(root);
 return temp;
 }
 else if (root->right == NULL) {
 struct Node* temp = root->left;
 free(root);
 return temp;
 }
 struct Node* temp = minValueNode(root->right);
 root->data = temp->data;
 root->right = deleteNode(root->right, temp->data);
 }
 return root;
}
struct Node* searchNode(struct Node* root, int value) {
 if (root == NULL || root->data == value)
 return root;
 if (value < root->data)
 return searchNode(root->left, value);
 return searchNode(root->right, value);
}
void inorderTraversal(struct Node* root) {
 if (root != NULL) {
 inorderTraversal(root->left);
 printf("%d ", root->data);
 inorderTraversal(root->right);
 }
}
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("Inorder traversal of the BST: ");
 inorderTraversal(root);
 printf("\n");
 int key = 50;
 root = deleteNode(root, key);
 printf("Inorder traversal after deletion of %d: ", key);
 inorderTraversal(root);
 printf("\n");
 key = 30;
 struct Node* searchResult = searchNode(root, key);
 if (searchResult != NULL)
 printf("%d found in the BST\n", key);
 else
 printf("%d not found in the BST\n", key);
 return 0;
}
```

Problem 3:

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
 int val;
 struct TreeNode* left;
 struct TreeNode* right;
};
struct TreeNode* createNode(int value) {
 struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
 newNode->val = value;
 newNode->left = NULL;
 newNode->right = NULL;
 return newNode;
}
int height(struct TreeNode* root) {
 if (root == NULL)
 return 0;
 int leftHeight = height(root->left);
 int rightHeight = height(root->right);
 return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
}
int isBalanced(struct TreeNode* root) {
 if (root == NULL)
 return 1;
 int leftHeight = height(root->left);
 int rightHeight = height(root->right);
 if (abs(leftHeight - rightHeight) <= 1 && isBalanced(root->left) && isBalanced(root->right))
 return 1;
 return 0;
}
int main() {
 struct TreeNode* root = createNode(1);
 root->left = createNode(2);
 root->right = createNode(3);
 root->left->left = createNode(4);
 root->left->right = createNode(5);
 root->right->right = createNode(6);
 if (isBalanced(root))
 printf("The binary tree is height-balanced.\n");
 else
 printf("The binary tree is not height-balanced.\n");
 return 0;
}
```

Problem 4:

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
 int val;
 struct TreeNode* left;
 struct TreeNode* right;
};
struct TreeNode* createNode(int value) {
 struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
 newNode->val = value;
 newNode->left = NULL;
 newNode->right = NULL;
 return newNode;
}
```

```
struct TreeNode* lowestCommonAncestor(struct TreeNode* root, struct TreeNode* p, struct TreeNode*
q) {
 if (root == NULL || root == p || root == q)
 return root;
 struct TreeNode* leftLCA = lowestCommonAncestor(root->left, p, q);
 struct TreeNode* rightLCA = lowestCommonAncestor(root->right, p, q);
 if (leftLCA && rightLCA)
 return root;
 return (leftLCA != NULL) ? leftLCA : rightLCA;
}
int main() {
 struct TreeNode* root = createNode(3);
 root->left = createNode(5);
 root->right = createNode(1);
 root->left->left = createNode(6);
 root->left->right = createNode(2);
 root->right->left = createNode(0);
 root->right->right = createNode(8);
 root->left->right->left = createNode(7);
 root->left->right->right = createNode(4);
 struct TreeNode* p = root->left;
 struct TreeNode* q = root->right;

 struct TreeNode* lca = lowestCommonAncestor(root, p, q);
 printf("Lowest Common Ancestor of %d and %d is: %d\n", p->val, q->val, lca->val);
 return 0;
}
```