Tutorial 7: Trees

ELEC 278: Fundamentals of Information Structures

The learning goals for Tutorial 7 are:

- Understand binary search trees (BST) and BST operations.
- Practice inserting and deleting nodes iteratively and recursively.

Problem 1. The code below contains recursive functions to insert and delete nodes from a BST. Rewrite the insert and delete operations using iterative functions.

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

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

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

// Recursive function to insert a new node
struct Node* insertRecursive(struct Node* node, int data) {
    if (node == NULL) return newNode(data);
    if (data < node->data)
        node->left = insertRecursive(node->left, data);
    else
        node->right = insertRecursive(node->right, data);
    return node;
}

// Recursive function to find the minimum node
struct Node* findMin(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

// Recursive function to delete a node
struct Node* deleteRecursive(struct Node* root, int key) {
    if (root == NULL) return root;
    if (key < root->data)
```

```
        root->left = deleteRecursive(root->left, key);
    else if (key > root->data)
        root->right = deleteRecursive(root->right, key);
    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 = findMin(root->right);
        root->data = temp->data;
        root->right = deleteRecursive(root->right, temp->data);
    }
    return root;
}

// Function to print BST nodes in inorder traversal
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

int main() {
    struct Node* root = NULL;
    root = insertRecursive(root, 50);
    root = insertRecursive(root, 30);
    root = insertRecursive(root, 20);
    root = insertRecursive(root, 40);
    root = insertRecursive(root, 70);
    root = insertRecursive(root, 60);
    root = insertRecursive(root, 80);

    printf("Inorder traversal of the BST: ");
    inorder(root);
    printf("\n");

    root = deleteRecursive(root, 20);
    root = deleteRecursive(root, 30);
    root = deleteRecursive(root, 50);

    printf("Inorder traversal after deletion: ");
    inorder(root);
    printf("\n");

    return 0;
}
```

Solution.

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

struct Node {
```

```
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Iterative function to insert a new node
struct Node* insertIterative(struct Node* root, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    if (root == NULL) return newNode;

    struct Node* parent = NULL;
    struct Node* current = root;
    while (current != NULL) {
        parent = current;
        if (data < current->data)
            current = current->left;
        else
            current = current->right;
    }
    if (data < parent->data)
        parent->left = newNode;
    else
        parent->right = newNode;
    return root;
}

// Iterative function to delete a node
struct Node* deleteIterative(struct Node* root, int key) {
    struct Node* parent = NULL;
    struct Node* current = root;
    while (current != NULL && current->data != key) {
        parent = current;
        if (key < current->data)
            current = current->left;
        else
            current = current->right;
    }
    if (current == NULL) return root; // Key not found

    if (current->left == NULL || current->right == NULL) {
        struct Node* newCurr;
        if (current->left == NULL)
            newCurr = current->right;
        else
            newCurr = current->left;

        if (parent == NULL) return newCurr;

        if (current == parent->left)
```

```
            parent->left = newCurr;
        else
            parent->right = newCurr;
    } else {
        struct Node* p = NULL;
        struct Node* temp;

        temp = current->right;
        while (temp->left != NULL) {
            p = temp;
            temp = temp->left;
        }

        if (p != NULL) p->left = temp->right;
        else current->right = temp->right;

        current->data = temp->data;
        current = temp;
    }
    free(current);
    return root;
}

// Function to print BST nodes in inorder traversal
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

int main() {
    struct Node* root = NULL;
    root = insertIterative(root, 50);
    root = insertIterative(root, 30);
    root = insertIterative(root, 20);
    root = insertIterative(root, 40);
    root = insertIterative(root, 70);
    root = insertIterative(root, 60);
    root = insertIterative(root, 80);

    printf("Inorder traversal of the BST: ");
    inorder(root);
    printf("\n");

    root = deleteIterative(root, 20);
    root = deleteIterative(root, 30);
    root = deleteIterative(root, 50);

    printf("Inorder traversal after deletion: ");
    inorder(root);
    printf("\n");

    return 0;
}
```