Binary search tree

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

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

typedef struct tree {
    Node* root;
} Tree;

Node* create_node(int data) {
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

Node* find_min(Node* node) {
    while (node->left != NULL) {
        node = node->left;
    }
    return node;
}

Node* find_max(Node* node) {
    while (node->right != NULL) {
        node = node->right;
    }
    return node;
}

void insert_node(Tree* t, int data) {
    Node* new_node = create_node(data);
    if (t->root == NULL) {
        t->root = new_node;
        return;
    }
    Node* current = t->root;
    while (1) {
        if (data < current->data) {
            if (current->left == NULL) {
                current->left = new_node;
```

```
                return;
            }
            current = current->left;
        } else {
            if (current->right == NULL) {
                current->right = new_node;
                return;
            }
            current = current->right;
        }
    }
}

void Inorder_traversal(Node* node) {
    if (node == NULL) {
        return;
    }
    Inorder_traversal(node->left);
    printf("%d ", node->data);
    Inorder_traversal(node->right);
}

void Preorder_traversal(Node* node) {
    if (node == NULL) {
        return;
    }
    printf("%d ", node->data);
    Preorder_traversal(node->left);
    Preorder_traversal(node->right);
}

void Postorder_traversal(Node* node) {
    if (node == NULL) {
        return;
    }
    Postorder_traversal(node->left);
    Postorder_traversal(node->right);
    printf("%d ", node->data);
}

void search(Node* root, int data) {
    Node* current = root;

    while (current != NULL)
    {
        if (data == current->data)
        {
            printf("Data %d is found\n", data);
```

```
            return;
        }
        else if (data < current->data)
        {
            current = current->left;
        }
        else
        {
            current = current->right;
        }
    }

    printf("Data %d is not found\n",data);
}

int searchRecursive(Node* current, int data) {
    if (current == NULL) {
        printf("Data %d not found\n",data);
        return 0;
    }

    if (data == current->data) {
        printf("Data %d is found\n", data);
        return 1;
    }

    if (data < current->data) {
        return searchRecursive(current->left, data);
    } else {
        return searchRecursive(current->right, data);
    }
}

int countNodesRecursive(Node* current) {
    if (current == NULL) {
        return 0;
    }

    // Count the current node and recursively count nodes in left and right
subtrees
    int leftCount = countNodesRecursive(current->left);
    int rightCount = countNodesRecursive(current->right);

    return 1 + leftCount + rightCount;
}

int countLeafNodesRecursive(Node* current) {
    if (current == NULL) {
```

```
        return 0;
    }

    if (current->left == NULL && current->right == NULL) {
        // The current node is a leaf node
        return 1;
    }

    // Recursively count leaf nodes in left and right subtrees
    int leftCount = countLeafNodesRecursive(current->left);
    int rightCount = countLeafNodesRecursive(current->right);

    return leftCount + rightCount;
}

int calculateHeightRecursive(Node* current) {
    if (current == NULL) {
        return 0;
    }

    // Recursively calculate the height of the left and right subtrees
    int leftHeight = calculateHeightRecursive(current->left);
    int rightHeight = calculateHeightRecursive(current->right);

    // The height of the tree is the maximum of the left and right subtree
heights, plus 1 for the current node.
    return 1 + (leftHeight > rightHeight ? leftHeight : rightHeight);
}

void mirrorTreeRecursive(Node* current) {
    if (current == NULL) {
        return;
    }

    // Swap the left and right subtrees
    Node* temp = current->left;
    current->left = current->right;
    current->right = temp;

    // Recursively mirror the left and right subtrees
    mirrorTreeRecursive(current->left);
    mirrorTreeRecursive(current->right);
}

Node* deleteNodeNonRecursive(Node* root, int key) {
    Node* current = root;
    Node* parent = NULL;
```

```c
    // Search for the node to delete
    while (current != NULL && current->data != key) {
        parent = current;
        if (key < current->data) {
            current = current->left;
        } else {
            current = current->right;
        }
    }

    // If the node is not found, return the original root
    if (current == NULL) {
        return root;
    }

    // Handle three cases for deletion

    // Case 1: Node with no child
    if (current->left == NULL && current->right == NULL) {
        if (parent == NULL) {
            free(current);
            return NULL;  // Root node is deleted
        } else if (parent->left == current) {
            parent->left = NULL;
        } else {
            parent->right = NULL;
        }
        free(current);
    }
    // Case 2: Node with one child
    else if (current->left == NULL) {
        Node* temp = current->right;
        if (parent == NULL) {
            free(current);
            return temp;
        }
        if (parent->left == current) {
            parent->left = temp;
        } else {
            parent->right = temp;
        }
        free(current);
    } else if (current->right == NULL) {
        Node* temp = current->left;
        if (parent == NULL) {
            free(current);
            return temp;
        }
```

```
        if (parent->left == current) {
            parent->left = temp;
        } else {
            parent->right = temp;
        }
        free(current);
    }
    // Case 3: Node with two children
    else {
        Node* successor = find_min(current->right);
        int successorData = successor->data;
        deleteNodeNonRecursive(root, successorData); // Recursively delete the
successor
        current->data = successorData; // Copy the successor data to the
current node
    }

    return root;
}


int main() {
    Tree t;
    t.root = NULL;

    int choice;
    int data;
    int del;

    while (1) {
        printf("\nBinary Tree Menu:\n");
        printf("1. Insert a node\n");
        printf("2. Delete a node\n");
        printf("3. Inorder traversal\n");
        printf("4. Preorder traversal\n");
        printf("5. Postorder traversal\n");
        printf("6. Find minimum and maximum values\n");
        printf("7. Search for a value\n");
        printf("8. Count nodes\n");
        printf("9. Count leaf nodes\n");
        printf("10. Calculate height\n");
        printf("11. Mirror the tree\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
```

```
                scanf("%d", &data);
                insert_node(&t, data);
                break;
             case 2:
            printf("Enter the data to be deleted: ");
            scanf("%d", &del);
            t.root = deleteNodeNonRecursive(t.root, del); // Update the root
with the result of the deletion
            printf("Tree after deletion: ");
            Inorder_traversal(t.root);
        break;
        case 3:
                printf("Inorder traversal: ");
                Inorder_traversal(t.root);
                printf("\n");
                break;
            case 4:
                printf("Preorder traversal: ");
                Preorder_traversal(t.root);
                printf("\n");
                break;
            case 5:
                printf("Postorder traversal: ");
                Postorder_traversal(t.root);
                printf("\n");
                break;
            case 6:
                // Find minimum and maximum values here
              // Node* min_node = find_min(t.root);
        //Node* max_node = find_max(t.root);

        printf("Minimum value in the tree: %d\n",  find_min(t.root)->data);
        printf("Maximum value in the tree: %d\n", find_max(t.root)->data);

                break;
            case 7:
                // Search for a value here
                printf("Enter the value to search for: ");
                scanf("%d", &data);
                search(t.root, data);
                searchRecursive(t.root, data);
                break;
            case 8:
                printf("The number of nodes is %d\n",
countNodesRecursive(t.root));
                break;
            case 9:
```

```
                printf("The number of leaf nodes is %d\n",
countLeafNodesRecursive(t.root));
                break;
            case 10:
                printf("The height of the tree is %d\n",
calculateHeightRecursive(t.root));
                break;
            case 11:
                mirrorTreeRecursive(t.root);
                printf("After mirroring: ");
                Inorder_traversal(t.root);
                printf("\n");
                break;

            case 0:
                printf("Exiting the program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    }

    return 0;
}
```

```
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 1
Enter the value to insert: 10

Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 1
Enter the value to insert: 12
```

```
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 1
Enter the value to insert: 43

Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 1
Enter the value to insert: 23
```

```
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 3
Inorder traversal: 10 12 23 43

Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 4
Preorder traversal: 10 12 43 23
```

```
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 5
Postorder traversal: 23 43 12 10

Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 6
Minimum value in the tree: 10
Maximum value in the tree: 43
```

```
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 7
Enter the value to search for: 43
Data 43 is found
Data 43 is found

Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 8
The number of nodes is 4
```

```
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 9
The number of leaf nodes is 1

Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 10
The height of the tree is 4
```

```
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 2
Enter the data to be deleted: 23
Tree after deletion: 10 12 43
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 11
After mirroring: 43 12 10
```

```
Tree after deletion: 10 12 43
Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 11
After mirroring: 43 12 10

Binary Tree Menu:
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Find minimum and maximum values
7. Search for a value
8. Count nodes
9. Count leaf nodes
10. Calculate height
11. Mirror the tree
0. Exit
Enter your choice: 0
Exiting the program.
PS C:\Users\Mark Lopes\Desktop\college\ds>
```