Practical 9: Implementation of Tree and Searching

Name: Sutariya Savankumar

Roll no: MA065

1. **Write a program to do the following operations.**
   - **Create a Binary Tree by collecting information from users.**
   - **Create a Binary Search Tree by collecting information from users.**
   - **Traverse the created trees using**
     - **preorder**
     - **postorder**
     - **inorder**
     - **levelorder**
   - **Search Element in Binary Search Tree**
   - **Find Internal Nodes, External Nodes, Total Nodes and Height of Tree**

Code

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
};
struct Node *create(int data)
{
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
struct Node *insertNode(struct Node *root, int data)
{
    if (root == NULL)
    {
        return create(data);
    }
    if (data < root->data)
    {
        root->left = insertNode(root->left, data);
```

```
    }
    else if (data > root->data)
    {
        root->right = insertNode(root->right, data);
    }
    return root;
}
struct Node *createBinaryTree()
{
    int data;
    struct Node *root = NULL;
    printf("Enter root node data: ");
    scanf("%d", &data);
    root = create(data);
    printf("Enter left child of %d : ", data);
    scanf("%d", &data);
    if (data != -1)
    {
        root->left = create(data);
        printf("Enter left child of %d : ", root->left->data);
        scanf("%d", &data);
        if (data != -1)
        {
            root->left->left = create(data);
        }
        printf("Enter right child of %d : ", root->left->data);
        scanf("%d", &data);
        if (data != -1)
        {
            root->left->right = create(data);
        }
    }
    printf("Enter right child of %d : ", data);
    scanf("%d", &data);
    if (data != -1)
    {
        root->right = create(data);
        printf("Enter left child of %d : ", root->right->data);
        scanf("%d", &data);
        if (data != -1)
        {
            root->right->left = create(data);
        }
        printf("Enter right child of %d : ", root->right->data);
        scanf("%d", &data);
        if (data != -1)
        {
            root->right->right = create(data);
```

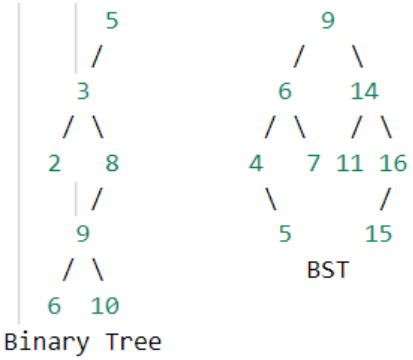
```
        }
    }
    return root;
}
void preorder(struct Node *root)
{
    if (root == NULL)
        return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}
void inorder(struct Node *root)
{
    if (root == NULL)
        return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
void postorder(struct Node *root)
{
    if (root == NULL)
        return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}
void levelorderTraversal(struct Node *root)
{
    if (root == NULL)
        return;
    struct Node **queue = (struct Node **)malloc(sizeof(struct Node *) * 100);
    int front = -1;
    int rear = -1;
    queue[++rear] = root;
    while (front < rear)
    {
        struct Node *node = queue[++front];
        printf("%d ", node->data);
        if (node->left != NULL)
            queue[++rear] = node->left;
        if (node->right != NULL)
            queue[++rear] = node->right;
    }
}
struct Node *search(struct Node *root, int key)
{
```

```
    if (root == NULL || root->data == key)
        return root;
    if (root->data < key)
        return search(root->right, key);
    else
        return search(root->left, key);
}
void countNodesAndHeight(struct Node *root, int *internalNodes, int *externalNodes, int
*totalNodes, int *height)
{
    if (root == NULL)
        return;
    (*totalNodes)++;
    if (root->left == NULL && root->right == NULL)
        (*externalNodes)++;
    else
        (*internalNodes)++;
    int leftHeight = 0;
    int rightHeight = 0;
    countNodesAndHeight(root->left, internalNodes, externalNodes, totalNodes, &leftHeight);
    countNodesAndHeight(root->right, internalNodes, externalNodes, totalNodes, &rightHeight);
    *height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
}
int main()
{
    struct Node *root = NULL;
    int choice, data, key, internalNodes = 0, externalNodes = 0, totalNodes = 0, height = 0;
    do
    {
        printf("\n--- Binary Tree and Binary Search Tree Operations ---\n");
        printf("1. Create Binary Tree\n");
        printf("2. Create Binary Search Tree\n");
        printf("3. Preorder Traversal\n");
        printf("4. Inorder Traversal\n");
        printf("5. Postorder Traversal\n");
        printf("6. Levelorder Traversal\n");
        printf("7. Search Element in Binary Search Tree\n");
        printf("8. Count Internal Nodes, External Nodes, Total Nodes, and Height of Tree\n");
        printf("9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            root = createBinaryTree();
            break;
        case 2:
            printf("Enter root node data: ");
```

```
            scanf("%d", &data);
            root = create(data);
            while (1)
            {
                printf("Enter data to be inserted : in stop then -1 ");
                scanf("%d", &data);
                if (data == -1)
                    break;
                insertNode(root, data);
            }
            break;
        case 3:
            printf("Preorder Traversal: ");
            preorder(root);
            break;
        case 4:
            printf("Inorder Traversal: ");
            inorder(root);
            break;
        case 5:
            printf("Postorder Traversal: ");
            postorder(root);
            break;
        case 6:
            printf("Levelorder Traversal: ");
            levelorderTraversal(root);
            break;
        case 7:
            printf("Enter element to search: ");
            scanf("%d", &key);
            if (search(root, key) != NULL)
                printf("Element found in the tree.\n");
            else
                printf("Element not found in the tree.\n");
            break;
        case 8:
            countNodesAndHeight(root, &internalNodes, &externalNodes, &totalNodes, &height);
            printf("Total number of nodes in the tree: %d\n", totalNodes);
            printf("Number of internal nodes in the tree: %d\n", internalNodes);
            printf("Number of external nodes in the tree: %d\n", externalNodes);
            printf("Height of the tree: %d\n", height);
            break;
        case 9:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice!\n");
            exit(0);
```

```
            break;
        }
    } while (choice != 9);
    return 0;
}
```

Output




```
Savan@Savan MINGW64 /c/Drive/Study/MCA/DSsubmission/MA065_Savan (master)
$ gcc prac09-01.c -o p1

Savan@Savan MINGW64 /c/Drive/Study/MCA/DSsubmission/MA065_Savan (master)
$ ./p1

--- Binary Tree and Binary Search Tree Operations ---
1. Create Binary Tree
2. Create Binary Search Tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Levelorder Traversal
7. Search Element in Binary Search Tree
8. Count Internal Nodes, External Nodes, Total Nodes, and Height of Tree
9. Exit
Enter your choice: 1
Enter root node data: 5
Enter left child of 5 : 3
Enter left child of 3 : 2
Enter right child of 3 : 8
Enter right child of 8 : 9
Enter left child of 9 : 6
Enter right child of 9 : 10

--- Binary Tree and Binary Search Tree Operations ---
1. Create Binary Tree
2. Create Binary Search Tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Levelorder Traversal
7. Search Element in Binary Search Tree
8. Count Internal Nodes, External Nodes, Total Nodes, and Height of Tree
9. Exit
Enter your choice: 3
Preorder Traversal: 5 3 2 8 9 6 10
```

```
--- Binary Tree and Binary Search Tree Operations ---
1. Create Binary Tree
2. Create Binary Search Tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Levelorder Traversal
7. Search Element in Binary Search Tree
8. Count Internal Nodes, External Nodes, Total Nodes, and Height of Tree
9. Exit
Enter your choice: 6
Levelorder Traversal: 5 3 9 2 8 6 10
 --- Binary Tree and Binary Search Tree Operations ---
1. Create Binary Tree
2. Create Binary Search Tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Levelorder Traversal
7. Search Element in Binary Search Tree
8. Count Internal Nodes, External Nodes, Total Nodes, and Height of Tree
9. Exit
Enter your choice: 2
Enter root node data: 9
Enter data to be inserted : in stop then -1 6
Enter data to be inserted : in stop then -1 4
Enter data to be inserted : in stop then -1 7
Enter data to be inserted : in stop then -1 5
Enter data to be inserted : in stop then -1 14
Enter data to be inserted : in stop then -1 11
Enter data to be inserted : in stop then -1 16
Enter data to be inserted : in stop then -1 15
Enter data to be inserted : in stop then -1 -1

--- Binary Tree and Binary Search Tree Operations ---
1. Create Binary Tree
2. Create Binary Search Tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Levelorder Traversal
7. Search Element in Binary Search Tree
8. Count Internal Nodes, External Nodes, Total Nodes, and Height of Tree
9. Exit
Enter your choice: 3
Preorder Traversal: 9 6 4 5 7 14 11 16 15
```

```
Preorder Traversal: 9 6 4 5 7 14 11 16 15
--- Binary Tree and Binary Search Tree Operations ---
1. Create Binary Tree
2. Create Binary Search Tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Levelorder Traversal
7. Search Element in Binary Search Tree
8. Count Internal Nodes, External Nodes, Total Nodes, and Height of Tree
9. Exit
Enter your choice: 7
Enter element to search: 7
Element found in the tree.

--- Binary Tree and Binary Search Tree Operations ---
1. Create Binary Tree
2. Create Binary Search Tree
3. Preorder Traversal
4. Inorder Traversal
5. Postorder Traversal
6. Levelorder Traversal
7. Search Element in Binary Search Tree
8. Count Internal Nodes, External Nodes, Total Nodes, and Height of Tree
9. Exit
Enter your choice: 8
Total number of nodes in the tree: 9
Number of internal nodes in the tree: 5
Number of external nodes in the tree: 4
Height of the tree: 4
```

2. **Write a program to do the following operations.**
   - **Create an array from user input.**
   - **Search Element in an array using linear search - prints iteration done to find the element**
   - **Search Element in an array using binary search - prints iteration done to find the element**

Code

```
#include <stdio.h>
int LinearSearch(int arr[], int size, int key)
{
    for (int i = 0; i < size; i++)
    {
        if (arr[i] == key)
            return i;
    }
    return -1;
}
int BinarySearch(int arr[], int size, int key)
{
    int start = 0;
```

```
    int end = size - 1;
    while (start <= end)
    {
        int mid = start + (end - start) / 2;
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] > key)
            end = mid - 1;
        else
            start = mid + 1;
    }
    return start;
}
int main()
{
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of the array: ");
    for (int i = 0; i < size; i++)
        scanf("%d", &arr[i]);
    int key;
    printf("Enter the element to be searched: ");
    scanf("%d", &key);
    printf("\nLinear Search\n");
    int index = LinearSearch(arr, size, key);
    if (index == -1)
        printf("Element not found in the array");
    else
        printf("Element found at index %d", index);
    printf("\nBinary Search\n");
    index = BinarySearch(arr, size, key);
    if (index == -1)
        printf("Element not found in the array");
    else
        printf("Element found at index %d", index);
    return 0;
}
```

Output

```
Savan@Savan MINGW64 /c/Drive/Study/MCA/DSsubmission/MA065_Savan/Lab9 (master)
$ ./p2
Enter the size of the array: 5
Enter the elements of the array: 2 4 5 3 1
Enter the element to be searched: 4

Linear Search
Element found at index 1
Binary Search
Element found at index 1
```