CS221-L Data Structures and Algorithms Lab

Lab # 08

Submitted by:

- **Shayan Rizwan** [2024585]

Submitted to: Sir Adnan Haider

Semester: 3rd

Faculty of Computer Science and Engineering

GIK Institute of Engineering Sciences and Technology

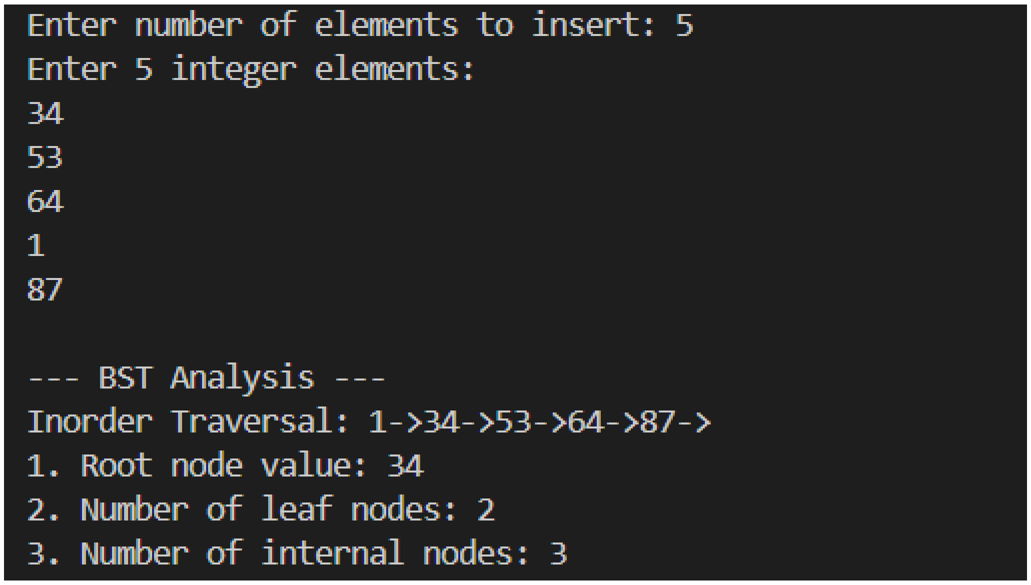
Task # 1:

Write a C++ program to implement a Binary Search Tree (BST) that allows insertion of integer elements.
After constructing the BST, the program should compute and display the following information:
1. The total number of leaf nodes (nodes that do not have any children).
2. The total number of internal nodes (nodes that have at least one child).
3. The value of the root node of the BST.

Code:

```cpp
#include <iostream>

using namespace std;

// Node structure for Binary Search Tree
struct TreeNode {
  int data;           // Value stored in node
  TreeNode* left;     // Pointer to left child
  TreeNode* right;    // Pointer to right child

  // Constructor to initialize node
  TreeNode(int value) {
    data = value;
    left = right = NULL;
  }
};

// Class for Binary Search Tree operations
class BST {
private:
  TreeNode* root;     // Root of the BST

  // Helper function to insert a value recursively
  TreeNode* insertRecursive(TreeNode* node, int value) {
    // If current position is empty, create new node
    if (node == NULL) {
      return new TreeNode(value);
    }

    // BST property: smaller values go left, larger values go right
    if (value < node->data) {
      node->left = insertRecursive(node->left, value);
    } else if (value > node->data) {
      node->right = insertRecursive(node->right, value);
    }

    return node;
  }

  int countLeafNodes(TreeNode* node) {
    // Base case: empty node
    if (node == NULL) {
      return 0;
    }

    // Base case: leaf node (no children)
    if (node->left == NULL && node->right == NULL) {
      return 1;
    }

    // Recursive case: node has children
    // Count leaves in left subtree + count leaves in right subtree
    return countLeafNodes(node->left) + countLeafNodes(node->right);
  }

  // Helper function to count internal nodes recursively
  int countInternalNodes(TreeNode* node) {
    // Base case: empty node or leaf node
    if (node == NULL || (node->left == NULL && node->right == NULL)) {
      return 0;
    }

    // Current node is internal (has at least one child)
    // Count it and recursively count internal nodes in subtrees
    return 1 + countInternalNodes(node->left) + countInternalNodes(node-
>right);
  }

    void InorderTraversal(TreeNode* root) {
  if (root == NULL) {
    return;
  }

  InorderTraversal(root->left);
  cout << root->data << "->";
  InorderTraversal(root->right);

}

  void DeleteTree(TreeNode*& root) { // pass the pointer by reference, so
that the original pointer can be modified (avoid segmentation faults)
    if (root == NULL) {
      return;
    }

    DeleteTree(root->left);
    DeleteTree(root->right);
    delete root;
    root = NULL; // set the pointer to NULL so that the dangling pointer does
not point to any garbage value;
  }

public:
  // Constructor initializes empty tree
  BST() {
    root = NULL;
  }

  // Public method to insert value into BST
  void insert(int value) {
    root = insertRecursive(root, value);
  }

  // Public method to count leaf nodes
  int getLeafCount() {
    return countLeafNodes(root);
  }

  // Public method to count internal nodes
  int getInternalCount() {
    return countInternalNodes(root);
  }

  // Public method to get root value
  int getRootValue() {
    if (root == NULL) {
      cout << "Tree is empty!" << endl;
      return -1; // Return -1 for empty tree
    }
    else {
      return root->data;
    }

  }

  void getInorderTraversal() {
    InorderTraversal(root);
  }



};

int main() {
  BST tree;
  int n, value;

  // Get number of elements from user
  cout << "Enter number of elements to insert: ";
  cin >> n;

  // Insert elements into BST
  cout << "Enter " << n << " integer elements:" << endl;
  for (int i = 0; i < n; i++) {
    cin >> value;
    tree.insert(value);
  }

  // Display required information
  cout << "\n--- BST Analysis ---" << endl;
  cout << "Inorder Traversal: ";
  tree.getInorderTraversal();
  cout << endl;
  cout << "1. Root node value: " << tree.getRootValue() << endl;
  cout << "2. Number of leaf nodes: " << tree.getLeafCount() << endl;
  cout << "3. Number of internal nodes: " << tree.getInternalCount() << endl;

  return 0;
}
```

Output:



```
Enter number of elements to insert: 5
Enter 5 integer elements:
34
53
64
1
87

--- BST Analysis ---
Inorder Traversal: 1->34->53->64->87->
1. Root node value: 34
2. Number of leaf nodes: 2
3. Number of internal nodes: 3
```

Task # 2:

Write a C++ program that constructs a Binary Search Tree (BST) by inserting a series of integer elements and then converts the BST into its mirror image.
The mirror image of a BST is obtained by recursively swapping the left and right subtrees of every node.

Example:

Input:

Number of nodes: 7
Elements: 50 30 70 20 40 60 80

Output:

Inorder traversal of original BST: 20 30 40 50 60 70 80
Inorder traversal of mirror image BST: 80 70 60 50 40 30 20

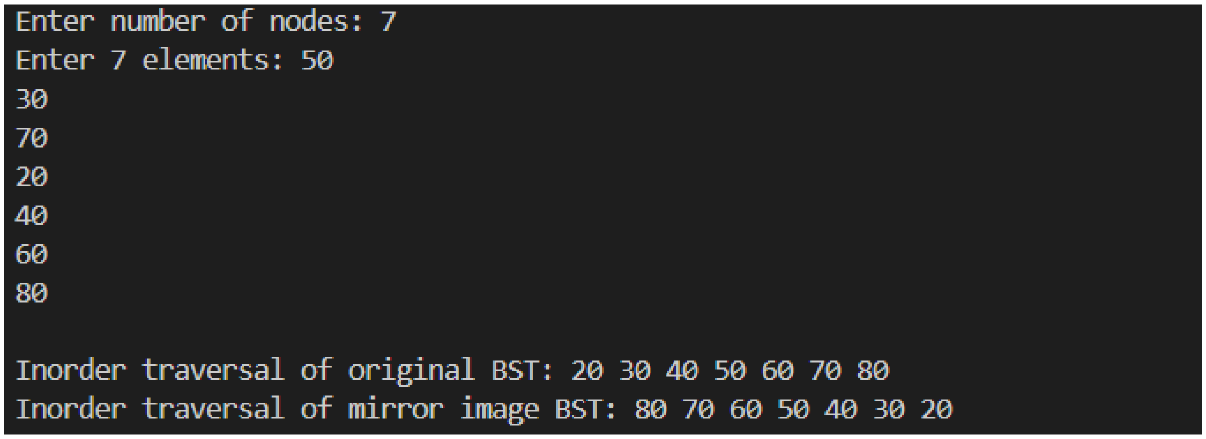
Code:

```
#include <iostream>
using namespace std;

// Node structure for Binary Search Tree
struct TreeNode {
  int data;           // Value stored in node
  TreeNode* left;     // Pointer to left child
  TreeNode* right;    // Pointer to right child

  // Constructor to initialize node
  TreeNode(int value) {
    data = value;
    left = right = NULL;
  }
};

// Class for Binary Search Tree operations
class BST {
private:
  TreeNode* root;     // Root of the BST

  // Insert a value recursively
  TreeNode* insertRecursive(TreeNode* node, int value) {
    // If current position is empty, create new node
    if (node == NULL) {
```

```
    return new TreeNode(value);
  }

  // BST property: smaller values go left, larger values go right
  if (value < node->data) {
    node->left = insertRecursive(node->left, value);
  } else if (value > node->data) {
    node->right = insertRecursive(node->right, value);
  }

  return node;
}

// Helper function for inorder traversal (Left-Root-Right)
void inorderRecursive(TreeNode* node) {
  if (node == NULL) {
    return;
  }

  // Recursively traverse left subtree
  inorderRecursive(node->left);

  // Visit current node
  cout << node->data << " ";

  // Recursively traverse right subtree
  inorderRecursive(node->right);
}

// Helper function to create mirror image recursively
TreeNode* mirrorRecursive(TreeNode* node) {
  // Base case: empty node
  if (node == NULL) {
    return NULL;
  }

  // Recursively mirror left and right subtrees
  TreeNode* mirroredLeft = mirrorRecursive(node->left);
  TreeNode* mirroredRight = mirrorRecursive(node->right);

  // Swap the mirrored subtrees
  node->left = mirroredRight;
  node->right = mirroredLeft;
```

```
    return node;
  }

public:
  // Constructor initializes empty tree
  BST() {
    root = NULL;
  }

  // Public method to insert value into BST
  void insert(int value) {
    root = insertRecursive(root, value);
  }

  // Public method for inorder traversal
  void inorderTraversal() {
    inorderRecursive(root);
    cout << endl;
  }

  // Public method to create mirror image
  void mirror() {
    root = mirrorRecursive(root);
  }

  // Getter for root (not used in main but useful for testing)
  TreeNode* getRoot() {
    return root;
  }
};

int main() {
  BST tree;
  int n, value;

  // Get number of elements from user
  cout << "Enter number of nodes: ";
  cin >> n;

  // Insert elements into BST
  cout << "Enter " << n << " elements: ";
  for (int i = 0; i < n; i++) {
    cin >> value;
    tree.insert(value);
  }

  // Display original BST using inorder traversal
  cout << "\nInorder traversal of original BST: ";
  tree.inorderTraversal();

  // Create mirror image of BST
  tree.mirror();

  // Display mirror BST using inorder traversal
  cout << "Inorder traversal of mirror image BST: ";
  tree.inorderTraversal();

  return 0;
}
```

Output:



```
Enter number of nodes: 7
Enter 7 elements: 50
30
70
20
40
60
80

Inorder traversal of original BST: 20 30 40 50 60 70 80
Inorder traversal of mirror image BST: 80 70 60 50 40 30 20
```