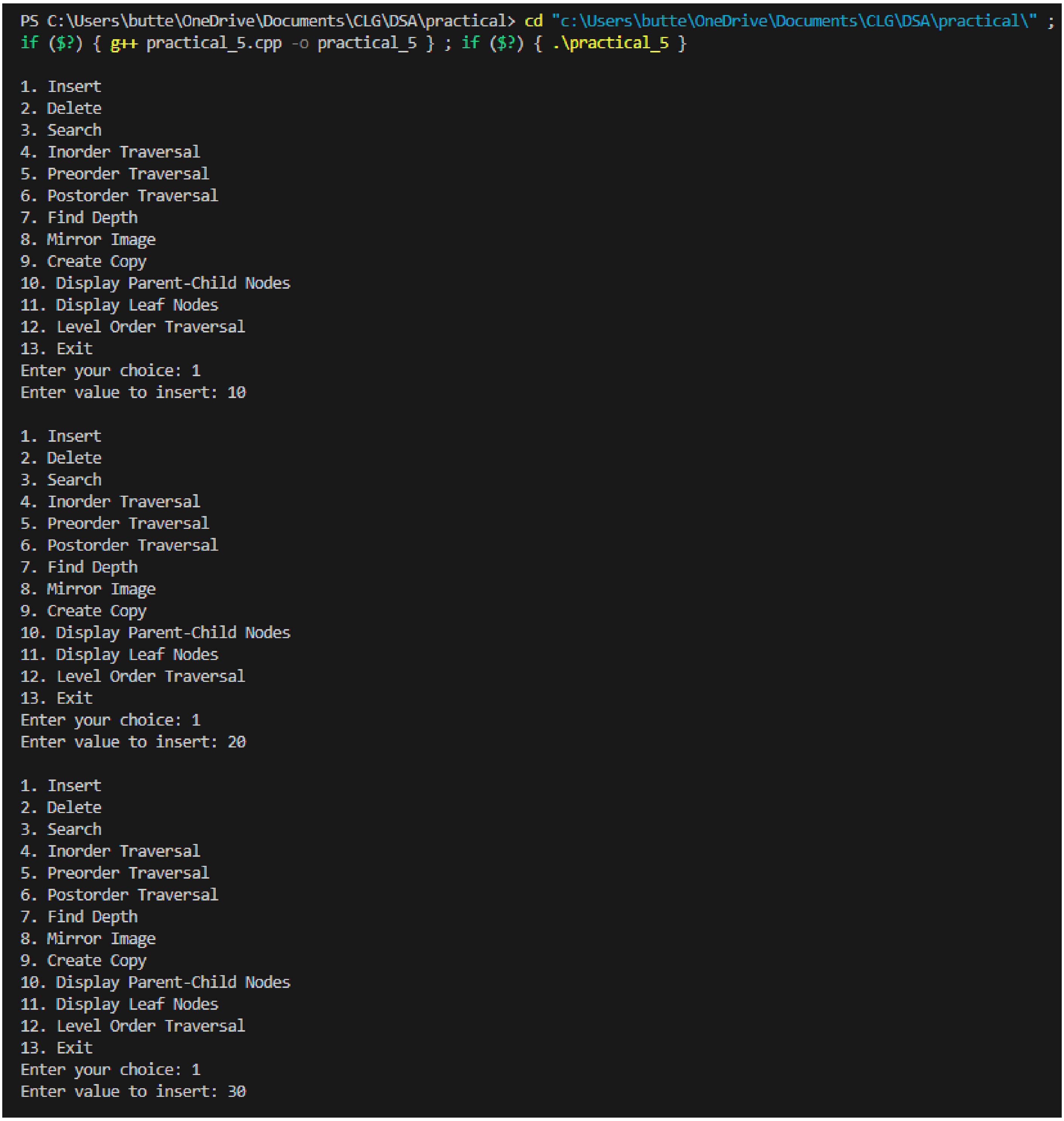
Practical 5
Source Code:-

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

struct TreeNode {
  int data;
  TreeNode* left;
  TreeNode* right;
  TreeNode(int value) {
    data = value;
    left = nullptr;
    right = nullptr;
  }
};

class BinarySearchTree {
public:
  BinarySearchTree() {
    root = nullptr;
  }

  void insert(int value) {
    root = insertHelper(root, value);
  }

  void deleteNode(int value) {
    root = deleteHelper(root, value);
  }

  TreeNode* search(int value) {
    return searchHelper(root, value);
  }

  void inorderTraversal() {
    inorderTraversalHelper(root);
    cout << endl;
  }

  void preorderTraversal() {
    preorderTraversalHelper(root);
    cout << endl;
  }

  void postorderTraversal() {
    postorderTraversalHelper(root);
    cout << endl;
  }

  int findDepth() {
    return findDepthHelper(root);
  }

  void mirrorImage() {
    mirrorImageHelper(root);
  }

  BinarySearchTree* createCopy() {
    return createCopyHelper(root);
  }

  void displayParentChildNodes() {
    displayParentChildNodesHelper(root);
  }

  void displayLeafNodes() {
    displayLeafNodesHelper(root);
    cout << endl;
  }

  void levelOrderTraversal() {
    levelOrderTraversalHelper(root);
    cout << endl;
  }

private:
  TreeNode* root;

  TreeNode* insertHelper(TreeNode* node, int value) {
    if (node == nullptr) {
        return new TreeNode(value);
    }
    if (value < node->data) {
        node->left = insertHelper(node->left, value);
    } else if (value > node->data) {
        node->right = insertHelper(node->right, value);
    } else {
        cout << "Duplicate value: " << value << endl;
    }
    return node;
  }

  TreeNode* searchHelper(TreeNode* node, int value) {
    if (node == nullptr || node->data == value) {
        return node;
    }
    if (value < node->data) {
        return searchHelper(node->left, value);
    } else {
        return searchHelper(node->right, value);
    }
  }

  TreeNode* deleteHelper(TreeNode* node, int value) {
```

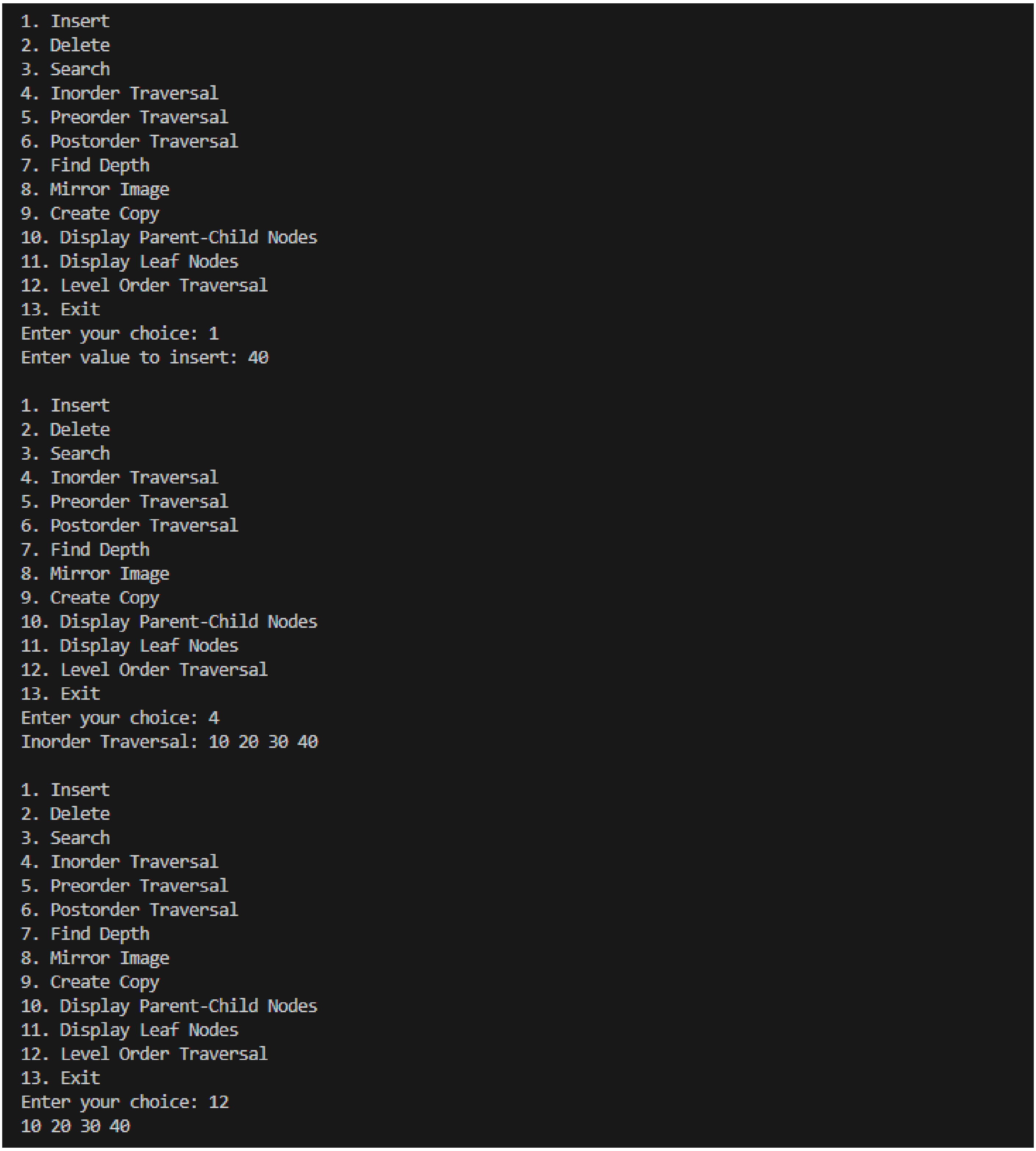
```
    if (node == nullptr) {
        return node;
    }
    if (value < node->data) {
        node->left = deleteHelper(node->left, value);
    } else if (value > node->data) {
        node->right = deleteHelper(node->right, value);
    } else {
        if (node->left == nullptr) {
            return node->right;
        } else if (node->right == nullptr) {
            return node->left;
        } else {
            TreeNode* temp = findMin(node->right);
            node->data = temp->data;
            node->right = deleteHelper(node->right, temp->data);
        }
    }
    return node;
}

TreeNode* findMin(TreeNode* node) {
    while (node->left != nullptr) {
        node = node->left;
    }
    return node;
}

void inorderTraversalHelper(TreeNode* node) {
    if (node != nullptr) {
        inorderTraversalHelper(node->left);
        cout << node->data << " ";
        inorderTraversalHelper(node->right);
    }
}

void preorderTraversalHelper(TreeNode* node) {
    if (node != nullptr) {
        cout << node->data << " ";
        preorderTraversalHelper(node->left);
        preorderTraversalHelper(node->right);
    }
}

void postorderTraversalHelper(TreeNode* node) {
    if (node != nullptr) {
        postorderTraversalHelper(node->left);
        postorderTraversalHelper(node->right);
        cout << node->data << " ";
    }
}

int findDepthHelper(TreeNode* node) {
    if (node == nullptr) {
        return 0;
    }
    int leftDepth = findDepthHelper(node->left);
    int rightDepth = findDepthHelper(node->right);
    return max(leftDepth, rightDepth) + 1;
}

void mirrorImageHelper(TreeNode* node) {
    if (node == nullptr) {
        return;
    }
    TreeNode* temp = node->left;
    node->left = node->right;
    node->right = temp;
    mirrorImageHelper(node->left);
    mirrorImageHelper(node->right);
}

BinarySearchTree* createCopyHelper(TreeNode* node) {
    if (node == nullptr) {
        return nullptr;
    }
    BinarySearchTree* newTree = new BinarySearchTree();
    newTree->root = createCopyNode(node);
    return newTree;
}

TreeNode* createCopyNode(TreeNode* node) {
    if (node == nullptr) {
        return nullptr;
    }
    TreeNode* newNode = new TreeNode(node->data);
    newNode->left = createCopyNode(node->left);
    newNode->right = createCopyNode(node->right);
    return newNode;
}

void displayParentChildNodesHelper(TreeNode* node, TreeNode* parent = nullptr) {
    if (node == nullptr) {
        return;
    }
    if (parent != nullptr) {
        cout << "Parent: " << parent->data << ", Child: " << node->data << endl;
    }
    displayParentChildNodesHelper(node->left, node);
    displayParentChildNodesHelper(node->right, node);
}

void displayLeafNodesHelper(TreeNode* node) {
    if (node == nullptr) {
        return;
    }
```

```
        if (node->left == nullptr && node->right == nullptr) {
            cout << node->data << " ";
        }
        displayLeafNodesHelper(node->left);
        displayLeafNodesHelper(node->right);
    }

    void levelOrderTraversalHelper(TreeNode* node) {
        if (node == nullptr) {
            return;
        }
        queue<TreeNode*> q;
        q.push(node);
        while (!q.empty()) {
            TreeNode* current = q.front();
            q.pop();
            cout << current->data << " ";
            if (current->left != nullptr) {
                q.push(current->left);
            }
            if (current->right != nullptr) {
                q.push(current->right);
            }
        }
    }
};

int main() {
    BinarySearchTree bst;
    int choice, value;
    TreeNode* foundNode = nullptr;  // Declared outside of switch
    BinarySearchTree* copy = nullptr;  // Declared outside of switch
    while (true) {
        cout << "\n1. Insert\n2. Delete\n3. Search\n4. Inorder Traversal\n5. Preorder Traversal\n6. Postorder Traversal\n7. Find Depth\n8. Mirror Image\n9. Create Copy\n10. Display Parent-Child Nodes\n11. Display Leaf Nodes\n12. Level Order Traversal\n13. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter value to insert: ";
                cin >> value;
                bst.insert(value);
                break;
            case 2:
                cout << "Enter value to delete: ";
                cin >> value;
                bst.deleteNode(value);
                break;
            case 3:
                cout << "Enter value to search: ";
                cin >> value;
                foundNode = bst.search(value);
                if (foundNode != nullptr) {
                    cout << "Found node: " << foundNode->data << endl;
                } else {
                    cout << "Node not found." << endl;
                }
                break;
            case 4:
                cout << "Inorder Traversal: ";
                bst.inorderTraversal();
                break;
            case 5:
                cout << "Preorder Traversal: ";
                bst.preorderTraversal();
                break;
            case 6:
                cout << "Postorder Traversal: ";
                bst.postorderTraversal();
                break;
            case 7:
                cout << "Depth of the tree: " << bst.findDepth() << endl;
                break;
            case 8:
                bst.mirrorImage();
                cout << "Tree mirrored." << endl;
                break;
            case 9:
                copy = bst.createCopy();
                cout << "Copy created." << endl;
                break;
            case 10:
                bst.displayParentChildNodes();
                break;
            case 11:
                bst.displayLeafNodes();
                break;
            case 12:
                bst.levelOrderTraversal();
                break;
            case 13:
                exit(0);
            default:
                cout << "Invalid choice!" << endl;
        }
    }

    return 0;
}
```

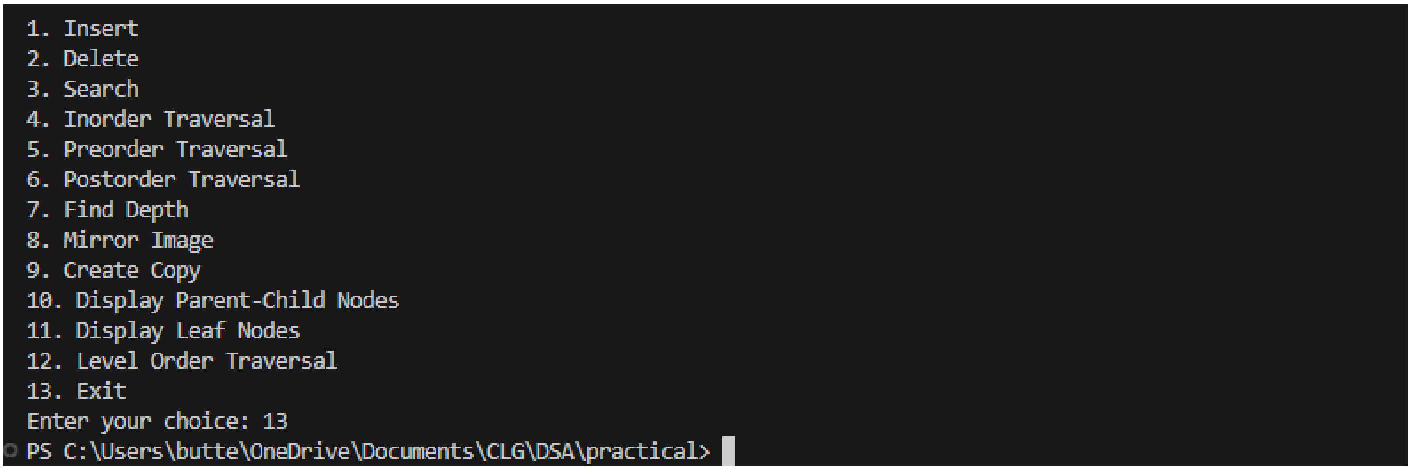
Output:-



```
PS C:\Users\butte\OneDrive\Documents\CLG\DSA\practical> cd "c:\Users\butte\OneDrive\Documents\CLG\DSA\practical\" ;
if ($?) { g++ practical_5.cpp -o practical_5 } ; if ($?) { .\practical_5 }

1. Insert
2. Delete
3. Search
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Find Depth
8. Mirror Image
9. Create Copy
10. Display Parent-Child Nodes
11. Display Leaf Nodes
12. Level Order Traversal
13. Exit
Enter your choice: 1
Enter value to insert: 10

1. Insert
2. Delete
3. Search
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Find Depth
8. Mirror Image
9. Create Copy
10. Display Parent-Child Nodes
11. Display Leaf Nodes
12. Level Order Traversal
13. Exit
Enter your choice: 1
Enter value to insert: 20

1. Insert
2. Delete
3. Search
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Find Depth
8. Mirror Image
9. Create Copy
10. Display Parent-Child Nodes
11. Display Leaf Nodes
12. Level Order Traversal
13. Exit
Enter your choice: 1
Enter value to insert: 30
```



```
1. Insert
2. Delete
3. Search
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Find Depth
8. Mirror Image
9. Create Copy
10. Display Parent-Child Nodes
11. Display Leaf Nodes
12. Level Order Traversal
13. Exit
Enter your choice: 1
Enter value to insert: 40

1. Insert
2. Delete
3. Search
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Find Depth
8. Mirror Image
9. Create Copy
10. Display Parent-Child Nodes
11. Display Leaf Nodes
12. Level Order Traversal
13. Exit
Enter your choice: 4
Inorder Traversal: 10 20 30 40

1. Insert
2. Delete
3. Search
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Find Depth
8. Mirror Image
9. Create Copy
10. Display Parent-Child Nodes
11. Display Leaf Nodes
12. Level Order Traversal
13. Exit
Enter your choice: 12
10 20 30 40
```



```
1. Insert
2. Delete
3. Search
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Find Depth
8. Mirror Image
9. Create Copy
10. Display Parent-Child Nodes
11. Display Leaf Nodes
12. Level Order Traversal
13. Exit
Enter your choice: 13
PS C:\Users\butte\OneDrive\Documents\CLG\DSA\practical>
```