**Name: Affan Shaikh**

**Roll no: COB227**

**Code :**

#include <iostream>

using namespace std;

structBstnode {

    int data;

    Bstnode\* left = NULL;

    Bstnode\* right = NULL;

};

class Btree {

public:

    Bstnode\* root;

    Btree() {

        root = NULL;

    }

    Bstnode\* GetNewNode(intin\_data) {

        Bstnode\* ptr = new Bstnode();

        ptr->data = in\_data;

        return ptr;

    }

    Bstnode\* insert(Bstnode\* temp, intin\_data) {

        if (temp == NULL) {

            return GetNewNode(in\_data);

        }

        if (in\_data< temp->data) {

            temp->left = insert(temp->left, in\_data);

        } else {

            temp->right = insert(temp->right, in\_data);

        }

        return temp;

    }

    void addNode() {

        int value;

        cout<< "Enter value to insert into the tree: ";

        cin>> value;

        root = insert(root, value);

        cout<< "Node " << value << " inserted successfully!" <<endl;

    }

    intfindDepth(Bstnode\* temp) {

        if (temp == NULL)

            return 0;

        return max(findDepth(temp->left), findDepth(temp->right)) + 1;

    }

    void findMinValue() {

        if (root == NULL) {

            cout<< "The tree is empty!" <<endl;

            return;

        }

        Bstnode\* temp = root;

        while (temp->left != NULL) {

            temp = temp->left;

        }

        cout<< "Minimum value in the tree: " << temp->data <<endl;

    }

    void mirrorTree(Bstnode\* temp) {

        if (temp == NULL)

            return;

        swap(temp->left, temp->right);

        mirrorTree(temp->left);

        mirrorTree(temp->right);

    }

    void mirror() {

        if (root == NULL) {

            cout<< "The tree is empty!" <<endl;

            return;

        }

        mirrorTree(root);

        cout<< "Tree mirrored successfully!" <<endl;

    }

    bool search(Bstnode\* temp, intin\_data) {

        if (temp == NULL)

            return false;

        if (temp->data == in\_data)

            return true;

        if (in\_data< temp->data)

            return search(temp->left, in\_data);

        return search(temp->right, in\_data);

    }

    void searchValue() {

        int value;

        cout<< "Enter value to search: ";

        cin>> value;

        if (search(root, value)) {

            cout<< "Value " << value << " found in the tree." <<endl;

        } else {

            cout<< "Value " << value << " not found in the tree." <<endl;

        }

    }

    void inorder(Bstnode\* temp) {

        if (temp == NULL)

            return;

        inorder(temp->left);

        cout<< temp->data << " ";

        inorder(temp->right);

    }

    void display() {

        if (root == NULL) {

            cout<< "The tree is empty!" <<endl;

            return;

        }

        cout<< "Inorder traversal of the tree: ";

        inorder(root);

        cout<<endl;

    }

};

intmain() {

    Btree tree;

    int choice;

    while (true) {

        cout<< "\nMenu:\n"

             << "1. Insert new node\n"

             << "2. Find number of nodes in the longest path (depth)\n"

             << "3. Find minimum data value in the tree\n"

             << "4. Mirror the tree\n"

             << "5. Search for a value\n"

             << "6. Display tree\n"

             << "7. Exit\n"

             << "Enter your choice: ";

        cin>> choice;

        switch (choice) {

            case 1:

                tree.addNode();

                break;

            case 2:

                cout<< "Number of nodes in the longest path (depth): " <<tree.findDepth(tree.root) <<endl;

                break;

            case 3:

                tree.findMinValue();

                break;

            case 4:

                tree.mirror();

                break;

            case 5:

                tree.searchValue();

                break;

            case 6:

                tree.display();

                break;

            case 7:

                cout<< "Exiting program!" <<endl;

                return 0;

            default:

                cout<< "Invalid choice. Please try again!" <<endl;

        }

    }

    return 0;

}

**Output :**

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 1

Enter value to insert into the tree: 6

Node 6 inserted successfully!

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 1

Enter value to insert into the tree: 7

Node 7 inserted successfully!

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 1

Enter value to insert into the tree: 8

Node 8 inserted successfully!

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 1

Enter value to insert into the tree: 3

Node 3 inserted successfully!

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 2

Number of nodes in the longest path (depth): 3

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 3

Minimum value in the tree: 3

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 4

Tree mirrored successfully!

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 5

Enter value to search: 6

Value 6 found in the tree.

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 6

Inorder traversal of the tree: 8 7 6 3

Menu:

1. Insert new node

2. Find number of nodes in the longest path (depth)

3. Find minimum data value in the tree

4. Mirror the tree

5. Search for a value

6. Display tree

7. Exit

Enter your choice: 7

Exiting program!