TASK 1:

#include <iostream>

using namespace std;

int multiply(int a, int b)

{

    if (a == 0 || b == 0)

    {

        return 0;

    }

    else if (b == 1)

    {

        return a;

    }

    else if (a == 1)

    {

        return b;

    }

    else if (b > 0)

    {

        return a + multiply(a, b - 1);

    }

    else

    {

        return -a + multiply(a, b + 1);

    }

}

int main()

{

    int a, b;

    cout << "Enter the value of a\n";

    cin >> a;

    cout << "Enter the value of b\n";

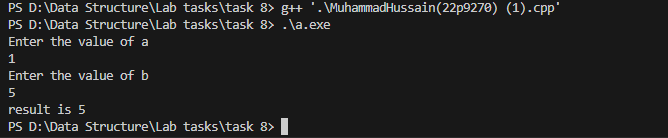
    cin >> b;

    cout<< "result is "<<multiply(a,b)<<endl;

    return 0;

}

OUTPUT:



TASK 2:

#include <iostream>

using namespace std;

class nodes

{

public:

    int data;

    nodes \*left, \*right;

    nodes()

    {

        data = 0;

        left = NULL;

        right = NULL;

    }

    nodes(int d)

    {

        data = d;

        left = NULL;

        right = NULL;

    }

};

class BST

{

public:

    nodes \*root, \*n1;

    BST()

    {

        root = NULL;

    }

    bool isTreeEmpty()

    {

        if (root == NULL)

        {

            return true;

        }

        return false;

    }

    void insert(int d)

    {

        n1 = new nodes(d);

        if (root == NULL)

        {

            root = n1;

            cout << "Root node inserted\n";

        }

        else

        {

            nodes \*temp = root;

            while (temp != NULL)

            {

                if (n1->data == temp->data)

                {

                    cout << "No Duplicates are allowed! Insert another value\n";

                    return;

                }

                else if ((n1->data < temp->data) && (temp->left == NULL))

                {

                    temp->left = n1;

                    cout << "Value inserted to the left\n";

                    break;

                }

                else if (n1->data < temp->data)

                {

                    temp = temp->left;

                }

                else if ((n1->data > temp->data) && (temp->right == NULL))

                {

                    temp->right = n1;

                    cout << "Value inserted the right\n";

                    break;

                }

                else if (n1->data > temp->data)

                {

                    temp = temp->right;

                }

            }

        }

    }

    void inOrderTraversal(nodes \*node)

    {

        if (node == NULL)

            return;

        inOrderTraversal(node->left);

        cout << node->data << " ";

        inOrderTraversal(node->right);

    }

    void preOrderTraversal(nodes \*node)

    {

        if (node == NULL)

            return;

        cout << node->data << " ";

        preOrderTraversal(node->left);

        preOrderTraversal(node->right);

    }

    void postOrderTraversal(nodes \*node)

    {

        if (node == NULL)

            return;

        postOrderTraversal(node->left);

        postOrderTraversal(node->right);

        cout << node->data << " ";

    }

    int findSmallestValue(nodes \*node)

    {

        if (node->left == NULL)

            return node->data;

        return findSmallestValue(node->left);

    }

    int countNodes(nodes \*node)

    {

        if (node == NULL)

            return 0;

        return 1 + countNodes(node->left) + countNodes(node->right);

    }

    ~BST()

    {

        delete root;

    }

};

int main()

{

    BST OBJ;

    int option;

    do

    {

        cout << "\nSelect one of the following Options\n";

        cout << "1: For insertion\n";

        cout << "2: For In order traversal\n";

        cout << "3: For Pre Order\n";

        cout << "4: For Post Order\n";

        cout << "5: For Finding smallest value in tree\n";

        cout << "6: For counting Number of nodes in the tree\n";

        cout << "7: For exiting the code\n";

        cin >> option;

        if (option == 1)

        {

            int data;

            cout << "Enter the value of the node you want to Insert\n";

            cin >> data;

            OBJ.insert(data);

        }

        else if (option == 2)

        {

            if (OBJ.isTreeEmpty())

            {

                cout << "Tree is empty\n";

            }

            else

            {

                cout << "In Order Traversal: ";

                OBJ.inOrderTraversal(OBJ.root);

                cout << endl;

            }

        }

        else if (option == 3)

        {

            if (OBJ.isTreeEmpty())

            {

                cout << "Tree is empty\n";

            }

            else

            {

                cout << "Pre Order Traversal: ";

                OBJ.preOrderTraversal(OBJ.root);

                cout << endl;

            }

        }

        else if (option == 4)

        {

            if (OBJ.isTreeEmpty())

            {

                cout << "Tree is empty\n";

            }

            else

            {

                cout << "Post Order Traversal: ";

                OBJ.postOrderTraversal(OBJ.root);

                cout << endl;

            }

        }

        else if (option == 5)

        {

            if (OBJ.isTreeEmpty())

            {

                cout << "Tree is empty\n";

            }

            else

            {

                cout << "Smallest value in the tree: " << OBJ.findSmallestValue(OBJ.root) << endl;

            }

        }

        else if (option == 6)

        {

            if (OBJ.isTreeEmpty())

            {

                cout << "Tree is empty\n";

            }

            else

            {

                cout << "Number of nodes in the tree: " << OBJ.countNodes(OBJ.root) << endl;

            }

        }

        else if (option == 7)

        {

            cout << "Exiting the program\n";

        }

        else

        {

            cout << "Invalid option! Please select a valid option\n";

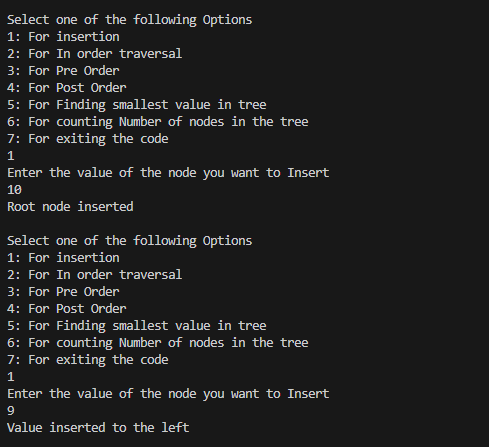
        }

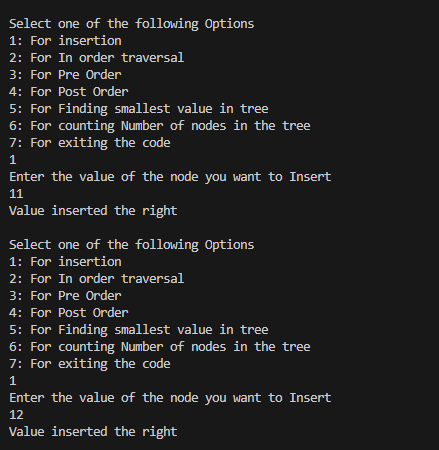
    } while (option != 7);

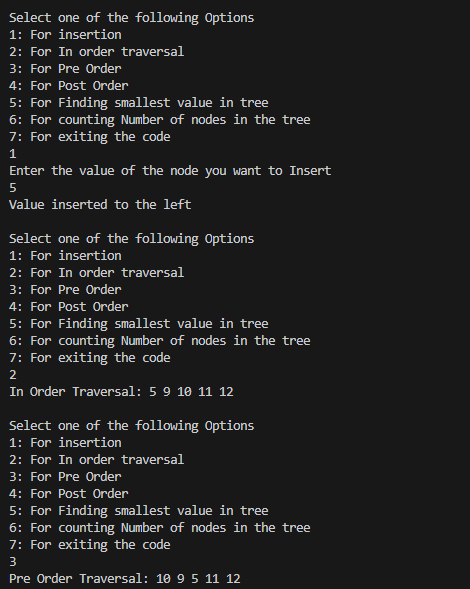
    return 0;

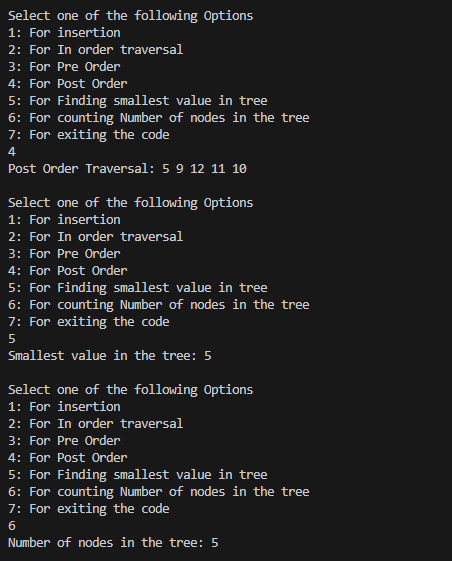
}

OUTPUTS:









TASK 3:

#include <iostream>

using namespace std;

class nodes

{

public:

    int data;

    nodes \*left, \*right;

    nodes()

    {

        data = 0;

        left = NULL;

        right = NULL;

    }

    nodes(int d)

    {

        data = d;

        left = NULL;

        right = NULL;

    }

};

class BST

{

public:

    nodes \*root, \*n1;

    BST()

    {

        root = NULL;

    }

    bool isTreeEmpty()

    {

        if (root == NULL)

        {

            return true;

        }

        return false;

    }

    void insert(int d)

    {

        n1 = new nodes(d);

        if (root == NULL)

        {

            root = n1;

            cout << "Root node inserted\n";

        }

        else

        {

            nodes \*temp = root;

            while (temp != NULL)

            {

                if (n1->data == temp->data)

                {

                    cout << "No Duplicates are allowed! Insert another value\n";

                    return;

                }

                else if ((n1->data < temp->data) && (temp->left == NULL))

                {

                    temp->left = n1;

                    cout << "Value inserted to the left\n";

                    break;

                }

                else if (n1->data < temp->data)

                {

                    temp = temp->left;

                }

                else if ((n1->data > temp->data) && (temp->right == NULL))

                {

                    temp->right = n1;

                    cout << "Value inserted the right\n";

                    break;

                }

                else if (n1->data > temp->data)

                {

                    temp = temp->right;

                }

            }

        }

    }

    nodes \*findNode(nodes \*node, int value)

    {

        if (node == NULL || node->data == value)

            return node;

        if (value < node->data)

            return findNode(node->left, value);

        else

            return findNode(node->right, value);

    }

};

int main()

{

    BST OBJ;

    OBJ.insert(5);

    OBJ.insert(3);

    OBJ.insert(7);

    OBJ.insert(2);

    OBJ.insert(4);

    OBJ.insert(6);

    OBJ.insert(8);

    int value;

    cout << "Enter the value to find in the BST: ";

    cin >> value;

    nodes \*foundNode = OBJ.findNode(OBJ.root, value);

    if (foundNode != NULL)

        cout << "Node with value " << value << " found in the BST." << endl;

    else

        cout << "Node with value " << value << " not found in the BST." << endl;

    return 0;

}

OUTPUT:  
