**TASK 1:**

#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

Node\* addNode(int key) {

Node\* ptr = new Node;

ptr->key = key;

ptr->L = ptr->R = NULL;

ptr->height = 1;

return ptr;

}

int BalFactor(Node\* T) {

if (T == NULL)

return 0;

return HEight(T->L) - HEight(T->R);

}

int HEight(Node\* T) {

if (T == NULL) {

return 0;

}

return T->height;

}

int getMax(int x, int y) {

if (x > y)

return x;

return y;

}

Node\* rotateRight(Node\* K2) {

Node\* K1 = K2->L;

Node\* temp = K1->R;

K1->R = K2;

K2->L = temp;

K2->height = getMax(HEight(K2->L), HEight(K2->R)) + 1;

K1->height = getMax(HEight(K1->L), HEight(K1->R)) + 1;

return K1;

}

Node\* rotateLeft(Node\* K1) {

Node\* K2 = K1->R;

Node\* temp = K2->L;

K2->L = K1;

K1->R = temp;

K1->height = getMax(HEight(K1->L), HEight(K1->R)) + 1;

K2->height = getMax(HEight(K2->L), HEight(K2->R)) + 1;

return K2;

}

public:

Tree() {

root = NULL;

};

Node\* getRoot() { return root; }

Node\* insertNode(Node\* T, int key) {

if (T == NULL) {

Node\* ptr = addNode(key);

return ptr;

}

if (key < T->key)

T->L = insertNode(T->L, key);

if (key > T->key)

T->R = insertNode(T->R, key);

T->height = 1 + getMax(HEight(T->L), HEight(T->R));

int factor = BalFactor(T);

if (factor > 1 && key < T->key)

{

cout << "Right Rotation takes place." << endl;

return rotateRight(T);

}

if (factor < -1 && key > T->key)

{

cout << "Left Rotation takes place." << endl;

return rotateLeft(T);

}

if (factor > 1 && key > T->key)

{

cout << "Left Right Rotation takes place." << endl;

T->L = rotateLeft(T->L);

return rotateRight(T);

}

if (factor < -1 && key < T->key)

{

cout << "Left Right Rotation takes place." << endl;

T->R = rotateRight(T->R);

return rotateLeft(T);

}

return T;

}

void inorderPrint(Node\* T) {

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for Printing values" << endl;

cout << "Press 3 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insertNode(T, num);

break;

case 2:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 3:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

}

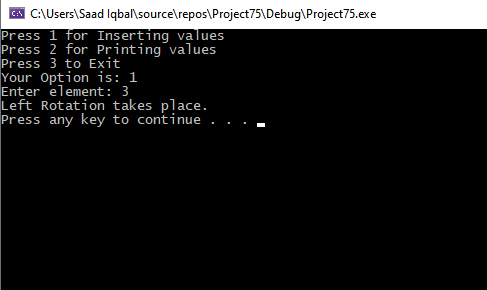
system("pause");

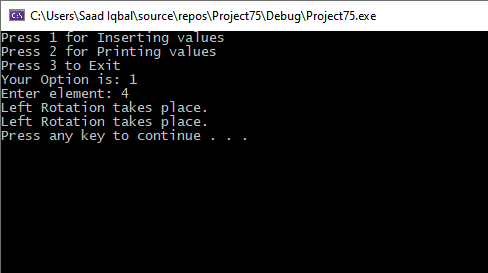
system("cls");

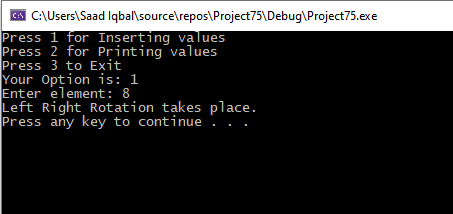
}

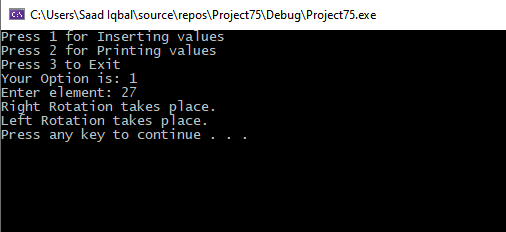
}

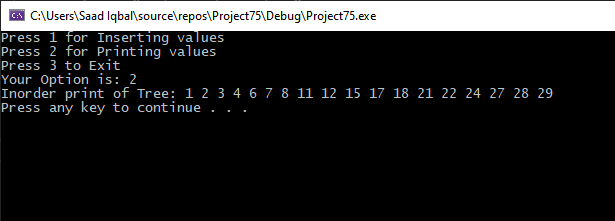
**OUTPUT:**











**TASK 2:**

#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

Node\* addNode(int key) {

Node\* ptr = new Node;

ptr->key = key;

ptr->L = ptr->R = NULL;

ptr->height = 1;

return ptr;

}

int BalFactor(Node\* T) {

if (T == NULL)

return 0;

return HEight(T->L) - HEight(T->R);

}

int HEight(Node\* T) {

if (T == NULL) {

return 0;

}

return T->height;

}

int getMax(int x, int y) {

if (x > y)

return x;

return y;

}

Node\* rotateRight(Node\* K2) {

Node\* K1 = K2->L;

Node\* temp = K1->R;

K1->R = K2;

K2->L = temp;

K2->height = getMax(HEight(K2->L), HEight(K2->R)) + 1;

K1->height = getMax(HEight(K1->L), HEight(K1->R)) + 1;

return K1;

}

Node\* rotateLeft(Node\* K1) {

Node\* K2 = K1->R;

Node\* temp = K2->L;

K2->L = K1;

K1->R = temp;

K1->height = getMax(HEight(K1->L), HEight(K1->R)) + 1;

K2->height = getMax(HEight(K2->L), HEight(K2->R)) + 1;

return K2;

}

public:

Tree() {

root = NULL;

};

Node\* getRoot() { return root; }

Node\* insertNode(Node\* T, int key) {

if (T == NULL) {

Node\* ptr = addNode(key);

return ptr;

}

if (key < T->key)

T->L = insertNode(T->L, key);

if (key > T->key)

T->R = insertNode(T->R, key);

T->height = 1 + getMax(HEight(T->L), HEight(T->R));

int factor = BalFactor(T);

if (factor > 1 && key < T->key)

{

cout << "Right Rotation takes place." << endl;

return rotateRight(T);

}

if (factor < -1 && key > T->key)

{

cout << "Left Rotation takes place." << endl;

return rotateLeft(T);

}

if (factor > 1 && key > T->key)

{

cout << "Left Right Rotation takes place." << endl;

T->L = rotateLeft(T->L);

return rotateRight(T);

}

if (factor < -1 && key < T->key)

{

cout << "Left Right Rotation takes place." << endl;

T->R = rotateRight(T->R);

return rotateLeft(T);

}

return T;

}

int getLeafCount(Node\* Node)

{

if (Node == NULL)

{

return 0;

}

else if (Node->L == NULL && Node->R == NULL)

{

return 1;

}

else

{

return (getLeafCount(Node->L) + getLeafCount(Node->R));

}

}

void inorderPrint(Node\* T)

{

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for Inorder display" << endl;

cout << "press 3 to print leaf counts " << endl;

cout << "Press 4 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insertNode(T, num);

break;

case 2:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 3:

cout << "Number of leaf node are: ";

cout<<tree.getLeafCount(T);

cout << endl;

break;

case 4:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

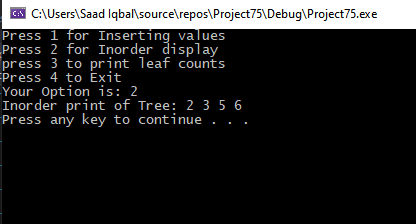
}

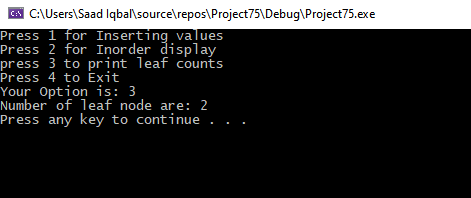
system("pause");

system("cls");

}

}

****

****

**TASK 3:**

#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

Node\* addNode(int key) {

Node\* ptr = new Node;

ptr->key = key;

ptr->L = ptr->R = NULL;

ptr->height = 1;

return ptr;

}

int BalFactor(Node\* T) {

if (T == NULL)

return 0;

return HEight(T->L) - HEight(T->R);

}

int HEight(Node\* T) {

if (T == NULL) {

return 0;

}

return T->height;

}

int getMax(int x, int y) {

if (x > y)

return x;

return y;

}

//Rotate right function for balancing

Node\* rotateRight(Node\* K2)

{

Node\* K1 = K2->L;

Node\* temp = K1->R;

K1->R = K2;

K2->L = temp;

K2->height = getMax(HEight(K2->L), HEight(K2->R)) + 1;

K1->height = getMax(HEight(K1->L), HEight(K1->R)) + 1;

return K1;

}

//Rotate Left function for balancing

Node\* rotateLeft(Node\* K1)

{

Node\* K2 = K1->R;

Node\* temp = K2->L;

K2->L = K1;

K1->R = temp;

K1->height = getMax(HEight(K1->L), HEight(K1->R)) + 1;

K2->height = getMax(HEight(K2->L), HEight(K2->R)) + 1;

return K2;

}

//Rotate left right function for balancing

Node\* left\_right\_rotation(Node\* node)

{

node->L = rotateLeft(node->L);

return rotateRight(node);

}

//Rotate right left function for balancing

Node\* right\_left\_rotation(Node\* node)

{

node->R = rotateRight(node->R);

return rotateLeft(node);

}

public:

Tree() {

root = NULL;

};

Node\* getRoot() { return root; }

Node\* insertNode(Node\* T, int key) {

if (T == NULL) {

Node\* ptr = addNode(key);

return ptr;

}

if (key < T->key)

T->L = insertNode(T->L, key);

if (key > T->key)

T->R = insertNode(T->R, key);

T->height = 1 + getMax(HEight(T->L), HEight(T->R));

int factor = BalFactor(T);

if (factor > 1 && key < T->key)

{

cout << "Right Rotation takes place." << endl;

return rotateRight(T);

}

if (factor < -1 && key > T->key)

{

cout << "Left Rotation takes place." << endl;

return rotateLeft(T);

}

if (factor > 1 && key > T->key)

{

cout << "Left Right Rotation takes place." << endl;

return left\_right\_rotation(T);

}

if (factor < -1 && key < T->key)

{

cout << "Left Right Rotation takes place." << endl;

return right\_left\_rotation(T);

}

return T;

}

//print display of tree

void display(Node\* r, int space)

{

if (r == NULL)

{

return;

}

display(r->R, space);

cout << endl;

cout << r->key << "\n";

display(r->L, space);

}

//print inorder of tree

void inorderPrint(Node\* T)

{

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

//print preorder of tree

void print\_preorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

cout << node->key << " ";

inorderPrint(node->L);

inorderPrint(node->R);

}

}

//print postorder of tree

void print\_postorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

inorderPrint(node->L);

inorderPrint(node->R);

cout << node->key << " ";

}

}

//height of tree

int height(Node\* t)

{

int h = 0;

if (t != NULL)

{

int l\_height = height(t->L);

int r\_height = height(t->R);

int max\_height = max(l\_height, r\_height);

h = max\_height + 1;

}

return h;

}

//difference of tree

int difference(Node\* t)

{

int l\_height = height(t->L);

int r\_height = height(t->R);

int b\_factor = l\_height - r\_height;

return b\_factor;

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for display" << endl;

cout << "Press 3 for Inorder display" << endl;

cout << "Press 4 for preorder display" << endl;

cout << "Press 5 for postorder display" << endl;

cout << "press 6 to print height of tree " << endl;

cout << "press 7 to print difference of tree " << endl;

cout << "Press 8 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insertNode(T, num);

break;

case 2:

cout << "Display of Tree: ";

tree.display(T,5);

cout << endl;

break;

case 3:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 4:

cout << "Preorder print of Tree: ";

tree.print\_preorder(T);

cout << endl;

break;

case 5:

cout << "postorder print of Tree: ";

tree.print\_postorder(T);

cout << endl;

break;

case 6:

cout << "Height of tree is : ";

cout<<tree.height(T);

cout << endl;

break;

case 7:

cout << "Difference of tree is : ";

cout << tree.difference(T);

cout << endl;

break;

case 8:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

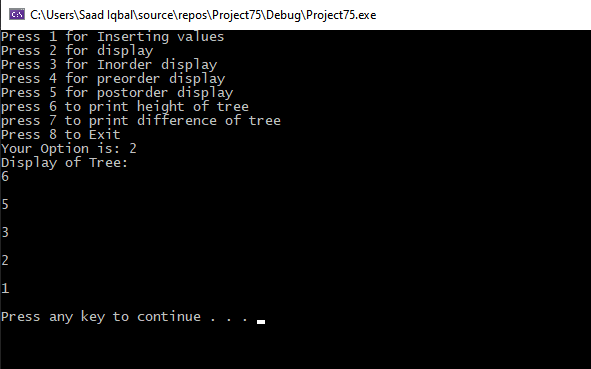
}

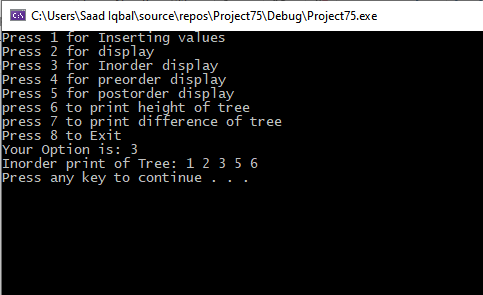
system("pause");

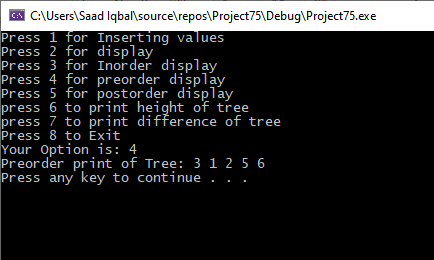
system("cls");

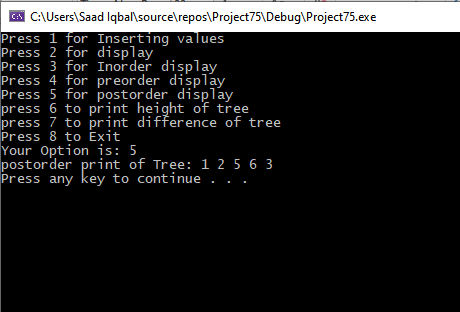
}

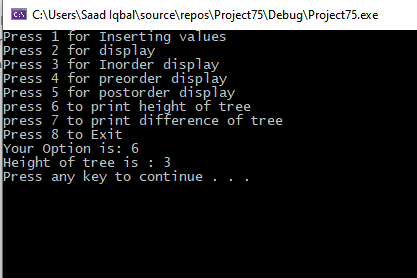
}

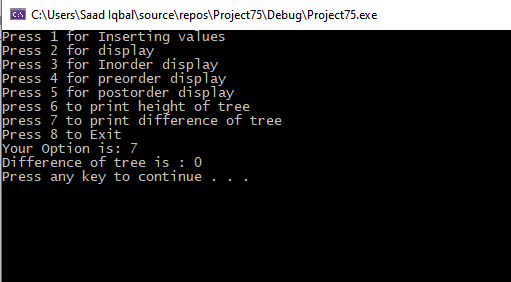
****

****

****

****

****

****