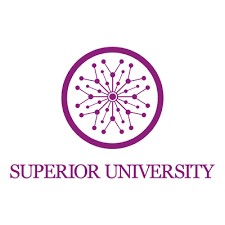
**TASK NO 12**



**M ZUHAIB ANWAR**

**SU92-BSSEM-S24-005**

**SE-3A**

**DSA LAB**

//BST TREE

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

class BST

{

private:

Node\* root;

Node\* insertHelper(Node\* node, int val)

{

if (node == nullptr) {

return new Node(val);

}

if (val < node->data) {

node->left = insertHelper(node->left, val);

} else if (val > node->data) {

node->right = insertHelper(node->right, val);

}

return node;

}

void inorderHelper(Node\* node)

{

if (node == nullptr) return;

inorderHelper(node->left);

cout << node->data << " ";

inorderHelper(node->right);

}

void preorderHelper(Node\* node)

{

if (node == nullptr) return;

cout << node->data << " ";

preorderHelper(node->left);

preorderHelper(node->right);

}

void postorderHelper(Node\* node)

{

if (node == nullptr) return;

postorderHelper(node->left);

postorderHelper(node->right);

cout << node->data << " ";

}

public:

BST() : root(nullptr) {}

void insert(int val)

{

root = insertHelper(root, val);

cout << "Inserted " << val << " into BST" << endl;

}

void inorder()

{

cout << "Inorder Traversal: ";

inorderHelper(root);

cout << endl;

}

void preorder()

{

cout << "Preorder Traversal: ";

preorderHelper(root);

cout << endl;

}

void postorder()

{

cout << "Postorder Traversal: ";

postorderHelper(root);

cout << endl;

}

};

int main() {

BST bst;

bst.insert(50);

bst.insert(30);

bst.insert(20);

bst.insert(40);

bst.insert(70);

bst.insert(60);

bst.insert(80);

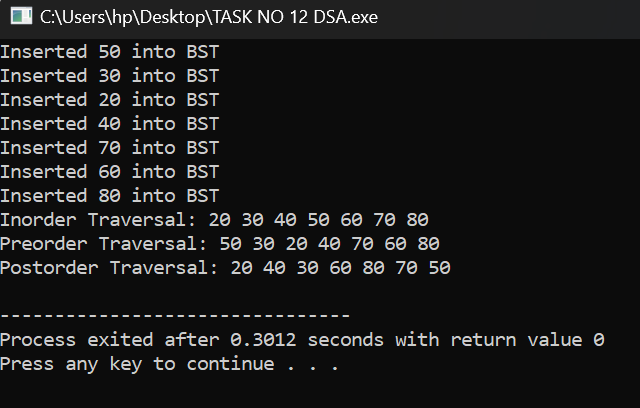
bst.inorder();

bst.preorder();

bst.postorder();

return 0;

}



//AVL TREE

#include <iostream>

#include <algorithm>

using namespace std;

class AVLNode

{

public:

int data;

AVLNode\* left;

AVLNode\* right;

int height;

AVLNode(int val) : data(val), left(nullptr), right(nullptr), height(1) {}

};

class AVLTree

{

private:

AVLNode\* root;

int height(AVLNode\* node)

{

return node ? node->height : 0;

}

int balanceFactor(AVLNode\* node)

{

return node ? height(node->left) - height(node->right) : 0;

}

void updateHeight(AVLNode\* node)

{

node->height = 1 + max(height(node->left), height(node->right));

}

AVLNode\* rightRotate(AVLNode\* y)

{

AVLNode\* x = y->left;

AVLNode\* T2 = x->right;

x->right = y;

y->left = T2;

updateHeight(y);

updateHeight(x);

return x;

}

AVLNode\* leftRotate(AVLNode\* x)

{

AVLNode\* y = x->right;

AVLNode\* T2 = y->left;

y->left = x;

x->right = T2;

updateHeight(x);

updateHeight(y);

return y;

}

AVLNode\* balance(AVLNode\* node)

{

if (!node) return nullptr;

updateHeight(node);

int bf = balanceFactor(node);

if (bf > 1 && balanceFactor(node->left) >= 0)

{

return rightRotate(node);

}

if (bf < -1 && balanceFactor(node->right) <= 0)

{

return leftRotate(node);

}

if (bf > 1 && balanceFactor(node->left) < 0)

{

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (bf < -1 && balanceFactor(node->right) > 0)

{

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

AVLNode\* insertHelper(AVLNode\* node, int val)

{

if (!node) return new AVLNode(val);

if (val < node->data) {

node->left = insertHelper(node->left, val);

} else if (val > node->data)

{

node->right = insertHelper(node->right, val);

} else

{

return node;

}

return balance(node);

}

void inorderHelper(AVLNode\* node)

{

if (!node) return;

inorderHelper(node->left);

cout << node->data << " ";

inorderHelper(node->right);

}

void preorderHelper(AVLNode\* node)

{

if (!node) return;

cout << node->data << " ";

preorderHelper(node->left);

preorderHelper(node->right);

}

void postorderHelper(AVLNode\* node)

{

if (!node) return;

postorderHelper(node->left);

postorderHelper(node->right);

cout << node->data << " ";

}

public:

AVLTree() : root(nullptr) {}

void insert(int val)

{

root = insertHelper(root, val);

cout << "Inserted " << val << " into AVL tree" << endl;

}

void inorder()

{

cout << "Inorder Traversal: ";

inorderHelper(root);

cout << endl;

}

void preorder()

{

cout << "Preorder Traversal: ";

preorderHelper(root);

cout << endl;

}

void postorder()

{

cout << "Postorder Traversal: ";

postorderHelper(root);

cout << endl;

}

};

int main() {

AVLTree avl;

avl.insert(10);

avl.insert(20);

avl.insert(30);

avl.insert(40);

avl.insert(50);

avl.insert(25);

avl.inorder();

avl.preorder();

avl.postorder();

return 0;

}

