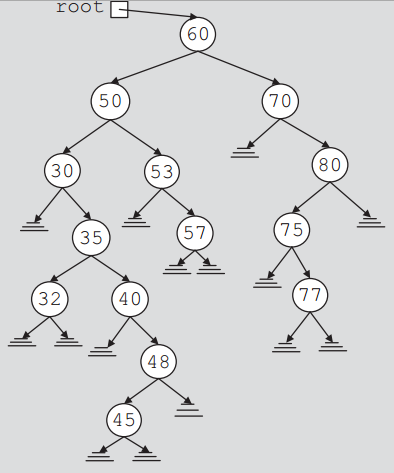
# Lab Task 1



Write the *recursive* code of **searching** in a BST. Using your recursive insertion code, create the binary tree (given on the right) and search 75, 40 and 1001 in it print the results.

**CODE:**

#include<iostream>

using namespace std;

class Node {

public:

Node\* left, \* right;

int data;

Node(int d=0):data(d),left(nullptr),right(nullptr){}

};

Node\* insert(Node\* root, int d) {

if (root == nullptr) {

return new Node(d);

}

if (root->data == d) {

return root;

}

if (d < root->data) {

root->left = insert(root->left, d);

}

else {

root->right = insert(root->right, d);

}

return root;

}

bool searchBst(Node\* root, int d) {

if (root == nullptr) {

return false;

}

if (root->data == d) {

return true;

}

if (d < root->data) {

return searchBst(root->left, d);

}

else {

return searchBst(root->right, d);

}

}

void preOrder(Node\* root) {

if (root == nullptr) {

return;

}

cout << root->data << " ";

preOrder(root->left);

preOrder(root->right);

}

int main() {

Node\* root = nullptr;

root = insert(root, 60);

root = insert(root, 50);

root = insert(root, 70);

root = insert(root, 30);

root = insert(root, 53);

root = insert(root, 80);

root = insert(root, 35);

root = insert(root, 57);

root = insert(root, 75);

root = insert(root, 32);

root = insert(root, 40);

root = insert(root, 77);

root = insert(root, 48);

root = insert(root, 45);

cout << "Elements in BST(PreOrder): ";

preOrder(root);

cout<< endl << endl;

cout << "Result of Searching 75: " << searchBst(root, 75) << endl;

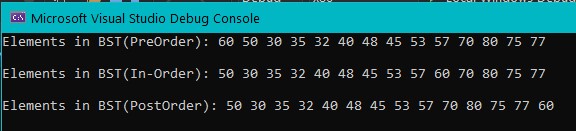
cout << "Result of Searching 40: " << searchBst(root, 40) << endl;

cout << "Result of Searching 1001: " << searchBst(root, 1001) << endl;

return 0;

}

**SCREENSHOT:**



# Lab Task 2

Now write three print functions through recursion which prints the tree in task 1 in following orders:

1. Pre-Order
2. In-Order
3. Post-Order

**CODE:**

#include<iostream>

using namespace std;

class Node {

public:

Node\* left, \* right;

int data;

Node(int d = 0) : data(d), left(nullptr), right(nullptr) {}

};

class BST {

private:

Node\* root;

Node\* insert(Node\* root, int d) {

if (root == nullptr) {

return new Node(d);

}

if (root->data == d) {

return root;

}

if (d < root->data) {

root->left = insert(root->left, d);

}

else {

root->right = insert(root->right, d);

}

return root;

}

bool search(Node\* root, int d) {

if (root == nullptr) {

return false;

}

if (root->data == d) {

return true;

}

if (d < root->data) {

return search(root->left, d);

}

else {

return search(root->right, d);

}

}

void preOrder(Node\* root) {

if (root == nullptr) {

return;

}

cout << root->data << " ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root) {

if (root == nullptr) {

return;

}

preOrder(root->left);

cout << root->data << " ";

preOrder(root->right);

}

void postOrder(Node\* root) {

if (root == nullptr) {

return;

}

preOrder(root->left);

preOrder(root->right);

cout << root->data << " ";

}

public:

BST() : root(nullptr) {}

void insert(int d) {

root = insert(root, d);

}

bool search(int d) {

return search(root, d);

}

void preOrder() {

preOrder(root);

cout << endl;

}

void inOrder() {

inOrder(root);

cout << endl;

}

void postOrder() {

postOrder(root);

cout << endl;

}

};

int main() {

BST tree;

// Insert values into BST

tree.insert(60);

tree.insert(50);

tree.insert(70);

tree.insert(30);

tree.insert(53);

tree.insert(80);

tree.insert(35);

tree.insert(57);

tree.insert(75);

tree.insert(32);

tree.insert(40);

tree.insert(77);

tree.insert(48);

tree.insert(45);

cout << "Elements in BST(PreOrder): ";

tree.preOrder();

cout << "\nElements in BST(In-Order): ";

tree.inOrder();

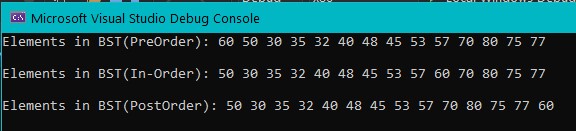
cout << "\nElements in BST(PostOrder): ";

tree.postOrder();

return 0;

}

**SCREENSHOT:**



# Lab Task 3

Now write a recursive function which prints the height of the BST and test it on the tree made in task1.

**CODE:**

#include<iostream>

using namespace std;

class Node {

public:

Node\* left, \* right;

int data;

Node(int d = 0) : data(d), left(nullptr), right(nullptr) {}

};

class BST {

private:

Node\* root;

Node\* insert(Node\* root, int d) {

if (root == nullptr) {

return new Node(d);

}

if (root->data == d) {

return root;

}

if (d < root->data) {

root->left = insert(root->left, d);

}

else {

root->right = insert(root->right, d);

}

return root;

}

bool search(Node\* root, int d) {

if (root == nullptr) {

return false;

}

if (root->data == d) {

return true;

}

if (d < root->data) {

return search(root->left, d);

}

else {

return search(root->right, d);

}

}

void preOrder(Node\* root) {

if (root == nullptr) {

return;

}

cout << root->data << " ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root) {

if (root == nullptr) {

return;

}

preOrder(root->left);

cout << root->data << " ";

preOrder(root->right);

}

void postOrder(Node\* root) {

if (root == nullptr) {

return;

}

preOrder(root->left);

preOrder(root->right);

cout << root->data << " ";

}

int height(Node\* root) {

if (root == nullptr) {

return 0;

}

int lh = height(root->left);

int rh = height(root->right);

return lh > rh ? lh+1 : rh+1;

}

public:

BST() : root(nullptr) {}

void insert(int d) {

root = insert(root, d);

}

bool search(int d) {

return search(root, d);

}

void preOrder() {

preOrder(root);

cout << endl;

}

void inOrder() {

inOrder(root);

cout << endl;

}

void postOrder() {

postOrder(root);

cout << endl;

}

int height() {

return height(root);

}

};

int main() {

BST tree;

// Insert values into BST

tree.insert(60);

tree.insert(50);

tree.insert(70);

tree.insert(30);

tree.insert(53);

tree.insert(80);

tree.insert(35);

tree.insert(57);

tree.insert(75);

tree.insert(32);

tree.insert(40);

tree.insert(77);

tree.insert(48);

tree.insert(45);

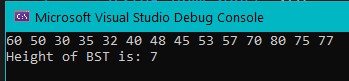
tree.preOrder();

cout << "Height of BST is: " << tree.height() << endl;

return 0;

}

**SCREENSHOT:**



# Lab Task 4

Now write a recursive function which counts the leaf nodes of the BST and test it on the tree made in task1.

**CODE:**

#include<iostream>

using namespace std;

class Node {

public:

Node\* left, \* right;

int data;

Node(int d = 0) : data(d), left(nullptr), right(nullptr) {}

};

class BST {

private:

Node\* root;

Node\* insert(Node\* root, int d) {

if (root == nullptr) {

return new Node(d);

}

if (root->data == d) {

return root;

}

if (d < root->data) {

root->left = insert(root->left, d);

}

else {

root->right = insert(root->right, d);

}

return root;

}

bool search(Node\* root, int d) {

if (root == nullptr) {

return false;

}

if (root->data == d) {

return true;

}

if (d < root->data) {

return search(root->left, d);

}

else {

return search(root->right, d);

}

}

void preOrder(Node\* root) {

if (root == nullptr) {

return;

}

cout << root->data << " ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root) {

if (root == nullptr) {

return;

}

preOrder(root->left);

cout << root->data << " ";

preOrder(root->right);

}

void postOrder(Node\* root) {

if (root == nullptr) {

return;

}

preOrder(root->left);

preOrder(root->right);

cout << root->data << " ";

}

int height(Node\* root) {

if (root == nullptr) {

return 0;

}

int lh = height(root->left);

int rh = height(root->right);

return lh > rh ? lh+1 : rh+1;

}

int leafNo(Node\* root) {

if (root == nullptr) {

return 0;

}

if (root->left == nullptr && root->right == nullptr) {

return 1;

}

return leafNo(root->left) + leafNo(root->right);

}

public:

BST() : root(nullptr) {}

void insert(int d) {

root = insert(root, d);

}

bool search(int d) {

return search(root, d);

}

void preOrder() {

preOrder(root);

cout << endl;

}

void inOrder() {

inOrder(root);

cout << endl;

}

void postOrder() {

postOrder(root);

cout << endl;

}

int height() {

return height(root);

}

int leafNo() {

return leafNo(root);

}

};

int main() {

BST tree;

// Insert values into BST

tree.insert(60);

tree.insert(50);

tree.insert(70);

tree.insert(30);

tree.insert(53);

tree.insert(80);

tree.insert(35);

tree.insert(57);

tree.insert(75);

tree.insert(32);

tree.insert(40);

tree.insert(77);

tree.insert(48);

tree.insert(45);

tree.preOrder();

cout << "Height of BST is: " << tree.height() << endl;

cout << "Number of Leaves in BST tree are: " << tree.leafNo();

return 0;

}

**SCREENSHOT:**

