**Group A: Assignment No:04**

**Problem Statement:**

**Beginning with an empty binary search tree, construct binary search tree by inserting the values in the order given. After constructing a binary tree - i. Insert new node ii. Find number of nodes in longest path from root iii. Minimum data value found in the tree iv. Change a tree so that the roles of the left and right pointers are swapped at every node v. Search a value**

**Program:**

**Problem Statement:**

**Beginning with an empty binary search tree, construct binary search tree by inserting the values in the order given. After constructing a binary tree - i. Insert new node ii. Find number of nodes in longest path from root iii. Minimum data value found in the tree iv. Change a tree so that the roles of the left and right pointers are swapped at every node v. Search a value**

**Program:**

**#include<iostream>**

**using namespace std;**

**class node {**

**public:**

**int data;**

**node \*left;**

**node \*right;**

**};**

**class Bst {**

**public:**

**node \*root;**

**Bst () {**

**root = NULL;**

**}**

**void create ();**

**void insert ();**

**void postorder (node\*);**

**void inorder (node\*);**

**void preorder (node\*);**

**void search (int key);**

**void minimum ();**

**int height (node\*);**

**void mirror (node\*);**

**};**

**void Bst::create () {**

**int ans;**

**cout << "\nEnter number of keys to insert: ";**

**cin >> ans;**

**cout << '\n';**

**while (ans--)**

**insert();**

**}**

**void Bst::inorder (node \*root) {**

**if (root != NULL) {**

**inorder (root -> left);**

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

**inorder (root -> right);**

**}**

**}**

**void Bst::preorder (node \*root) {**

**if (root != NULL) {**

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

**preorder (root -> left);**

**preorder (root -> right);**

**}**

**}**

**void Bst::postorder (node \*root) {**

**if (root != NULL) {**

**postorder (root -> left);**

**postorder (root -> right);**

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

**}**

**}**

**void Bst::insert () {**

**node \*curr,\*temp;**

**cout << "Enter data: ";**

**curr = new node;**

**cin >> curr -> data;**

**curr -> left = curr -> right = NULL;**

**if (root == NULL)**

**root = curr;**

**else {**

**temp = root;**

**while (1) {**

**if (curr -> data <= temp -> data) {**

**if (temp -> left == NULL) {**

**temp -> left = curr;**

**break;**

**}**

**else**

**temp = temp -> left;**

**}**

**else {**

**if (temp -> right == NULL) {**

**temp -> right = curr;**

**break;**

**}**

**else**

**temp = temp -> right;**

**}**

**}**

**}**

**}**

**void Bst::search (int key) {**

**node \*curr;**

**curr = root;**

**while (curr != NULL) {**

**if (curr -> data == key) {**

**cout << key << " found";**

**break;**

**}**

**else {**

**if (key<curr -> data)**

**curr = curr -> left;**

**else**

**curr = curr -> right;**

**}**

**}**

**if (curr == NULL)**

**cout << key << " not found";**

**}**

**void Bst::minimum () {**

**node \*temp = root;**

**int min;**

**while (temp -> left != NULL) {**

**min = temp -> data;**

**temp = temp -> left;**

**if (temp -> data<min)**

**min = temp -> data;**

**else**

**temp = temp -> left;**

**}**

**cout << "\nMinimum number is: " << min;**

**}**

**int Bst::height (node \*root) {**

**if (root == NULL)**

**return 0;**

**else {**

**if (height (root -> right) > height (root -> left))**

**return (1 + height (root -> right));**

**else**

**return (1 + height (root -> left));**

**}**

**}**

**void Bst::mirror (node \*root) {**

**if (root == NULL)**

**return;**

**else {**

**mirror(root -> left);**

**mirror(root -> right);**

**swap(root -> left, root -> right);**

**}**

**}**

**int main () {**

**Bst b;**

**int key,ch;**

**do {**

**cout << "\n\n1.Create 2.Insert 3.Inorder 4.Preorder 5.Postorder 6.Search 7.Minimum 8.Height 9.Mirror\n";**

**cout << "Your choice [1/2/3/4/5/6/7/8/9] ";**

**cin >> ch;**

**switch (ch) {**

**case 1:**

**b.create ();**

**break;**

**case 2:**

**cout << '\n';**

**b.insert ();**

**break;**

**case 3:**

**cout << "\nInorder traversal is:";**

**b.inorder (b.root);**

**break;**

**case 4:**

**cout << "\nPreorder traversal is:";**

**b.preorder (b.root);**

**break;**

**case 5:**

**cout << "\nPostorder traversal is:";**

**b.postorder (b.root);**

**break;**

**case 6:**

**cout << "\nEnter search key: ";**

**cin >> key;**

**b.search (key);**

**break;**

**case 7:**

**b.minimum ();**

**break;**

**case 8:**

**cout << "\nHeight of tree: " << b.height (b.root);**

**break;**

**case 9:**

**b.mirror (b.root);**

**cout << "\nTree is now mirrored!!!"**

**<< "\nInorder traversal is:";**

**b.inorder (b.root);**

**cout << "\nPreorder traversal is:";**

**b.preorder (b.root);**

**cout << "\nPostorder traversal is:";**

**b.postorder (b.root);**

**break;**

**}**

**}while (ch < 10);**

**return 0;**

**}**



