***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

***Data Structure Lab – II [15B17CI271]***

***Lab A Week 10***

***Q1.***

*#include<iostream>*

*#include<stack>*

*#include<queue> using*

*namespace std;*

*class Node*

*{*

*public: int data;*

*Node\* left;*

*Node\* right;*

*};*

*Node\* createnode(int data)*

*{*

*Node\* root=new*

*Node(); root-*

*>data=data;*

*root>left=NULL;*

*root>right=NULL; return*

*root;*

*}*

*int getheight(Node\* n)*

*{*

*if(n==NULL)*

*{*

*return 0;*

*}*

*else*

*{*

*int height=max(getheight(n->left),getheight(n->right));*

*return height+1;*

*}*

*}*

*void Insert(Node\* root,int key)*

*{*

*Node\* prev=NULL;*

*while(root!=NULL) {*

*prev=root;*

*if(key==root->data)*

*{*

*cout<<"CANNOT INSERT "<<key<<endl;*

*return;*

*}*

*else if(key<root->data)*

*{*

*root= root->left;*

*}*

*else*

*{*

*root = root->right;*

*}*

*}*

*Node\* n = createnode(key);*

*if(key<prev->data)*

*{*

*prev->left=n;*

*}*

*else*

*{*

*prev->right=n;*

*}*

*}*

*void inorder(Node\* root)*

*{*

*stack<Node\*>s; Node\* curr=root;*

*while(curr!=NULL || s.empty()==false)*

*{*

*while(curr!=NULL)*

*{*

*s.push(curr);*

*curr=curr->left;*

*}*

*curr=s.top();*

*s.pop();*

*cout<<curr>data<<" ";*

*curr=curr->right;*

*}*

*}*

*void preorder(Node\* root)*

*{*

*if(root==NULL)*

*return; stack<Node\*>s;*

*s.push(root);*

*while(s.empty()==false)*

*{*

*Node\* n=s.top();*

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

*s.pop(); if(n->right)*

*{*

*s.push(n->right);*

*} if(n-*

*>left)*

*{*

*s.push(n->left);*

*}*

*}*

*}*

*void postorder(Node\* root)*

*{*

*if(root==NULL)*

*return;*

*stack<Node\*>s1,s2;*

*s1.push(root); Node\**

*node;*

*while(s1.empty()==false)*

*{*

*node=s1.top();*

*s1.pop(); s2.push(node);*

*if(node-*

*>left)*

*{*

*s1.push(node->left);*

*}*

*if(node->right)*

*{*

*s1.push(node->right);*

*}*

*}*

*while(s2.empty()==false)*

*{*

*node=s2.top();*

*s2.pop(); cout<<node-*

*>data<<" ";*

*}*

*}*

*void levelorder(Node\* root)*

*{*

*if(root==NULL)*

*return;*

*queue<Node\*>q;*

*q.push(root);*

*while(q.empty()==false)*

*{*

*Node\* node=q.front();*

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

*q.pop(); if(node->left)*

*q.push(node->left);*

*if(node->right)*

*q.push(node->right);*

*}*

*}*

*int main()*

*{*

*Node\* root=NULL;*

*root=createnode(10);*

*Insert(root,20);*

*Insert(root,30);*

*Insert(root,40);*

*Insert(root,50);*

*Insert(root,60);*

*Insert(root,70);*

*Insert(root,75); Insert(root,80); int*

*height=getheight(root); cout<<"Height of*

*binary tree: "<<height<<endl;*

*cout<<endl; cout<<"Inorder: ";*

*inorder(root); cout<<endl;*

*cout<<"Preorder: "; preorder(root);*

*cout<<endl; cout<<"Postorder: ";*

*postorder(root); cout<<endl;*

*cout<<"Levelorder: ";*

*levelorder(root); return 0;*

*}*

***OUTPUT:***

**

***Q2.***

*#include <iostream>*

*#include <stack>*

*using namespace std;*

*class Node*

*{*

*public:*

*int key; Node\**

*left,\*right; int*

*height; Node(int n)*

*{*

*key=n;*

*left=right=NULL;*

*}*

*};*

*int height(Node \*root)*

*{*

*if(root==NULL)*

*return 0;*

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

*}*

*Node \*rightRotate(Node \*y)*

*{*

*Node \*x = y->left;*

*Node \*T2 = x->right;*

*x->right = y; y->left*

*= T2; y->height = max(height(y->left),height(y->right)) + 1;*

*x>height = max(height(x->left),height(x->right)) + 1; return*

*x;*

*}*

*Node \*leftRotate(Node \*x)*

*{*

*Node \*y = x->right;*

*Node \*T2 = y->left;*

*y->left = x; x->right*

*= T2;*

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

*y>height = max(height(y->left),height(y->right)) + 1; return*

*y;*

*}*

*int getBalance(Node \*N)*

*{*

*if (N == NULL) return 0; return*

*height(N->left) - height(N->right);*

*}*

*Node\* insertNode(Node\* node, int key)*

*{*

*if (node == NULL)*

*{*

*Node \* n=new Node(key);*

*return n;*

*}*

*if (key < node->key)*

*node->left = insertNode(node->left, key); else if*

*(key > node->key) node->right = insertNode(node-*

*>right, key); else return*

*node;*

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

*height(node->right)); int*

*balance = getBalance(node); if (balance > 1*

*&& key < node->left->key) return*

*rightRotate(node); if (balance < -1 && key*

*> node->right->key) return*

*leftRotate(node); if (balance > 1 && key*

*> node->left->key)*

*{*

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

*return rightRotate(node);*

*}*

*if (balance < -1 && key < node->right->key)*

*{*

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

*return leftRotate(node);*

*}*

*return node;*

*}*

*int main()*

*{*

*Node \*root=NULL; root=insertNode(root,10);*

*root=insertNode(root,20);*

*root=insertNode(root,30);*

*root=insertNode(root,40);*

*root=insertNode(root,50);*

*root=insertNode(root,60);*

*root=insertNode(root,70);*

*root=insertNode(root,75);*

*root=insertNode(root,80); cout<<"Height*

*of tree: "<<height(root);*

*return 0;*

*}*

***OUTPUT:***

******

***Q3.***

*#include<iostream>*

*using namespace std;*

*class Node*

*{*

*public: int*

*key; Node\* left;*

*Node\* right;*

*int height;*

*};*

*int getheight(Node\* root)*

*{*

*if(root==NULL)*

*{*

*return 0;*

*}*

*else*

*{*

*return root->height;*

*}*

*}*

*int max(int a,int b)*

*{*

*return (a>b)?a:b;*

*}*

*Node\* createnode(int val)*

*{*

*Node\* root=new Node();*

*root>key=val; root->left=NULL;*

*root->right=NULL; root>height=1;*

*return root;*

*}*

*int getbf(Node\* root)*

*{*

*if(root==NULL)*

*{*

*return 0;*

*}*

*return getheight(root->left) - getheight(root->right);*

*}*

*Node\* rightrotate(Node\* y)*

*{*

*Node\* x = y->left;*

*Node\* T2= x->right; x>right=y;*

*y->left=T2;*

*y->height=max(getheight(y->right),getheight(y->left))+1;*

*x->height=max(getheight(x->right),getheight(x->left))+1;*

*return x;*

*}*

*Node\* leftrotate(Node\* x)*

*{*

*Node\* y = x->right;*

*Node\* T2= y->left; y>left=x; x->right=T2; y-*

*>height=max(getheight(y->right),getheight(y->left))+1; x-*

*>height=max(getheight(x->right),getheight(x->left))+1;*

*return y;*

*}*

*Node\* insert(Node\* node, int key)*

*{*

*if (node == NULL)*

*return(createnode(key));*

*if (key < node->key) insert(node->left,*

*key);*

*node->left =*

*else if (key > node->key)*

*>right = insert(node->right, key);*

*else*

*return node;*

*nodenode->*

*height= 1 + max(getheight(node->left),*

*getheight(node->right));*

*int balance = getbf(node); if (balance*

*> 1 && key < node->left->key) return*

*rightrotate(node);*

*if (balance < -1 && key > node->right->key)*

*return leftrotate(node);*

*if (balance > 1 && key > node->left->key)*

*{*

*node->left = leftrotate(node->left); return*

*rightrotate(node);*

*}*

*if (balance < -1 && key < node->right->key)*

*{*

*node->right = rightrotate(node->right);*

*return leftrotate(node);*

*}*

*return node;*

*}*

*void inorder(Node\* root)*

*{*

*if (root!=NULL)*

*{*

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

*>key<<" "; inorder(root-*

*>right);*

*}*

*}*

*Node \* minValueNode(Node\* node)*

*{*

*Node\* current = node;*

*while (current->left != NULL)*

*current = current->left;*

*return current;*

*}*

*Node\* deleteNode(Node\* root, int key)*

*{*

*if (root == NULL)*

*return root;*

*if ( key < root->key ) root->left =*

*deleteNode(root->left, key); else if( key > root->key )*

*root->right = deleteNode(root-*

*>right, key);*

*else*

*{*

*if( (root->left == NULL) ||*

*(root->right == NULL) )*

*{*

*Node \*temp = root->left ?*

*root->left : root->right;*

*if (temp == NULL)*

*{*

*temp = root;*

*root = NULL;*

*}*

*else*

*\*root = \*temp; delete(temp);*

*}*

*else*

*{*

*Node\* temp = minValueNode(root->right);*

*root->key = temp->key; root>right =*

*deleteNode(root->right,*

*temp->key);*

*}*

*}*

*if (root == NULL) return root; root-*

*>height = 1 + max(getheight(root->left),*

*getheight(root->right));*

*int balance = getbf(root);*

*if (balance > 1 && getbf(root-*

*>left) >= 0)*

*return rightrotate(root);*

*if (balance > 1 && getbf(root-*

*>left) < 0)*

*{*

*root->left = leftrotate(root->left);*

*return rightrotate(root);*

*}*

*if (balance < -1 && getbf(root-*

*>right) <= 0) return*

*leftrotate(root);*

*if (balance < -1 && getbf(root-*

*>right) > 0)*

*{*

*root->right = rightrotate(root->right);*

*return leftrotate(root);*

*}*

*return root;*

*}*

*int main()*

*{*

*Node\* head=NULL; head=insert(head,10);*

*head=insert(head,20); head=insert(head,30);*

*head=insert(head,40); head=insert(head,50);*

*head=insert(head,45); head=insert(head,35);*

*head=insert(head,25); head=insert(head,15);*

*head=insert(head,5); head=insert(head,8);*

*head=insert(head,18); head=insert(head,28);*

*head=insert(head,38); head=insert(head,48);*

*cout<<endl;*

*cout<<"TREE BEFORE DELETION:";*

*inorder(head); deleteNode(head,38);*

*deleteNode(head,50);*

*deleteNode(head,10); cout<<endl;*

*cout<<"TREE AFTER DELETION:"; inorder(head);*

*return 0;*

*}*

***OUTPUT:***

******

***Q4.***

*#include<iostream>*

*using namespace std;*

*class Node*

*{*

*public: int data;*

*Node\* left;*

*Node\* right;*

*};*

*Node\* createnode(int data)*

*{*

*Node\* root=new Node(); root>data=data;*

*root->left=NULL;*

*root->right=NULL;*

*return root;*

*}*

*int getheight(Node\* n)*

*{*

*if(n==NULL)*

*{*

*return 0;*

*}*

*else*

*{*

*int height=max(getheight(n->left),getheight(n->right));*

*return height+1;*

*}*

*}*

*int max(int a,int b)*

*{*

*return (a>b)?a:b;*

*}*

*bool AVL(Node \*root) {*

*int lh; int rh;*

*if(root == NULL)*

*return 1;*

*lh = getheight(root->left);*

*rh = getheight(root->right);*

*if(abs(lh-rh) <= 1 && AVL(root->left) && AVL(root->right)) return 1;*

*return 0;*

*}*

*int main()*

*{*

*Node \*root = createnode(7); root>left =*

*createnode(6); root->right =*

*createnode(12); root->left->left =*

*createnode(4); root->left->right =*

*createnode(5); root->right->right =*

*createnode(13);*

*if(AVL(root))*

*cout << "The Tree is AVL Tree"<<endl;*

*else*

*cout << "The Tree is not AVL Tree "<<endl;*

*return 0;*

*}*

***OUTPUT:***

******

***Q5.***

*#include <iostream>*

*#include <vector>*

*#include<algorithm>*

*using namespace std;*

*class Node*

*{*

*public:*

*int val;*

*vector<Node \*> child;*

*Node(int data)*

*{*

*val=data;*

*}*

*};*

*void insert(Node \*root, int parent, Node \*node)*

*{ if*

*(!root)*

*{*

*root = node;*

*}*

*else*

*{*

*if (root->val == parent)*

*{*

*root->child.push\_back(node);*

*}*

*else*

*{*

*int l = root->child.size();*

*for(int i = 0; i < l; i++)*

*{*

*if (root->child[i]->val == parent)*

*insert(root->child[i], parent, node);*

*else*

*insert(root->child[i], parent, node);*

*}*

*}*

*}*

*}*

*void levelorder(vector<Node \*> &prev\_level)*

*{*

*vector<Node \*> cur\_level;*

*vector<int> print\_data; int*

*l = prev\_level.size();*

*if (l == 0)*

*{*

*exit(0);*

*}*

*for(int i = 0; i < l; i++)*

*{*

*int prev\_level\_len = prev\_level[i]->child.size();*

*for(int j = 0; j < prev\_level\_len; j++)*

*{*

*cur\_level.push\_back(prev\_level[i]->child[j]);*

*print\_data.push\_back(prev\_level[i]->child[j]->val);*

*}*

*}*

*prev\_level = cur\_level;*

*for(auto i : print\_data)*

*{*

*cout << i << " ";*

*}*

*levelorder(prev\_level);*

*}*

*void levelorder\_root(Node \*root)*

*{ if*

*(root)*

*{*

*vector<Node \*> level; level.push\_back(root);*

*cout<<root-*

*>val<<endl;*

*levelorder(level);*

*}*

*}*

*int main()*

*{*

*int arr[] = {10,20,30,40,50,45,35,25,15,5,8,18,28,38}; int*

*arr2[]={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1}; Node*

*\*root = new Node(-1);*

*int l = sizeof(arr) / sizeof(int); vector<int>*

*que; que.push\_back(-1); while (true)*

*{*

*vector<int> temp;*

*for(int i = 0; i < l; i++)*

*{*

*if (find(que.begin(),que.end(), arr2[i]) != que.end())*

*{*

*insert(root, arr2[i], new Node(arr[i]));*

*temp.push\_back(i);*

*}*

*}*

*que = temp;*

*if (que.size() == 0)*

*{*

*break;*

*}*

*}*

*levelorder\_root(root);*

*}*

***OUTPUT:***

******