***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

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

***Lab A Week 9***

***Q1)***

*#include <iostream>*

*using namespace std;*

*#define SIZE(arr) (sizeof(arr)/sizeof(arr[0]))*

*class Node*

*{*

*public:*

*int data;*

*Node \*next;*

*Node \*child;*

*};*

*Node \*createList(int \*arr, int n)*

*{*

*Node \*head = NULL;*

*Node \*p;*

*int i;*

*for (i = 0; i < n; ++i)*

*{*

*if (head == NULL)*

*head = p = new Node();*

*else*

*{*

*p->next = new Node();*

*p = p->next;*

*}*

*p->data = arr[i];*

*p->next = p->child = NULL;*

*}*

*return head;*

*}*

*Node \*createList(void)*

*{*

*int arr1[] = {10, 5, 12, 7, 11};*

*int arr2[] = {4, 20, 13};*

*int arr3[] = {17, 6};*

*int arr4[] = {9, 8};*

*int arr5[] = {19, 15};*

*int arr6[] = {2};*

*int arr7[] = {16};*

*int arr8[] = {3};*

*Node \*head1 = createList(arr1, SIZE(arr1));*

*Node \*head2 = createList(arr2, SIZE(arr2));*

*Node \*head3 = createList(arr3, SIZE(arr3));*

*Node \*head4 = createList(arr4, SIZE(arr4));*

*Node \*head5 = createList(arr5, SIZE(arr5));*

*Node \*head6 = createList(arr6, SIZE(arr6));*

*Node \*head7 = createList(arr7, SIZE(arr7));*

*Node \*head8 = createList(arr8, SIZE(arr8));*

*head1->child = head2;*

*head1->next->next->next->child = head3;*

*head3->child = head4;*

*head4->child = head5;*

*head2->next->child = head6;*

*head2->next->next->child = head7;*

*head7->child = head8;*

*return head1;*

*}*

*int main(void)*

*{*

*Node \*head = NULL;*

*head = createList();*

*return 0;*

*}*

***Q2)***

***1)***

*include <iostream>*

*#include <queue>*

*using namespace std;*

*struct Node {*

*int data;*

*Node\* left;*

*Node\* right;*

*};*

*Node\* CreateNode(int data)*

*{*

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

*if (!newNode) {*

*cout << "Memory error\n";*

*return NULL;*

*}*

*newNode->data = data;*

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

*return newNode;*

*}*

*Node\* InsertNode(Node\* root, int data)*

*{*

*if (root == NULL) {*

*root = CreateNode(data);*

*return root;*

*}*

*queue<Node\*> q;*

*q.push(root);*

*while (!q.empty()) {*

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

*q.pop();*

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

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

*else {*

*temp->left = CreateNode(data);*

*return root;*

*}*

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

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

*else {*

*temp->right = CreateNode(data);*

*return root;*

*}*

*}*

*}*

*void inorder(Node\* temp)*

*{*

*if (temp == NULL)*

*return;*

*inorder(temp->left);*

*cout << temp->data << ' ';*

*inorder(temp->right);*

*}*

*// Driver code*

*int main()*

*{*

*Node\* root = CreateNode(10);*

*root->left = CreateNode(11);*

*root->left->left = CreateNode(7);*

*root->right = CreateNode(9);*

*root->right->left = CreateNode(15);*

*root->right->right = CreateNode(8);*

*cout << "Inorder traversal before insertion: ";*

*inorder(root);*

*cout << endl;*

*int key = 12;*

*root = InsertNode(root, key);*

*cout << "Inorder traversal after insertion: ";*

*inorder(root);*

*cout << endl;*

*return 0;*

*}*

***2)***

*#include <iostream>*

*using namespace std;*

*struct Node {*

*int data;*

*int rcount;*

*int lcount;*

*struct Node\* left;*

*struct Node\* right;*

*};*

*bool isPBT(int count)*

*{*

*count = count + 1;*

*while (count % 2 == 0)*

*count = count / 2;*

*if (count == 1)*

*return true;*

*else*

*return false;*

*}*

*struct Node\* newNode(int data)*

*{*

*struct Node\* temp =*

*(struct Node\*)malloc(*

*sizeof(struct Node)*

*);*

*temp->data = data;*

*temp->right = NULL;*

*temp->left = NULL;*

*temp->rcount = 0;*

*temp->lcount = 0;*

*}*

*struct Node\* insert(struct Node\* root,*

*int data)*

*{*

*if (root == NULL) {*

*struct Node\* n = newNode(data);*

*return n;*

*}*

*if (root->rcount == root->lcount) {*

*root->left = insert(root->left, data);*

*root->lcount += 1;*

*}*

*else if (root->rcount < root->lcount) {*

*if (isPBT(root->lcount)) {*

*root->right = insert(root->right, data);*

*root->rcount += 1;*

*}*

*else {*

*root->left = insert(root->left, data);*

*root->lcount += 1;*

*}*

*}*

*return root;*

*}*

*void inorder(struct Node\* root)*

*{*

*if (root != NULL) {*

*inorder(root->left);*

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

*inorder(root->right);*

*}*

*}*

*// Driver Code*

*int main()*

*{*

*int arr[] = { 8, 6, 7, 12, 5, 1, 9 };*

*int size = sizeof(arr) / sizeof(int);*

*struct Node\* root = NULL;*

*// Loop to insert nodes in*

*// Binary Tree in level order*

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

*root = insert(root, arr[i]);*

*inorder(root);*

*return 0;*

*}*

***3)***

*#include <bits/stdc++.h>*

*using namespace std;*

*class node {*

*public:*

*int data;*

*node\* left;*

*node\* right;*

*};*

*int maxDepth(node\* node)*

*{*

*if (node == NULL)*

*return 0;*

*else {*

*int lDepth = maxDepth(node->left);*

*int rDepth = maxDepth(node->right);*

*if (lDepth > rDepth)*

*return (lDepth + 1);*

*else*

*return (rDepth + 1);*

*}*

*}*

*node\* newNode(int data)*

*{*

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

*Node->data = data;*

*Node->left = NULL;*

*Node->right = NULL;*

*return (Node);*

*}*

*int main()*

*{*

*node\* root = newNode(1);*

*root->left = newNode(2);*

*root->right = newNode(3);*

*root->left->left = newNode(4);*

*root->left->right = newNode(5);*

*cout << "Height of tree is " << maxDepth(root);*

*return 0;*

*}*

***4)***

*#include <bits/stdc++.h>*

*using namespace std;*

*class TreeNode {*

*public:*

*int val;*

*TreeNode\* left;*

*TreeNode\* right;*

*TreeNode(int data)*

*{*

*val = data;*

*left = NULL;*

*right = NULL;*

*}*

*};*

*bool isThereDuplicate(TreeNode\* root)*

*{*

*if (!root)*

*return false;*

*queue<TreeNode\*> q;*

*q.push(root);*

*set<int> hash;*

*while (!q.empty()) {*

*TreeNode\* cur = q.front();*

*q.pop();*

*if (hash.find(cur->val) != hash.end())*

*return true;*

*hash.insert(cur->val);*

*if (cur->left)*

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

*if (cur->right)*

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

*}*

*return false;*

*}*

*int main()*

*{*

*TreeNode\* root1 = new TreeNode(1);*

*root1->left = new TreeNode(2);*

*root1->right = new TreeNode(3);*

*root1->left->left = new TreeNode(1);*

*if (isThereDuplicate(root1))*

*cout << "Tree in example 1 has duplicate values\n";*

*else*

*cout << "Tree in example 1 doesn't have duplicate values\n";*

*TreeNode\* root2 = new TreeNode(1);*

*root2->left = new TreeNode(2);*

*root2->right = new TreeNode(3);*

*root2->left->left = new TreeNode(4);*

*root2->left->right = new TreeNode(5);*

*root2->right->right = new TreeNode(6);*

*if (isThereDuplicate(root2))*

*cout << "Tree in example 2 has duplicate values\n";*

*else*

*cout << "Tree in example 2 doesn't have duplicate values\n";*

*return 0;*

*}*