1. Design an algorithm to serialize and deserialize a binary tree. Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment. Implement the serialize and deserialize methods.

#include<iostream>

#include <queue>

#include<string>

#include <bits/stdc++.h>

using namespace std;

class TreeNode{

public:

int data;

TreeNode\* right;

TreeNode\* left;

TreeNode(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

// creating tree using level order traversal

TreeNode\* buildTree(TreeNode\* &root){

cout<<"enter the root data"<<endl;

int data;

cin>>data;

root =new TreeNode(data);

queue<TreeNode\*> q;

q.push(root);

while(!q.empty()){

TreeNode\* front=q.front();

q.pop();

cout<<"enter the left of: "<<front->data<<endl;

int leftData;

cin>>leftData;

if(leftData!=-1){

front->left=new TreeNode(leftData);

q.push(front->left);

}

cout<<"enter the right of: "<<front->data<<endl;

int rightData;

cin>>rightData;

if(rightData!=-1){

front->right=new TreeNode(rightData);

q.push(front->right);

}

}

return root;

}

void printLevelOrder(TreeNode\* root){

queue<TreeNode\*> q;

q.push(root);

q.push(NULL);

while(!q.empty()){

TreeNode\* front= q.front();

q.pop();

if(front==NULL){

cout<<endl;

if(!q.empty()) {

q.push(NULL);

}

continue;

}

cout<<front->data<<" ";

if(front->left)

q.push(front->left);

if(front->right)

q.push(front->right);

}

}

string ser(TreeNode\* &root){

if(root==NULL) return "";

string s="";

queue<TreeNode\*> q;

q.push(root);

while(!q.empty()){

TreeNode\* front=q.front();

q.pop();

if(front==NULL){

s=s+"#,";

}

else{

s=s+to\_string(front->data)+',';

}

if(front){

q.push(front->left);

q.push(front->right);

}

}

return s;

}

TreeNode\* deSer(string data){

if (data.size()==0) {

return NULL;

}

stringstream s(data);

string str;

getline(s, str, ',');

TreeNode\* root = new TreeNode(stoi(str));

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

TreeNode\* node = q.front();

q.pop();

getline(s, str, ',');

if (str != "#") {

TreeNode\* leftNode = new TreeNode(stoi(str));

node->left = leftNode;

q.push(leftNode);

}

getline(s, str, ',');

if (str != "#") {

TreeNode\* rightNode = new TreeNode(stoi(str));

node->right = rightNode;

q.push(rightNode);

}

}

return root;

}

// -10 5 6 7 8 9 -1 11 -1 -1 -1 -1 -1 -1 -1

int main(){

TreeNode\* root=NULL;

root= buildTree(root);

printLevelOrder(root);

string s;

s= ser(root);

cout<<s<<endl;

TreeNode\* newRoot=NULL;

newRoot=deSer(s);

printLevelOrder(newRoot);

}

OUTPUT

