```
/*
    File name: binaryTree.cpp
    Created by: Tan Qi Hao
    Created on: 11/1/2019
    Synopsis: This program test the function (eg. size, count,
    height, isSameTree, hasPathSum & isBalanced) for the binary tree.
    If the test succed, it will output the word "Congratulation!".
*/

#include <iostream>
#include <cstddef>
#include <cassert>

using namespace std;

struct TreeNode{
    int data;
    TreeNode *left;
    TreeNode *right;
};

// Function declarations
int size(TreeNode* root);
int count(TreeNode* root, int target);
int height(TreeNode* root);
bool isSameTree(TreeNode* root1, TreeNode* root2);
bool hasPathSum(TreeNode* root, int target);
bool isBalanced(TreeNode* root);

// feel free to define your own helper functions
/* your code here */




// A helper function to build a tree, you do not need to modify it
// Inputs:
//     nodeValues, a list of all node values from top to bottom, left to right. If
no node at one place, use -1 as the placeholder
//     root, a pointer to the root node
//     i, set it to 0 when calling this function from outside
//     n, size of the 'nodeValues' array
// Postcondition: a tree is built with the second input argument as its root
pointer
TreeNode* insertNodes(int nodeValues[], TreeNode* root, int i, int n);

int main(){
    // the binary trees in instruction examples
    TreeNode *root, *root2;
    int nodedatas[] = {1,2,3,4,5,6,7,8,1,-1,-1,-1,-1,-1,1};
    int nodedatas2[] = {1,2,3,4,5,6,7,8,1,-1,-1,-1,-1,1,-1};
    root = insertNodes(nodedatas, root, 0, 15);
    root2 = insertNodes(nodedatas2, root2, 0, 15);

    assert(size(root) == 10);
    assert(size(root->left) == 5);
    assert(count(root,1) == 3);
    assert(count(root,9) == 0);
    assert(height(root) == 3);
    assert(height(root->left->right) == 0);
    assert(hasPathSum(root,10) == true);
    assert(hasPathSum(root,1) == false);
    assert(isBalanced(root) == true);
    assert(isSameTree(root,root2) == false);
    assert(isSameTree(root->left,root2->left) == true);

    cout << "Congratulation!" << endl;
    return 0;
}

// A helper function to build the test cases. You do not need to modify it.
TreeNode* insertNodes(int nodeValues[], TreeNode* root, int i, int n){
    if(i<n && nodeValues[i]!=-1){
        root = new TreeNode;
        root->data = nodeValues[i];
        root->left = insertNodes(nodeValues, root->left, 2*i+1, n);
        root->right = insertNodes(nodeValues, root->right, 2*i+2, n);
    }
    else{
        root = NULL;
    }
    return root;
}

//This function calculate the size of the binary tree
int size(TreeNode* root){

  //initialize the size on left and right
  int sizeLeft = 0;
  int sizeRight = 0;

  //Recursion stop when root is NULL
  if(root == NULL){
    return 0;

  }else{

    //Find the size on left and right
    sizeLeft = size(root -> left);
    sizeRight = size(root -> right);

    //Return the total size
    return sizeLeft + sizeRight + 1;

  }
}

//This function the amount of parameter target in the tree
int count(TreeNode* root, int target){

  //Recursion stop when root is NULL
  if (root == NULL){

    return 0;

  }else{

    //Determine whether or not the data is the target
    if (root -> data == target){

      //Add 1 when it is the target
      return 1 + count(root -> left, target) + count(root -> right, target);

    }else{

      //Remain the same, when it is not the target
      return count(root -> left, target) + count(root -> right, target);

    }

  }

}

//This function calculate the height
int height(TreeNode* root){

  int sumheight = 0;
  int heightLeft = 0;
  int heightRight = 0;

  if(root == NULL){

    sumheight = -1;

  }else{
    //Find the height on left and right
    heightLeft = height(root -> left);
    heightRight = height(root -> right);

    //Add 1 to the side with the largest height
    if(heightLeft > heightRight){
      sumheight = heightLeft + 1;

    }else{

      sumheight = heightRight + 1;
    }


  }
  //return the height
  return sumheight;

}

//This function determine whether or not both tree are the same
bool isSameTree(TreeNode* root1, TreeNode* root2){

  //Both root are NULL
  if(root1 == NULL && root2 == NULL){

    return true;

    //if one of the roots are NULL, it is not the same
  }else if((root1 != NULL && root2 == NULL) || (root1 == NULL && root2 != NULL)){
    return false;
    //Both root are not NULL
  }else if(root1 != NULL && root2 != NULL){

    //Determine the next node until root are NULL
    bool testLeft = isSameTree(root1 -> left, root2 -> left);
    bool testRight = isSameTree(root1 -> right, root2 -> right);

    //Determination if all node the same
    if((root1 -> data == root2 -> data) && testLeft && testRight){

      return true;

    }else{

      return false;
    }


  }

}

//This function determine whether or not the sum of path equal to the parameter
target
bool hasPathSum(TreeNode* root, int target){

  //Return the determination of target, when root is NULL
  if(root == NULL){

    if(target == 0){
      return true;

    }else{

      return false;

    }

  }else{

    //if target is zero and the left and right are NULL, return true
    if(target - (root -> data) == 0 && root -> left == NULL
       && root -> right == NULL){

      return true;

      //Set condition when right is NULL
    }else if(root -> left != NULL && root -> right == NULL){

      //return the determination of left
      return hasPathSum(root -> left, target - (root -> data));

      //Set condition when left is NULL
    }else if(root -> left == NULL && root -> right != NULL){

      //Return the determination of right
      return hasPathSum(root -> right, target - (root -> data));
      //Set condition when left and right are not NULL
    }else if(root -> left != NULL && root -> right != NULL){

      //Return true when one of the left and right target is zero
      return hasPathSum(root -> left, target - (root -> data)) ||  hasPathSum(root
-> right, target - (root -> data));

    }else{

      return false;

    }
  }
}

//This function determine whether or not the tree is balanced.
bool isBalanced(TreeNode* root){

  //Return true, when root is NULL
  if(root == NULL){

    return true;

  }else{

    //Find the height difference
    int heightdifference = height(root -> left) - height(root -> right);

    //Make sure the heightdifference is positive
    if(heightdifference < 0){

      heightdifference = heightdifference * (-1);

    }

    /*Make sure that heightdifference smaller than 2 and next height of
      on the left and right are all true.
     */
    return heightdifference < 2 && isBalanced(root -> left) && isBalanced(root ->
right);

  }

}
```