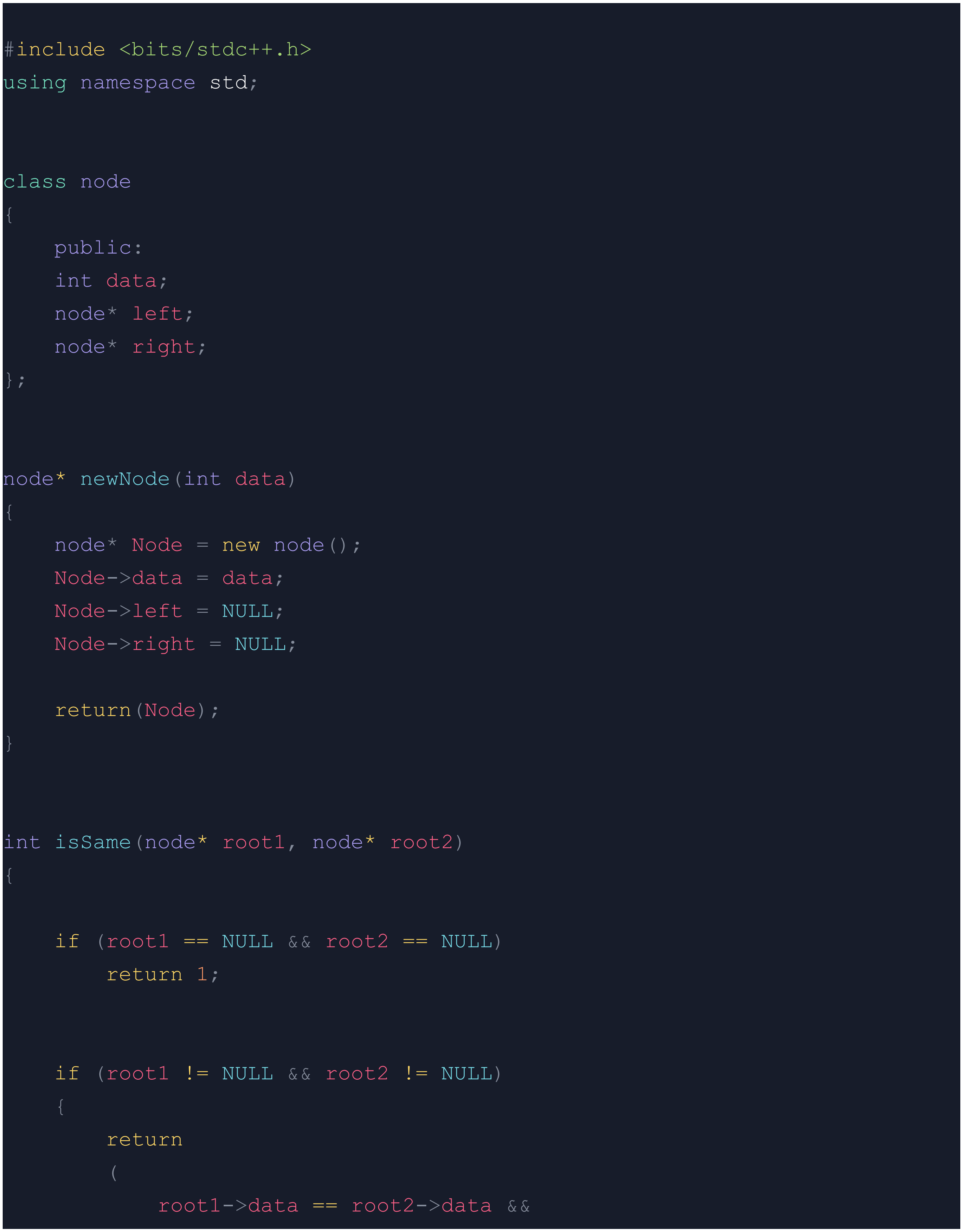
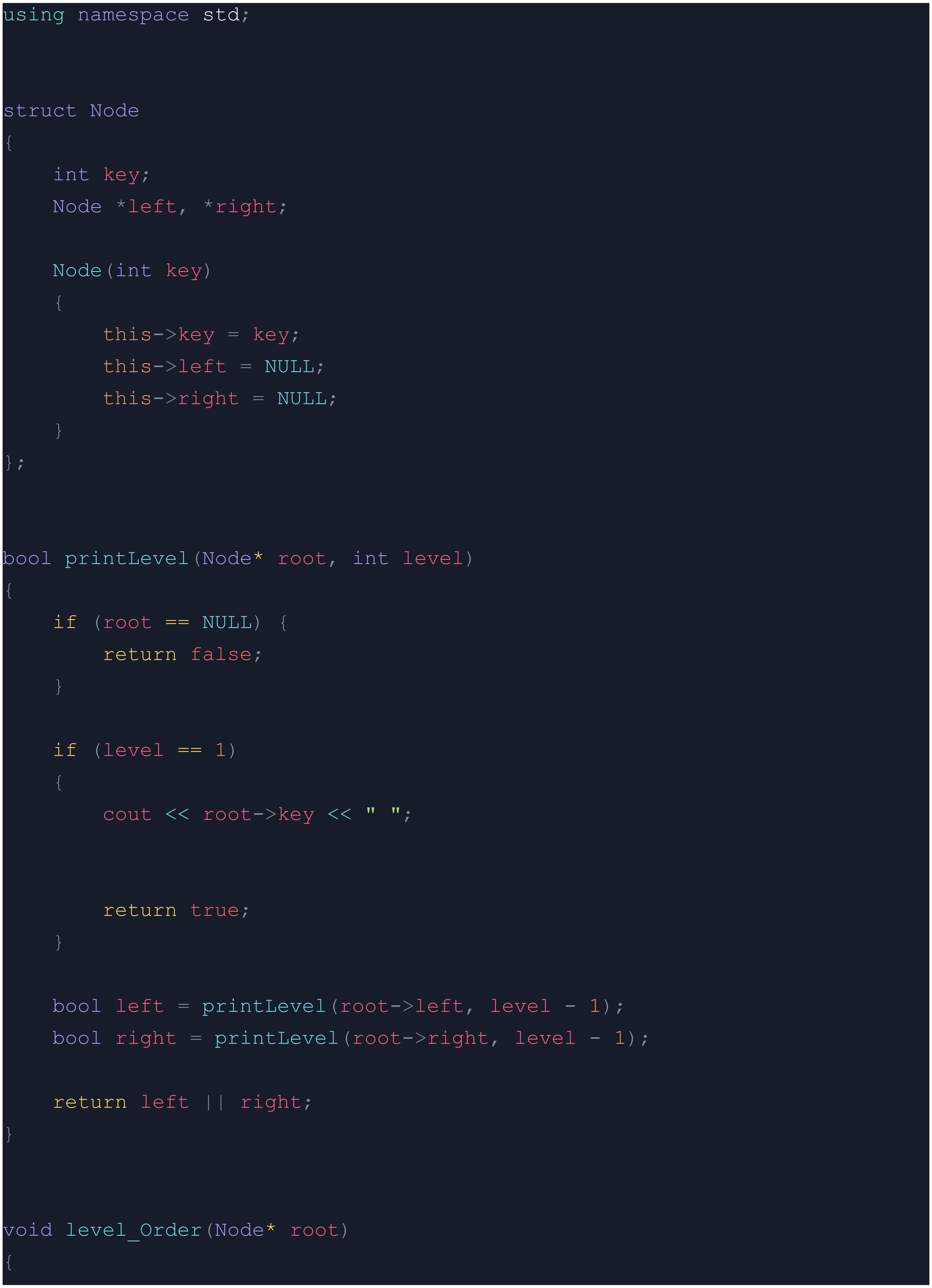
Answer to the question No : 01



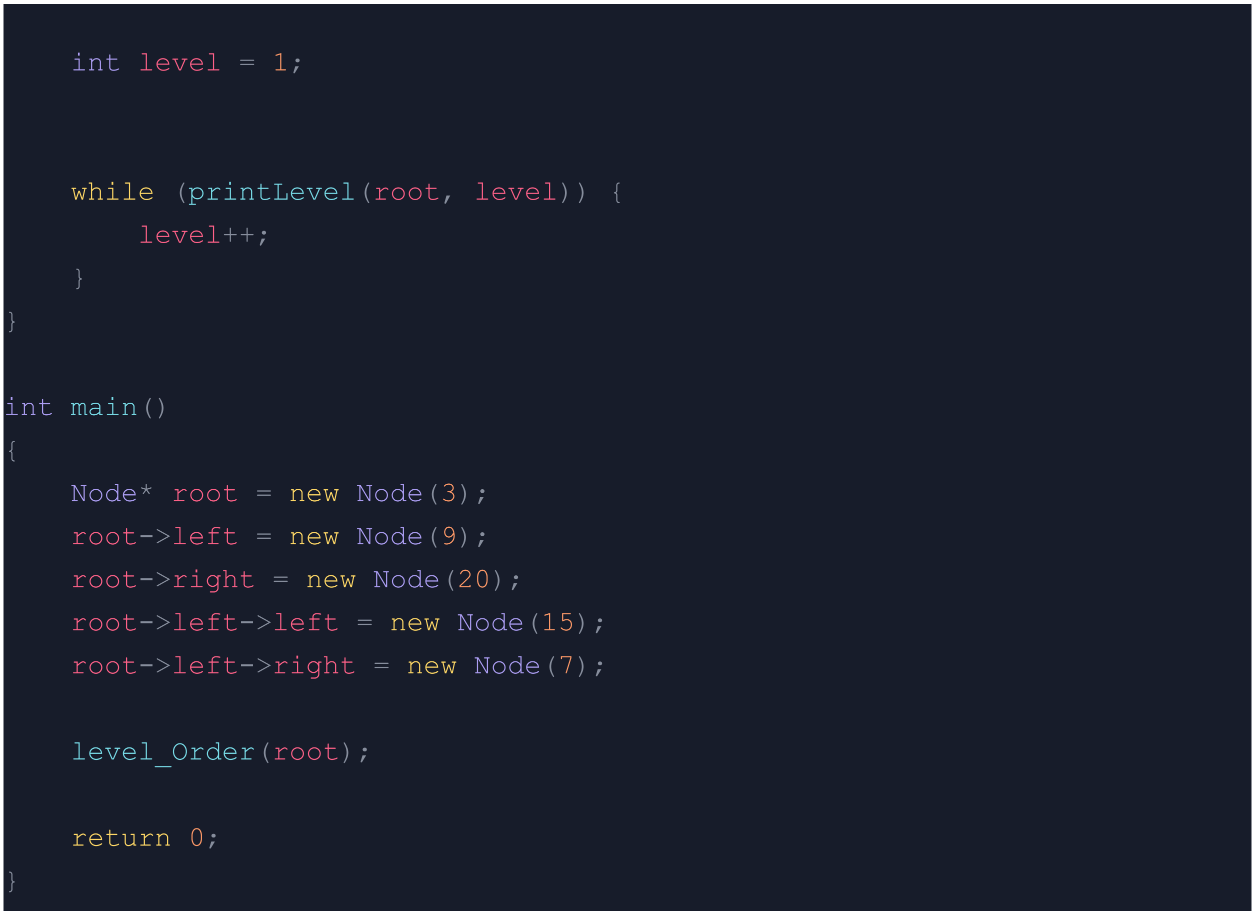
```
#include <bits/stdc++.h>
using namespace std;


class node
{
    public:
    int data;
    node* left;
    node* right;
};


node* newNode(int data)
{
    node* Node = new node();
    Node->data = data;
    Node->left = NULL;
    Node->right = NULL;

    return(Node);
}


int isSame(node* root1, node* root2)
{

    if (root1 == NULL && root2 == NULL)
        return 1;


    if (root1 != NULL && root2 != NULL)
    {
        return
        (
            root1->data == root2->data &&
            isSame(root1->left, root2->left) &&
            isSame(root1->right, root2->right)
        );
    }


    return 0;
}



int main()
{
    node *root1 = newNode(1);
    node *root2 = newNode(1);
    root1->left = newNode(2);
    root1->right = newNode(3);
    root1->left->left = newNode(4);
    root1->left->right = newNode(5);

    root2->left = newNode(2);
    root2->right = newNode(3);
    root2->left->left = newNode(4);
    root2->left->right = newNode(5);

    if(isSame(root1, root2))
        cout << "Both tree are Same."<<endl;
    else
        cout << "Trees are not Same."<<endl;

return 0;
}
```

Answer to the question No : 02

```
#include <iostream>
```

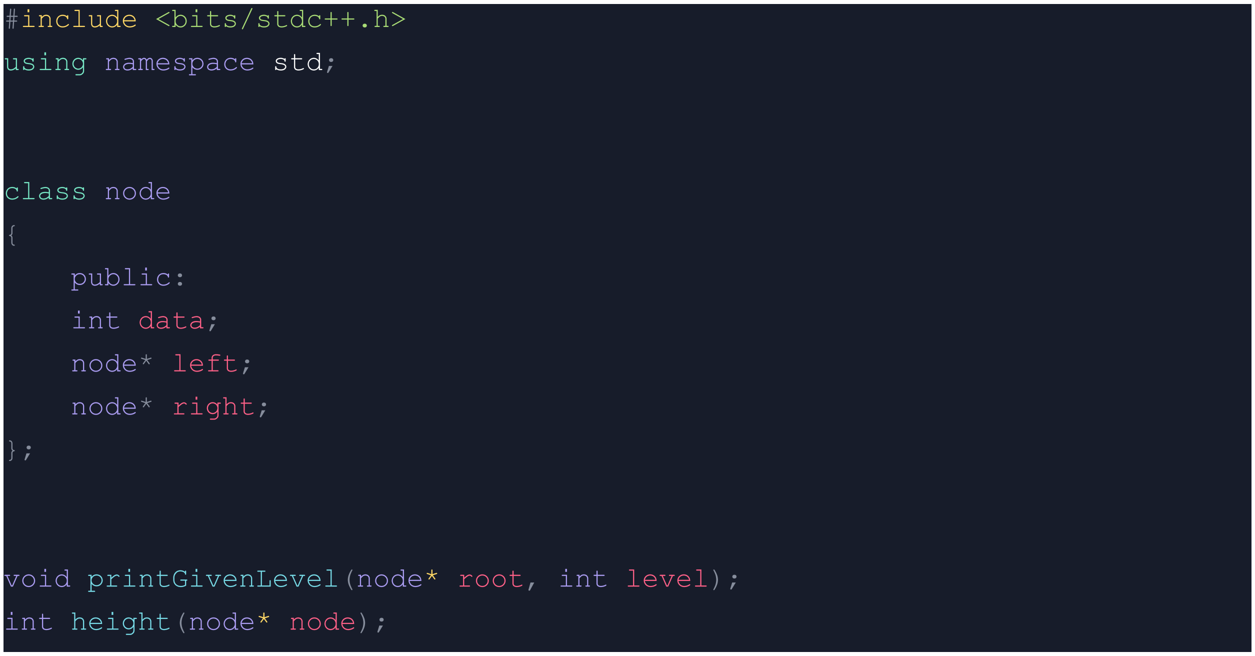


```
using namespace std;


struct Node
{
    int key;
    Node *left, *right;

    Node(int key)
    {
        this->key = key;
        this->left = NULL;
        this->right = NULL;
    }
};


bool printLevel(Node* root, int level)
{
    if (root == NULL) {
        return false;
    }

    if (level == 1)
    {
        cout << root->key << " ";


        return true;
    }

    bool left = printLevel(root->left, level - 1);
    bool right = printLevel(root->right, level - 1);

    return left || right;
}


void level_Order(Node* root)
{
```

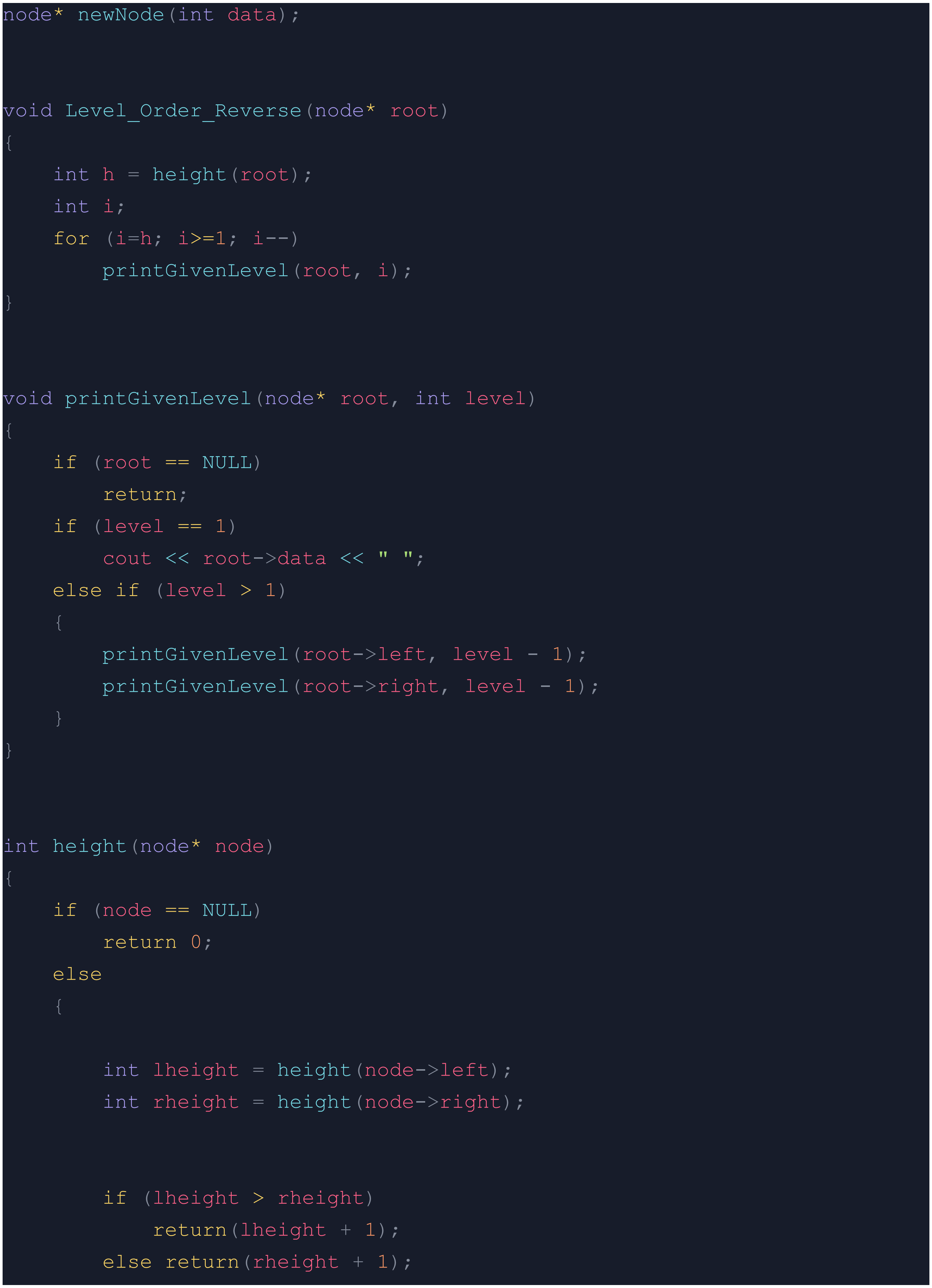


```
    int level = 1;


    while (printLevel(root, level)) {
        level++;
    }
}

int main()
{
    Node* root = new Node(3);
    root->left = new Node(9);
    root->right = new Node(20);
    root->left->left = new Node(15);
    root->left->right = new Node(7);

    level_Order(root);

    return 0;
}
```

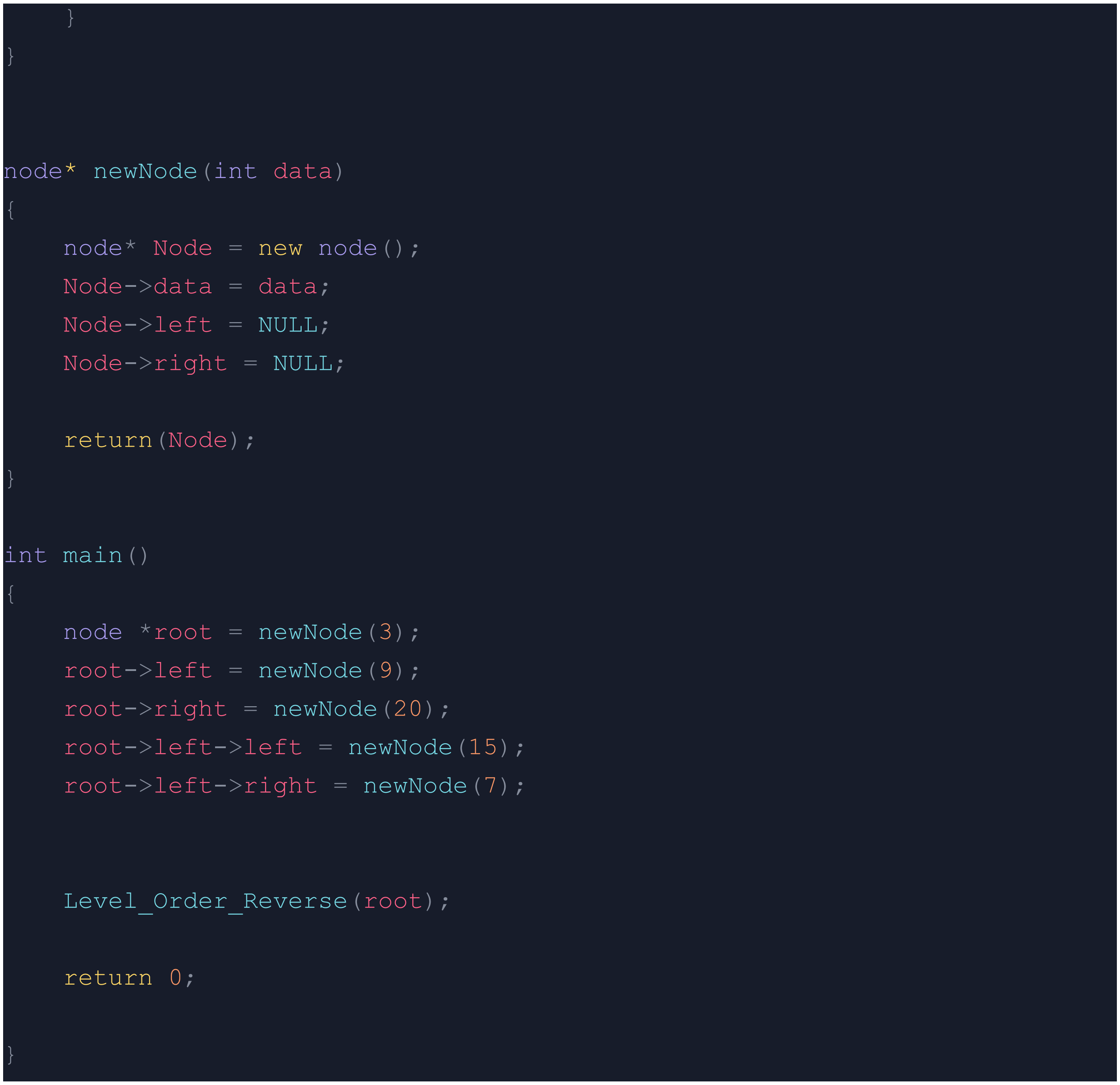
Answer to the question No : 03



```
#include <bits/stdc++.h>
using namespace std;


class node
{
    public:
    int data;
    node* left;
    node* right;
};


void printGivenLevel(node* root, int level);
int height(node* node);
```

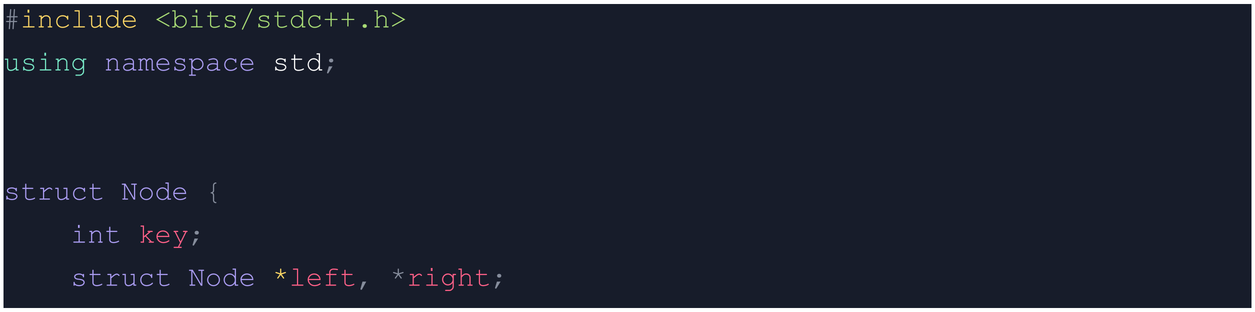


```
node* newNode(int data);


void Level_Order_Reverse(node* root)
{
    int h = height(root);
    int i;
    for (i=h; i>=1; i--)
        printGivenLevel(root, i);
}


void printGivenLevel(node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        cout << root->data << " ";
    else if (level > 1)
    {
        printGivenLevel(root->left, level - 1);
        printGivenLevel(root->right, level - 1);
    }
}


int height(node* node)
{
    if (node == NULL)
        return 0;
    else
    {

        int lheight = height(node->left);
        int rheight = height(node->right);


        if (lheight > rheight)
            return(lheight + 1);
        else return(rheight + 1);
```

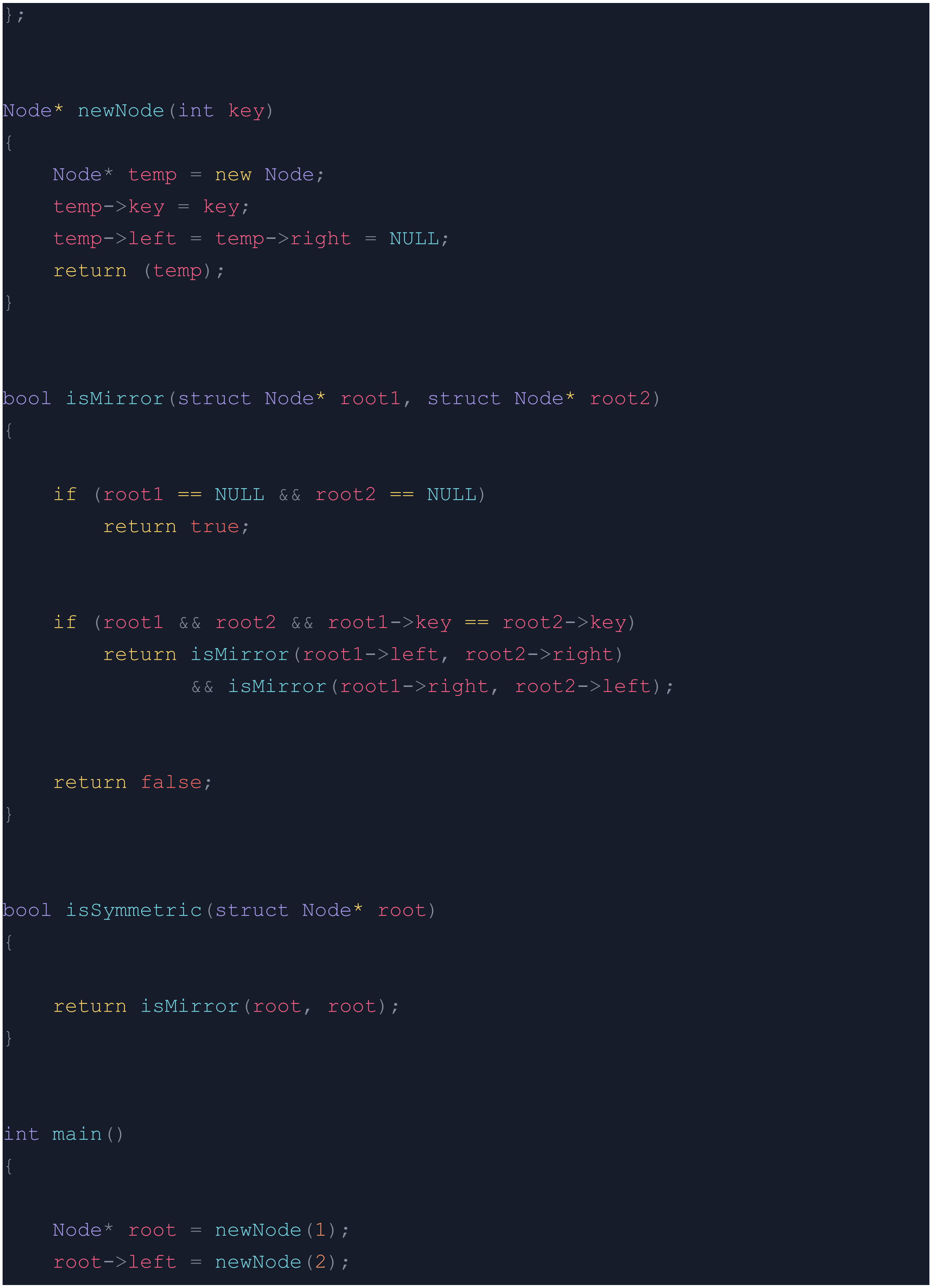


```
    }
}


node* newNode(int data)
{
    node* Node = new node();
    Node->data = data;
    Node->left = NULL;
    Node->right = NULL;

    return(Node);
}

int main()
{
    node *root = newNode(3);
    root->left = newNode(9);
    root->right = newNode(20);
    root->left->left = newNode(15);
    root->left->right = newNode(7);


    Level_Order_Reverse(root);

    return 0;

}
```

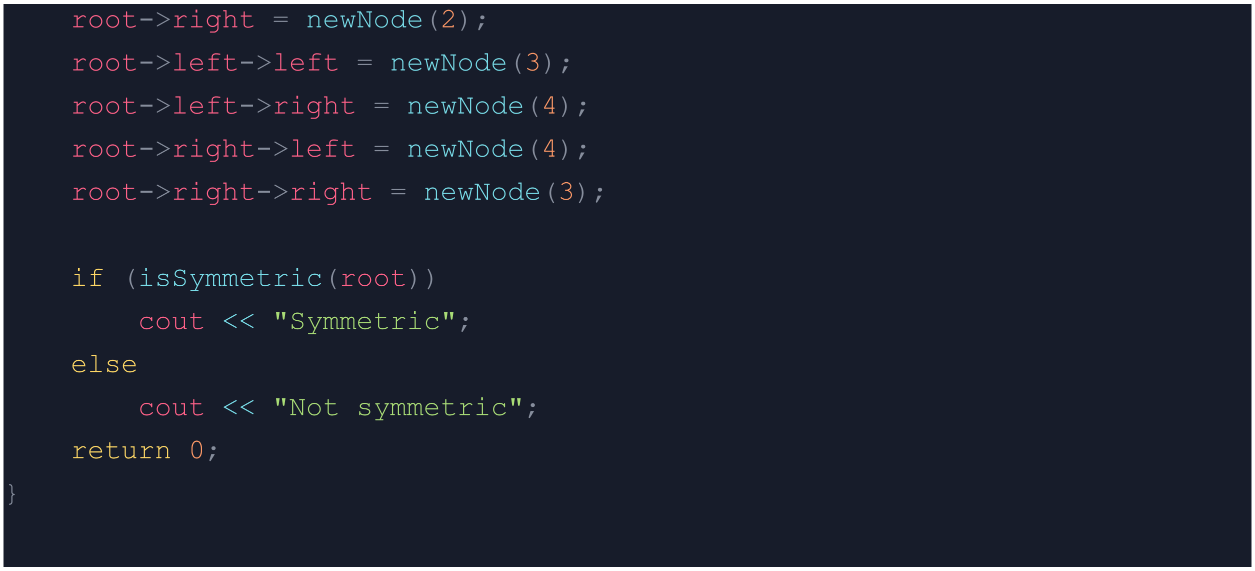
Answer to the question No : 04



```
#include <bits/stdc++.h>
using namespace std;


struct Node {
    int key;
    struct Node *left, *right;
```

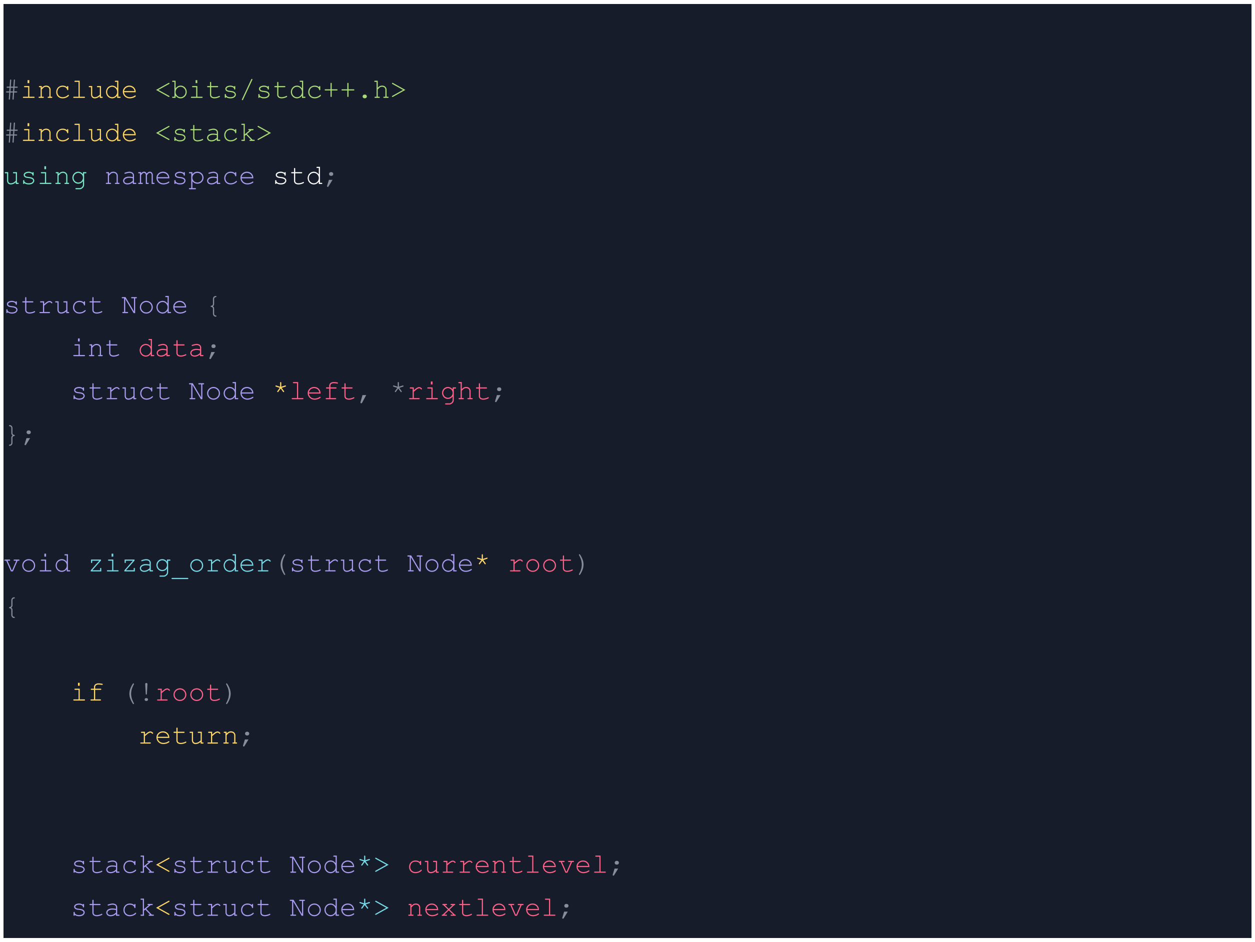


```
};


Node* newNode(int key)
{
    Node* temp = new Node;
    temp->key = key;
    temp->left = temp->right = NULL;
    return (temp);
}


bool isMirror(struct Node* root1, struct Node* root2)
{

    if (root1 == NULL && root2 == NULL)
        return true;


    if (root1 && root2 && root1->key == root2->key)
        return isMirror(root1->left, root2->right)
               && isMirror(root1->right, root2->left);


    return false;
}


bool isSymmetric(struct Node* root)
{

    return isMirror(root, root);
}


int main()
{

    Node* root = newNode(1);
    root->left = newNode(2);
```

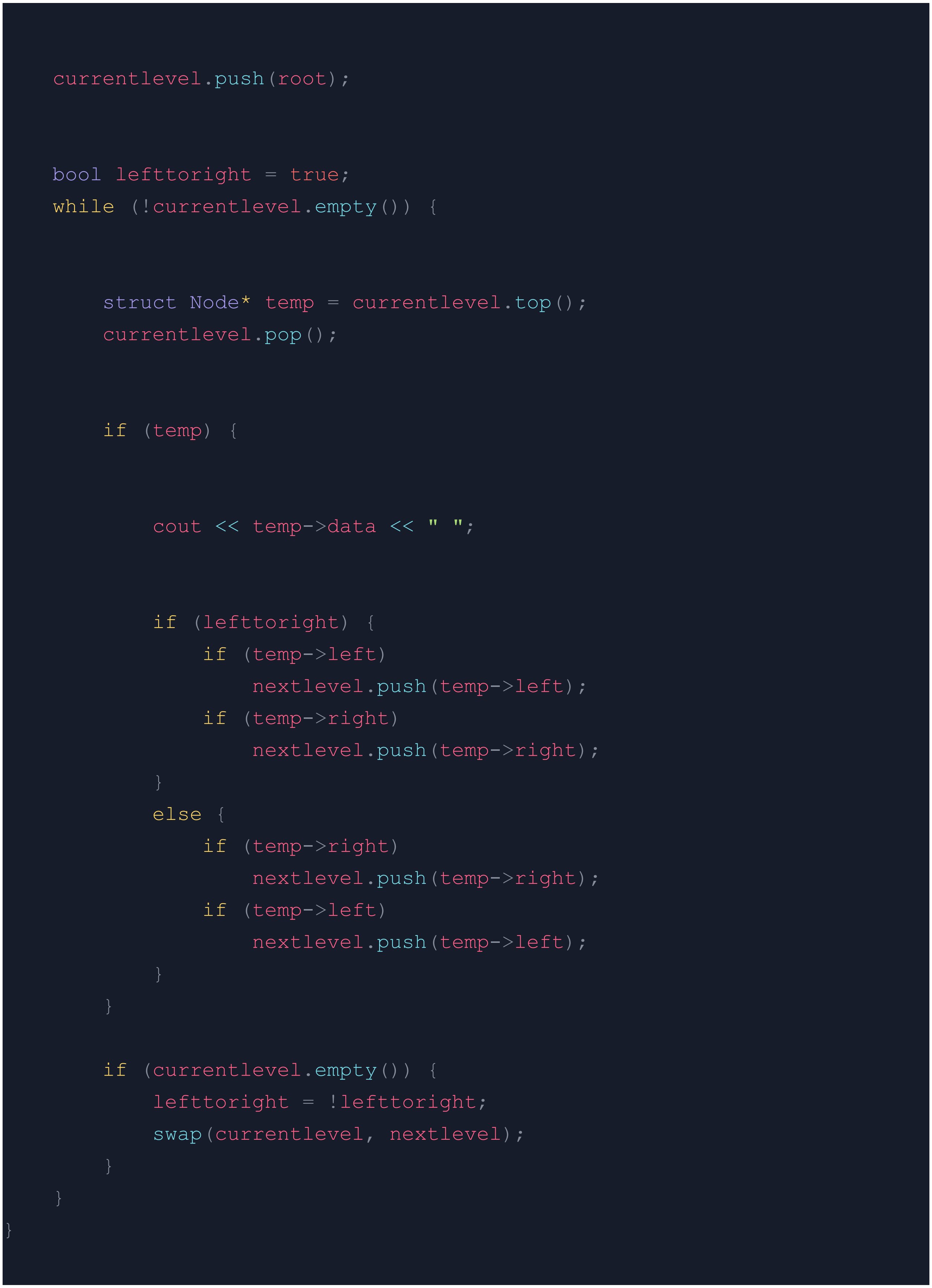


```
    root->right = newNode(2);
    root->left->left = newNode(3);
    root->left->right = newNode(4);
    root->right->left = newNode(4);
    root->right->right = newNode(3);

    if (isSymmetric(root))
        cout << "Symmetric";
    else
        cout << "Not symmetric";
    return 0;
}
```

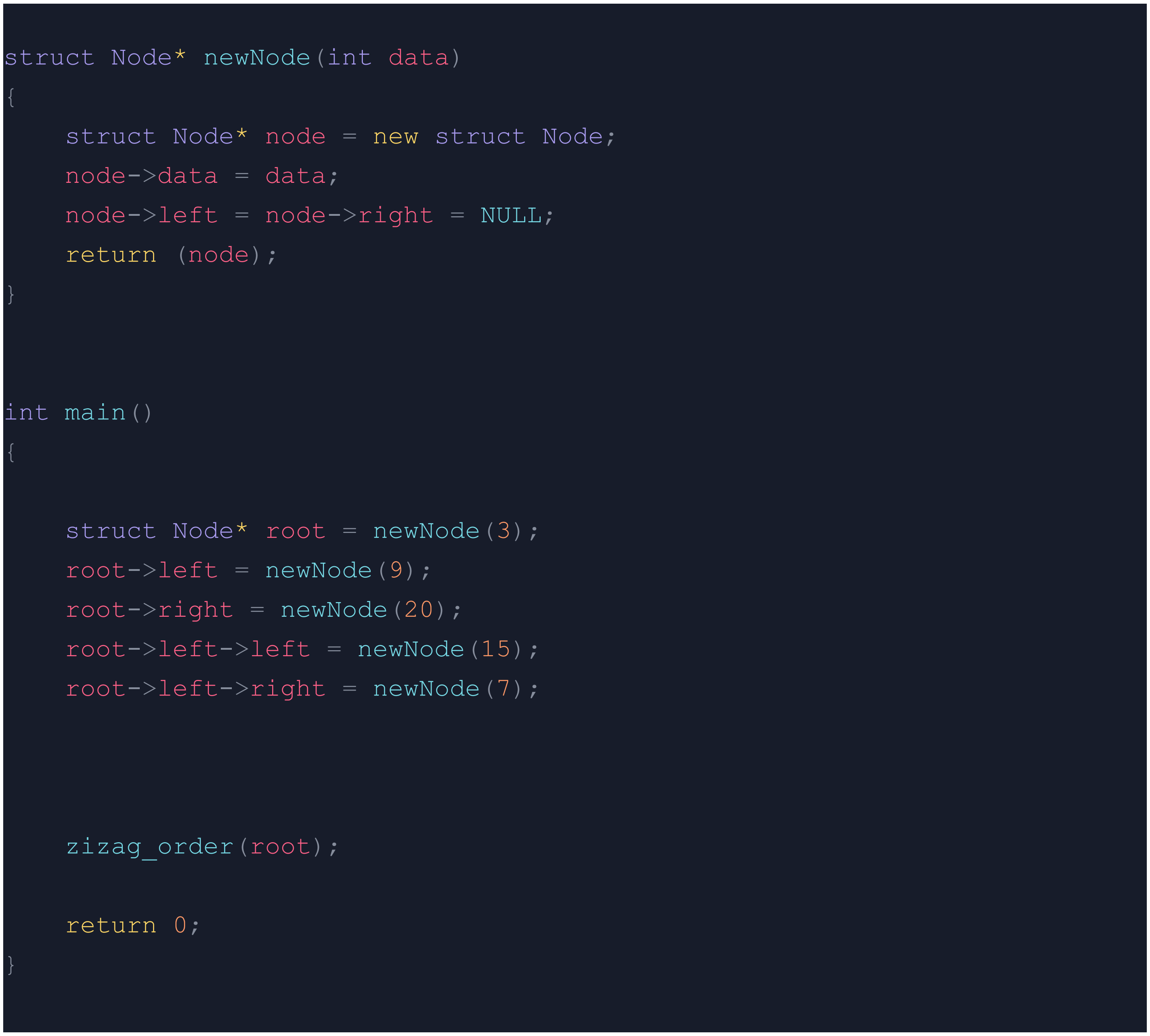
Answer to the question No : 05



```


#include <bits/stdc++.h>
#include <stack>
using namespace std;


struct Node {
    int data;
    struct Node *left, *right;
};


void zizag_order(struct Node* root)
{

    if (!root)
        return;


    stack<struct Node*> currentlevel;
    stack<struct Node*> nextlevel;
```

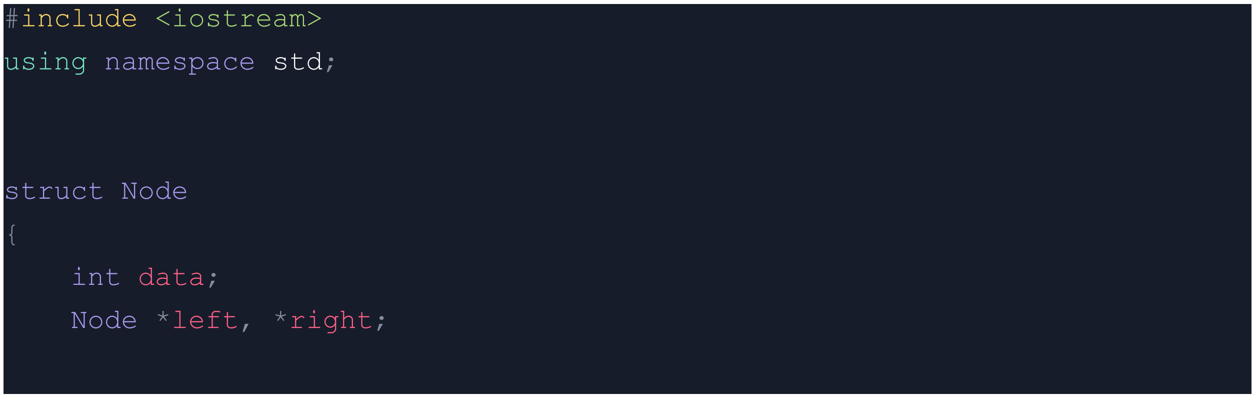


```
    currentlevel.push(root);


    bool lefttoright = true;
    while (!currentlevel.empty()) {


        struct Node* temp = currentlevel.top();
        currentlevel.pop();


        if (temp) {


            cout << temp->data << " ";


            if (lefttoright) {
                if (temp->left)
                    nextlevel.push(temp->left);
                if (temp->right)
                    nextlevel.push(temp->right);
            }
            else {
                if (temp->right)
                    nextlevel.push(temp->right);
                if (temp->left)
                    nextlevel.push(temp->left);
            }
        }

        if (currentlevel.empty()) {
            lefttoright = !lefttoright;
            swap(currentlevel, nextlevel);
        }
    }
}
```

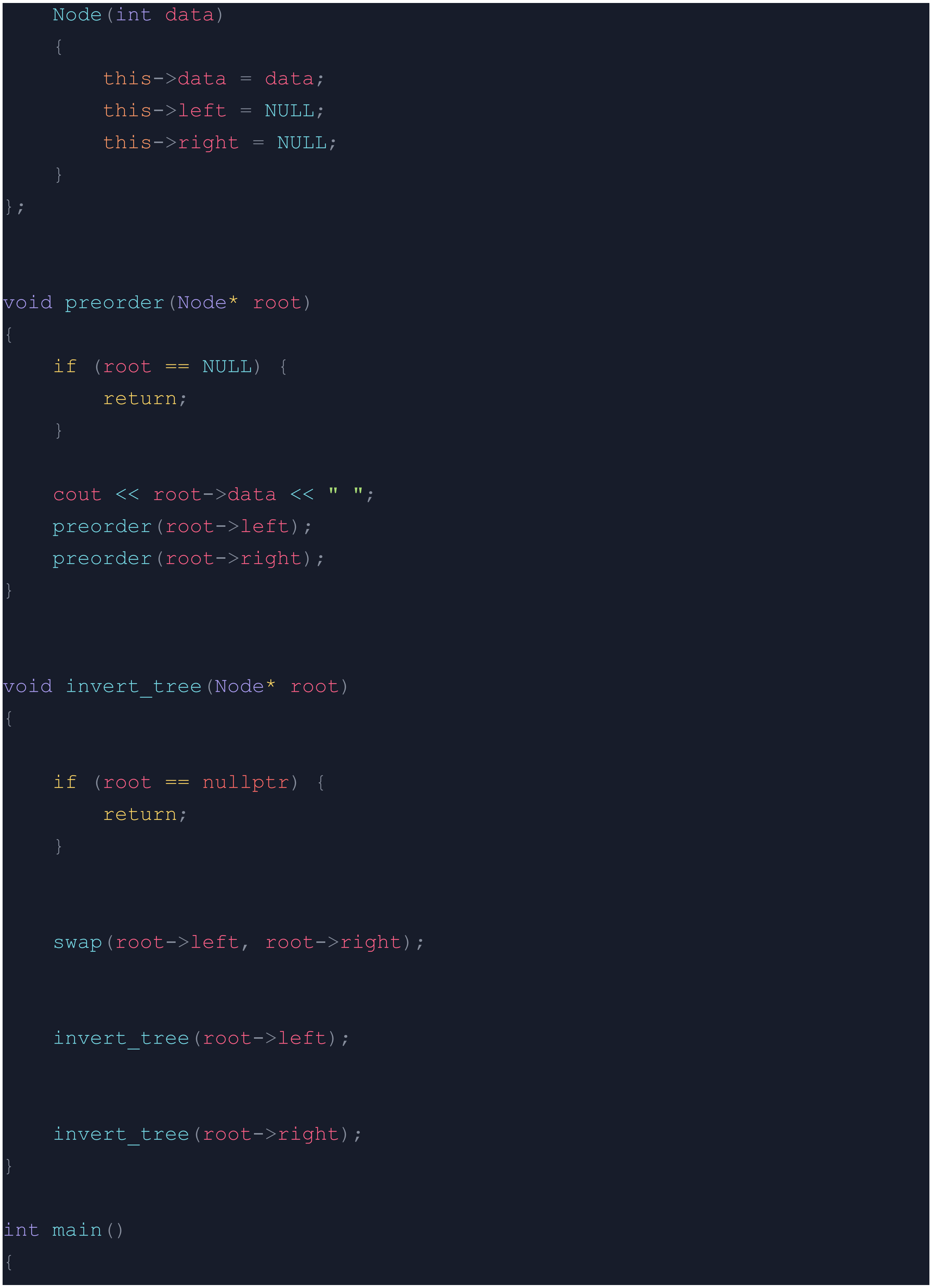


```
struct Node* newNode(int data)
{
    struct Node* node = new struct Node;
    node->data = data;
    node->left = node->right = NULL;
    return (node);
}


int main()
{

    struct Node* root = newNode(3);
    root->left = newNode(9);
    root->right = newNode(20);
    root->left->left = newNode(15);
    root->left->right = newNode(7);



    zizag_order(root);

    return 0;
}
```

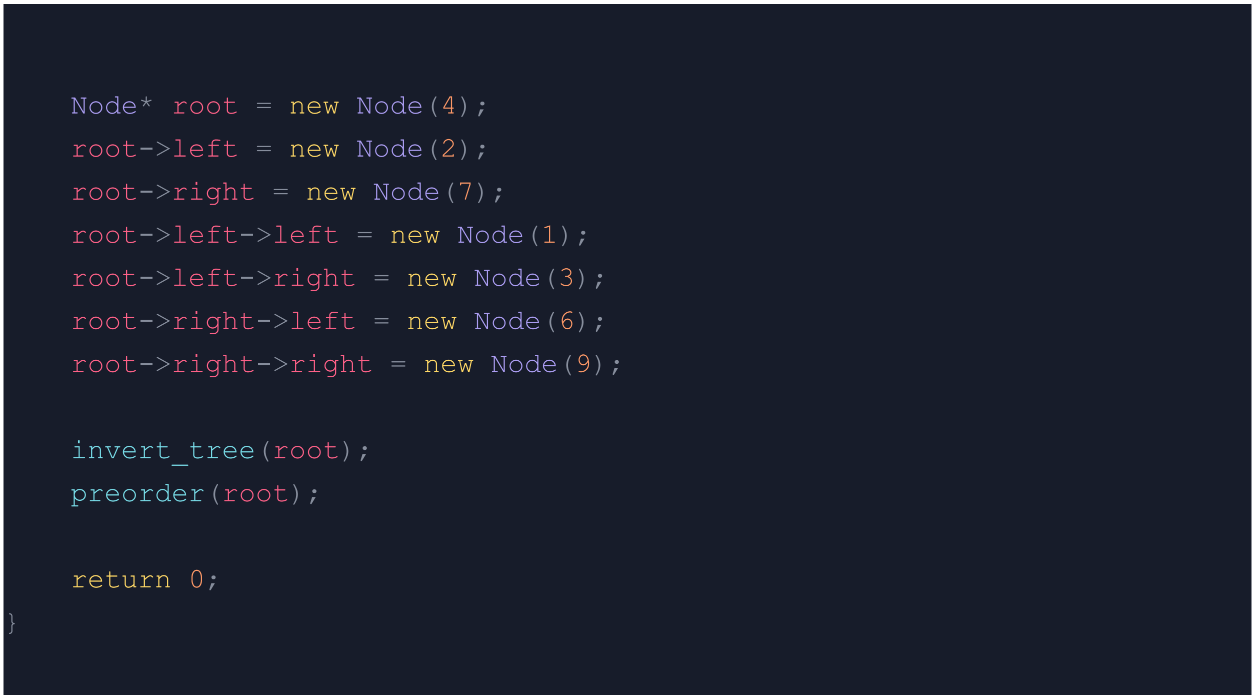
Answer to the question No : 06



```
#include <iostream>
using namespace std;


struct Node
{
    int data;
    Node *left, *right;
```

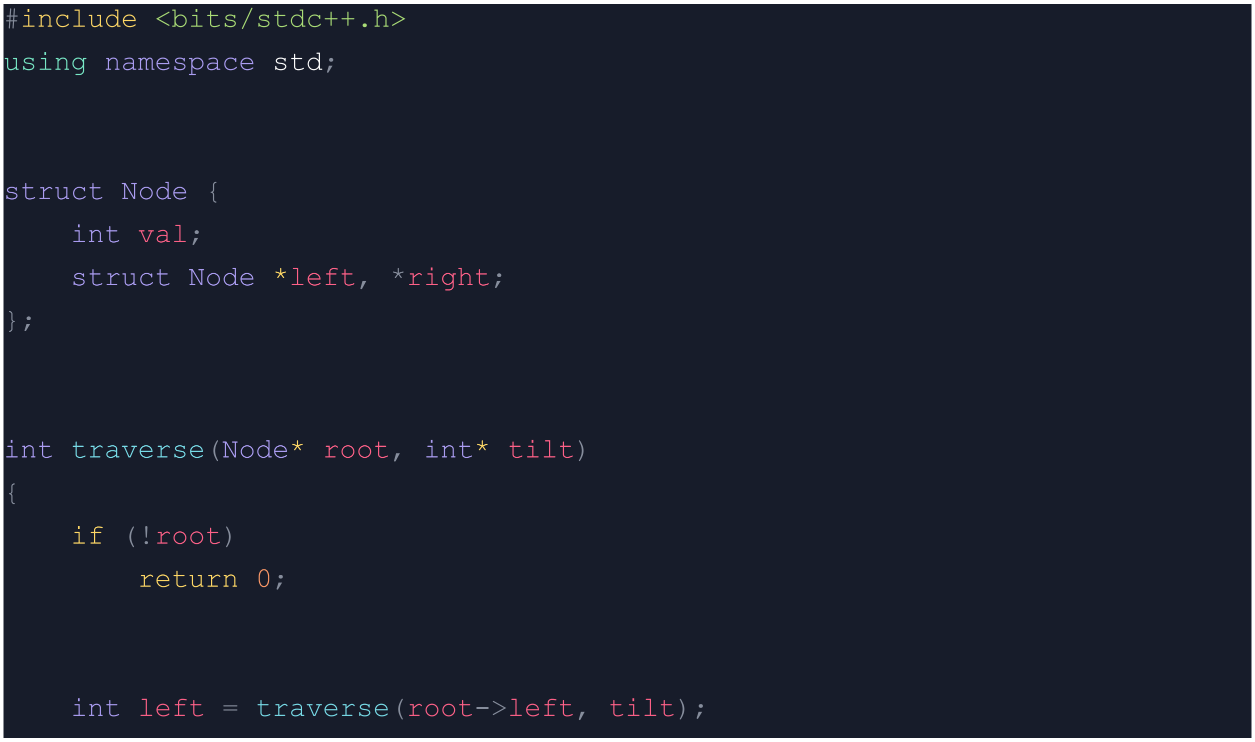


```
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};


void preorder(Node* root)
{
    if (root == NULL) {
        return;
    }

    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}


void invert_tree(Node* root)
{

    if (root == nullptr) {
        return;
    }


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


    invert_tree(root->left);


    invert_tree(root->right);
}

int main()
{
```



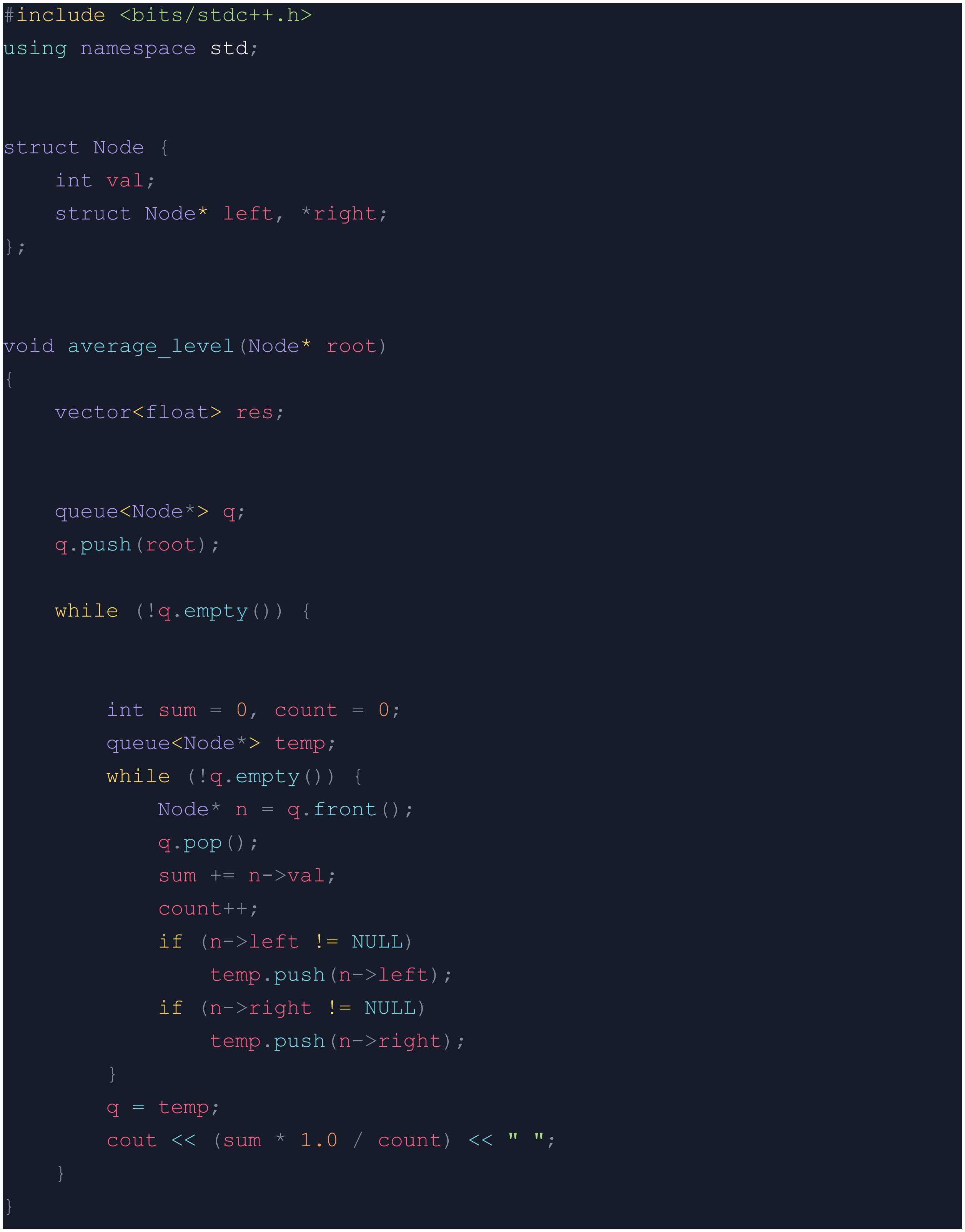
```
    Node* root = new Node(4);
    root->left = new Node(2);
    root->right = new Node(7);
    root->left->left = new Node(1);
    root->left->right = new Node(3);
    root->right->left = new Node(6);
    root->right->right = new Node(9);

    invert_tree(root);
    preorder(root);

    return 0;
}
```

Answer to the question No : 07

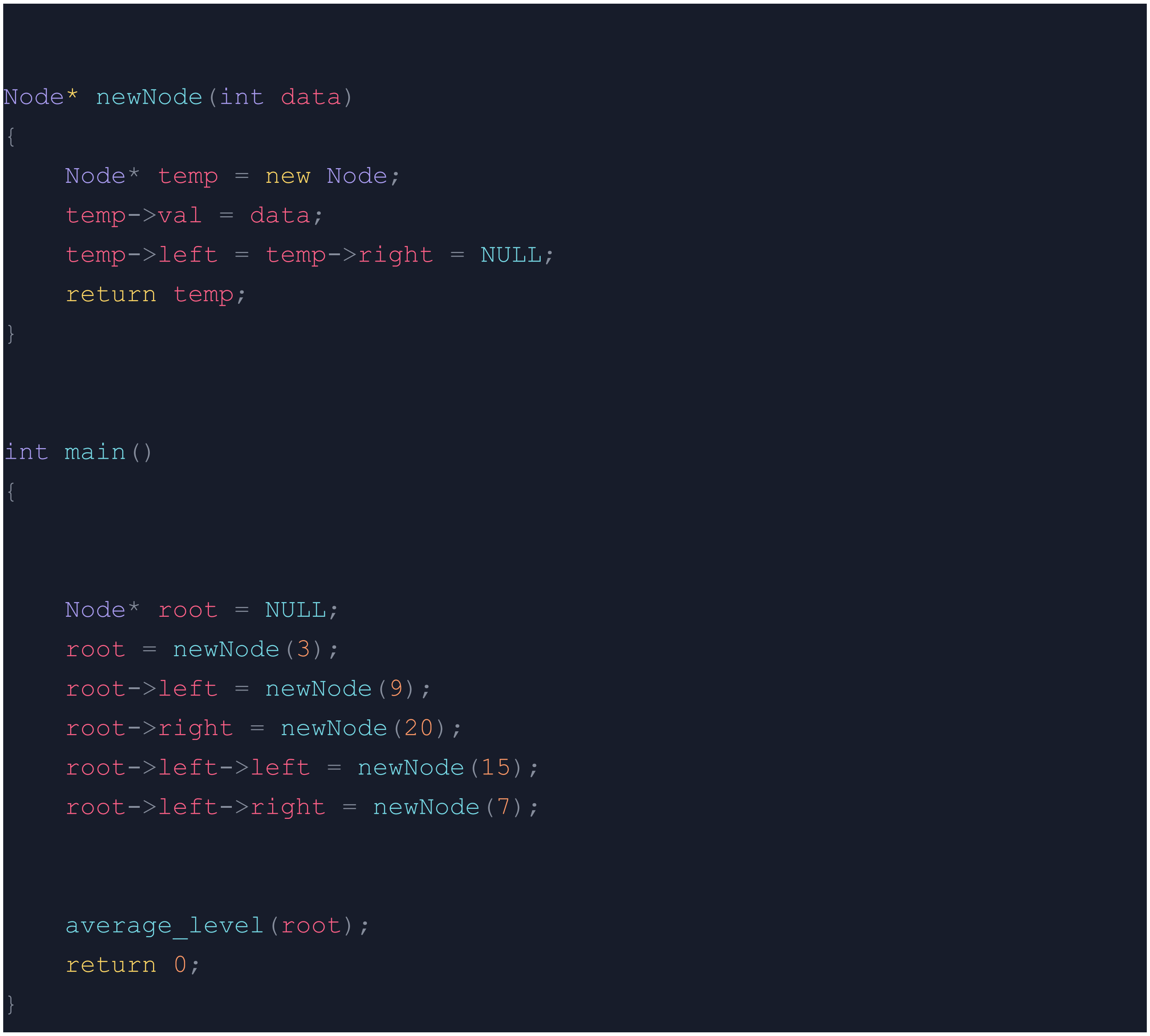


```
#include <bits/stdc++.h>
using namespace std;


struct Node {
    int val;
    struct Node *left, *right;
};


int traverse(Node* root, int* tilt)
{
    if (!root)
        return 0;


    int left = traverse(root->left, tilt);
    int right = traverse(root->right, tilt);


    *tilt += abs(left - right);

    return left + right + root->val;
}


int findTilt(Node* root)
{
    int tilt = 0;
    traverse(root, &tilt);
    return tilt;
}


Node* newNode(int data)
{
    Node* temp = new Node;
    temp->val = data;
    temp->left = temp->right = NULL;
    return temp;
}


int main()
{



    Node* root = NULL;
    root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);

    cout << findTilt(root);
    return 0;
}
```

Answer to the question No : 08

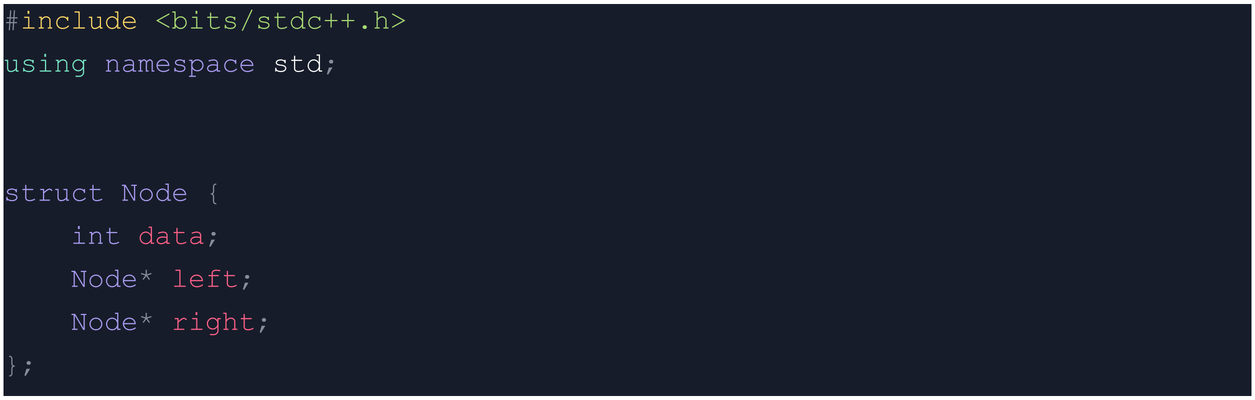


```
#include <bits/stdc++.h>
using namespace std;


struct Node {
    int val;
    struct Node* left, *right;
};


void average_level(Node* root)
{
    vector<float> res;


    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {


        int sum = 0, count = 0;
        queue<Node*> temp;
        while (!q.empty()) {
            Node* n = q.front();
            q.pop();
            sum += n->val;
            count++;
            if (n->left != NULL)
                temp.push(n->left);
            if (n->right != NULL)
                temp.push(n->right);
        }
        q = temp;
        cout << (sum * 1.0 / count) << " ";
    }
}
```



```
Node* newNode(int data)
{
    Node* temp = new Node;
    temp->val = data;
    temp->left = temp->right = NULL;
    return temp;
}


int main()
{



    Node* root = NULL;
    root = newNode(3);
    root->left = newNode(9);
    root->right = newNode(20);
    root->left->left = newNode(15);
    root->left->right = newNode(7);


    average_level(root);
    return 0;
}
```

Answer to the question No : 09

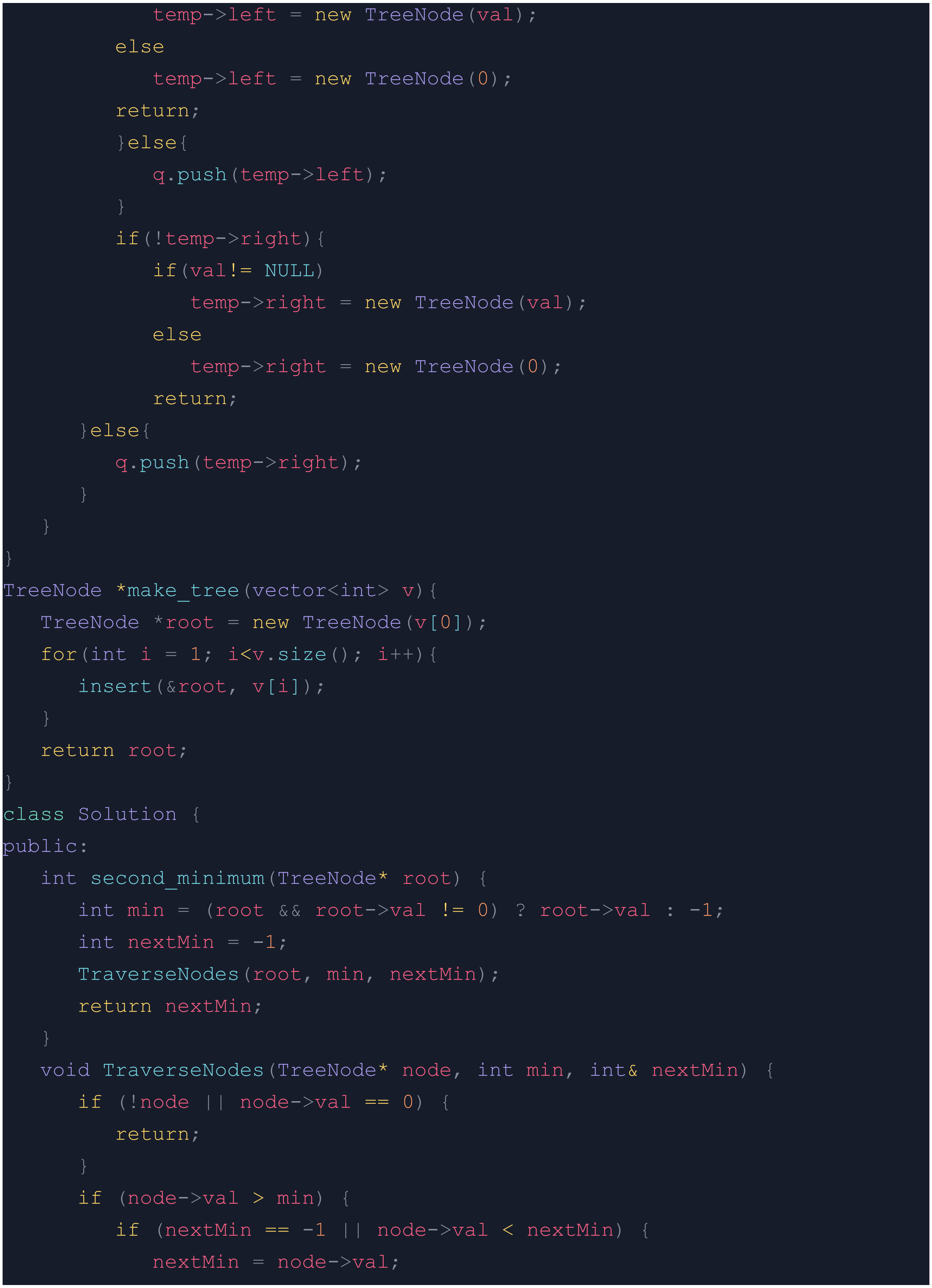


```
#include <bits/stdc++.h>
using namespace std;


struct Node {
    int data;
    Node* left;
    Node* right;
};
```

```
Node* newNode(int data)
{
    Node* temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return (temp);
}


bool is_unival(Node* root)
{


    if (!root) {
        return true;
    }


    if (root->left != NULL
        && root->data != root->left->data)
        return false;


    if (root->right != NULL
        && root->data != root->right->data)
        return false;


    return is_unival(root->left)
           && is_unival(root->right);
}

int main()
{


    Node* root = newNode(1);
    root->left = newNode(1);
    root->right = newNode(1);
    root->left->left = newNode(1);
    root->left->right = newNode(1);
    root->right->right = newNode(1);

    if (is_unival(root) == 1) {

        cout << "YES";
    }
    else {

        cout << "NO";
    }
    return 0;
}
```

Answer to the question No : 10

```
#include <bits/stdc++.h>
using namespace std;
class TreeNode{
   public:
      int val;
      TreeNode *left, *right;
      TreeNode(int data){
         val = data;
         left = NULL;
         right = NULL;
   }
};
void insert(TreeNode **root, int val){
   queue<TreeNode*> q;
   q.push(*root);
   while(q.size()){
      TreeNode *temp = q.front();
      q.pop();
      if(!temp->left){
         if(val != NULL)
```



```
            temp->left = new TreeNode(val);
         else
            temp->left = new TreeNode(0);
         return;
         }else{
            q.push(temp->left);
         }
         if(!temp->right){
            if(val!= NULL)
               temp->right = new TreeNode(val);
            else
               temp->right = new TreeNode(0);
            return;
      }else{
         q.push(temp->right);
      }
   }
}
TreeNode *make_tree(vector<int> v){
   TreeNode *root = new TreeNode(v[0]);
   for(int i = 1; i<v.size(); i++){
      insert(&root, v[i]);
   }
   return root;
}
class Solution {
public:
   int second_minimum(TreeNode* root) {
      int min = (root && root->val != 0) ? root->val : -1;
      int nextMin = -1;
      TraverseNodes(root, min, nextMin);
      return nextMin;
   }
   void TraverseNodes(TreeNode* node, int min, int& nextMin) {
      if (!node || node->val == 0) {
         return;
      }
      if (node->val > min) {
         if (nextMin == -1 || node->val < nextMin) {
            nextMin = node->val;
         }
      }
      TraverseNodes(node->left, min, nextMin);
      TraverseNodes(node->right, min, nextMin);
   }
};
main(){
   Solution ob;
   vector<int> v = {2,2,5,NULL,NULL,5,7};
   TreeNode *root = make_tree(v);
   cout << (ob.second_minimum(root));
}
```