```cpp
// A C++ program to print REVERSE level order traversal usin
// This approach is adopted from following link
// http://tech-queries.blogspot.in/2008/12/level-order-tree-
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left and right ch
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* Given a binary tree, print its nodes in reverse level ord
void reverseLevelOrder(node* root)
{
    stack <node *> S;
    queue <node *> Q;
    Q.push(root);

    // Do something like normal level order traversal order. Fol
    // differences with normal level order traversal
    // 1) Instead of printing a node, we push the node to stack
    // 2) Right subtree is visited before left subtree
    while (Q.empty() == false)
    {
        /* Dequeue node and make it root */
        root = Q.front();
        Q.pop();
        S.push(root);

        /* Enqueue right child */
        if (root->right)
            Q.push(root->right); // NOTE: RIGHT CHILD IS ENQUEUED E

        /* Enqueue left child */
        if (root->left)
            Q.push(root->left);
```

```
40:     }
41:
42:     // Now pop all items from stack one by one and print them
43:     while (S.empty() == false)
44:     {
45:         root = S.top();
46:         cout << root->data << " ";
47:         S.pop();
48:     }
49: }
50:
51: /* Helper function that allocates a new node with the
52: given data and NULL left and right pointers. */
53: node* newNode(int data)
54: {
55:     node* temp = new node;
56:     temp->data = data;
57:     temp->left = NULL;
58:     temp->right = NULL;
59:
60:     return (temp);
61: }
62:
63: /* Driver program to test above functions*/
64: int main()
65: {
66:     struct node *root = newNode(1);
67:     root->left   = newNode(2);
68:     root->right  = newNode(3);
69:     root->left->left = newNode(4);
70:     root->left->right = newNode(5);
71:     root->right->left = newNode(6);
72:     root->right->right = newNode(7);
73:
74:     cout << "Level Order traversal of binary tree is \n";
75:     reverseLevelOrder(root);
76:
77:     return 0;
78: }
```

79: