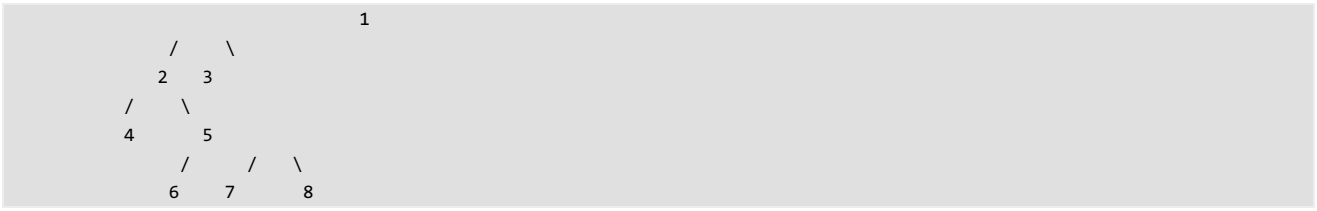
Convert left-right representation of a binary tree to down-right

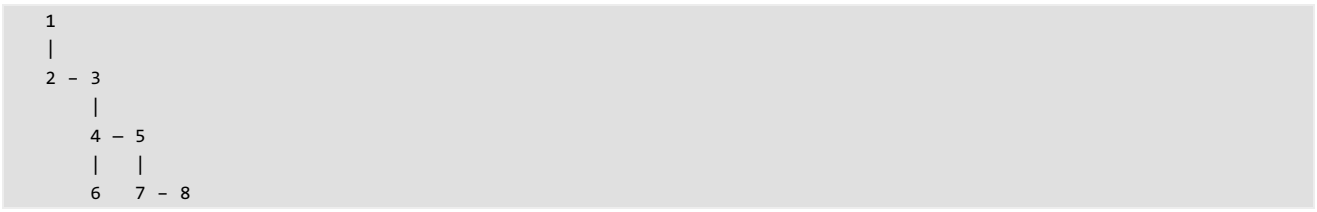
Left-Right representation of a binary tree is standard representation where every node has a pointer to left child and another pointer to right child.

Down-Right representation is an alternate representation where every node has a pointer to left (or first) child and another pointer to next sibling. So siblings at every level are connected from left to right.

Given a binary tree in left-right representation as below



Convert the structure of the tree to down-right representation like the below tree.



The conversion should happen in-place, i.e., left child pointer should be used as down pointer and right child pointer should be used as right sibling pointer.

We strongly recommend to minimize your browser and try this yourself.

The idea is to first convert left and right children, then convert the root. Following is C++ implementation of the idea.

```
/* C++ program to convert left-right to down-right
   representation of binary tree */
#include <iostream>
#include <queue>
using namespace std;

// A Binary Tree Node
struct node
{
    int key;
    struct node *left, *right;
};

// An Iterative level order traversal based function to
// convert left-right to down-right representation.
void convert(node *root)
{
    // Base Case
    if (root == NULL)  return;

    // Recursively convert left an right subtrees
    convert(root->left);
    convert(root->right);

    // If left child is NULL, make right child as left
    // as it is the first child.
    if (root->left == NULL)
       root->left = root->right;

    // If left child is NOT NULL, then make right child
    // as right of left child
    else
       root->left->right = root->right;
```

```
    // Set root's right as NULL
    root->right = NULL;
}

// A utility function to traverse a tree stored in
// down-right form.
void downRightTraversal(node *root)
{
    if (root != NULL)
    {
        cout << root->key << " ";
        downRightTraversal(root->right);
        downRightTraversal(root->left);
    }
}

// Utility function to create a new tree node
node* newNode(int key)
{
    node *temp = new node;
    temp->key = key;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree shown in above diagram
    /*
           1
         /   \
        2     3
             / \
            4   5
           /   /  \
          6   7    8
    */
    node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->right->left = newNode(4);
    root->right->right = newNode(5);
    root->right->left->left = newNode(6);
    root->right->right->left = newNode(7);
    root->right->right->right = newNode(8);

    convert(root);

    cout << "Traversal of the tree converted to down-right form\n";
    downRightTraversal(root);

    return 0;
}
```

Output:

```
Traversal of the tree converted to down-right form
1 2 3 4 5 7 8 6
```

Time complexity of the above program is O(n).