Populate Inorder Successor for all nodes

Given a Binary Tree where each node has following structure, write a function to populate next pointer for all nodes. The next pointer for every node should be set to point to inorder successor.

```
struct node
{
  int data;
  struct node* left;
  struct node* right;
  struct node* next;
}
```

Initially, all next pointers have NULL values. Your function should fill these next pointers so that they point to inorder successor.

Solution (Use Reverse Inorder Traversal)

Traverse the given tree in reverse inorder traversal and keep track of previously visited node. When a node is being visited, assign previously visited node as next.

C

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *left;
    struct node *right;
    struct node *next;
};

/* Set next of p and all descendents of p by traversing them in reverse Inorder */
void populateNext(struct node* p)
{
    // The first visited node will be the rightmost node
    // next of the rightmost node will be NULL
    static struct node *next = NULL;

    if (p)
    {
        // First set the next pointer in right subtree
        populateNext(p->right);

        // Set the next as previously visited node in reverse Inorder
        p->next = next;

        // Change the prev for subsequent node
        next = p;

        // Finally, set the next pointer in left subtree
        populateNext(p->left);
    }
}

/* UTILITY FUNCTIONS */
/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newnode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
```

```
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->next = NULL;

    return(node);
}

/* Driver program to test above functions*/
int main()
{

    /* Constructed binary tree is
              10
            /   \
          8      12
        /
      3
    */
    struct node *root = newnode(10);
    root->left        = newnode(8);
    root->right       = newnode(12);
    root->left->left  = newnode(3);

    // Populates nextRight pointer in all nodes
    populateNext(root);

    // Let us see the populated values
    struct node *ptr = root->left->left;
    while(ptr)
    {
        // -1 is printed if there is no successor
        printf("Next of %d is %d \n", ptr->data, ptr->next? ptr->next->data: -1);
        ptr = ptr->next;
    }

    return 0;
}
```

Java

```
// Java program to populate inorder traversal of all nodes

// A binary tree node
class Node
{
    int data;
    Node left, right, next;

    Node(int item)
    {
        data = item;
        left = right = next = null;
    }
}

class BinaryTree
{
    Node root;
    static Node next = null;

    /* Set next of p and all descendents of p by traversing them in
       reverse Inorder */
    void populateNext(Node node)
    {
        // The first visited node will be the rightmost node
        // next of the rightmost node will be NULL
        if (node != null)
        {
            // First set the next pointer in right subtree
            populateNext(node.right);
```

```
            populateNext(node.right);

            // Set the next as previously visited node in reverse Inorder
            node.next = next;

            // Change the prev for subsequent node
            next = node;

            // Finally, set the next pointer in left subtree
            populateNext(node.left);
        }
    }

    /* Driver program to test above functions*/
    public static void main(String args[])
    {
        /* Constructed binary tree is
            10
           /   \
          8      12
         /
        3    */
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(8);
        tree.root.right = new Node(12);
        tree.root.left.left = new Node(3);

        // Populates nextRight pointer in all nodes
        tree.populateNext(tree.root);

        // Let us see the populated values
        Node ptr = tree.root.left.left;
        while (ptr != null)
        {
            // -1 is printed if there is no successor
            int print = ptr.next != null ? ptr.next.data : -1;
            System.out.println("Next of " + ptr.data + " is: " + print);
            ptr = ptr.next;
        }
    }
}

// This code has been contributed by Mayank Jaiswal
```

We can avoid the use of static variable by passing reference to next as paramater.

C

```
// An implementation that doesn't use static variable

// A wrapper over populateNextRecur
void populateNext(struct node *root)
{
    // The first visited node will be the rightmost node
    // next of the rightmost node will be NULL
    struct node *next = NULL;

    populateNextRecur(root, &next);
}

/* Set next of all descendents of p by traversing them in reverse Inorder */
void populateNextRecur(struct node* p, struct node **next_ref)
{
    if (p)
    {
        // First set the next pointer in right subtree
        populateNextRecur(p->right, next_ref);

        // Set the next as previously visited node in reverse Inorder
        p->next = *next_ref;

        // Change the prev for subsequent node
        *next_ref = p;

        // Finally, set the next pointer in right subtree
        populateNextRecur(p->left, next_ref);
    }
}
```

Java

```
// A wrapper over populateNextRecur
    void populateNext(Node node) {

        // The first visited node will be the rightmost node
        // next of the rightmost node will be NULL
        populateNextRecur(node, next);
    }

    /* Set next of all descendents of p by traversing them in reverse Inorder */
    void populateNextRecur(Node p, Node next_ref) {
        if (p != null) {

           // First set the next pointer in right subtree
            populateNextRecur(p.right, next_ref);

            // Set the next as previously visited node in reverse Inorder
            p.next = next_ref;

            // Change the prev for subsequent node
            next_ref = p;

            // Finally, set the next pointer in right subtree
            populateNextRecur(p.left, next_ref);
        }
    }
```

Time Complexity: O(n)