Morris traversal for Preorder

Using Morris Traversal, we can traverse the tree without using stack and recursion. The algorithm for Preorder is almost similar to Morris traversal for Inorder.

1...**If** left child is null, print the current node data. Move to right child.
....**Else**, Make the right child of the inorder predecessor point to the current node. Two cases arise:
.........**a)** The right child of the inorder predecessor already points to the current node. Set right child to NULL. Move to right child of current node.
.........**b)** The right child is NULL. Set it to current node. Print current node's data and move to left child of current node.
2...Iterate until current node is not NULL.

Following is the implementation of the above algorithm.

C

```c
// C program for Morris Preorder traversal
#include <stdio.h>
#include <stdlib.h>

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

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* temp = (struct node*) malloc(sizeof(struct node));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Preorder traversal without recursion and without stack
void morrisTraversalPreorder(struct node* root)
{
    while (root)
    {
        // If left child is null, print the current node data. Move to
        // right child.
        if (root->left == NULL)
        {
            printf( "%d ", root->data );
            root = root->right;
        }
        else
        {
            // Find inorder predecessor
            struct node* current = root->left;
            while (current->right && current->right != root)
                current = current->right;

            // If the right child of inorder predecessor already points to
            // this node
            if (current->right == root)
            {
                current->right = NULL;
                root = root->right;
            }
            // If right child doesn't point to this node, then print this
            // node and make right child point to this node
            else
            {
                printf("%d ", root->data);
                current->right = root;
                root = root->left;
            }
        }
    }
}

// Function for sStandard preorder traversal
void preorder(struct node* root)
{
    if (root)
    {
        printf( "%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

/* Driver program to test above functions*/
int main()
{
    struct node* root = NULL;

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

    root->left->left = newNode(4);
    root->left->right = newNode(5);

    root->right->left = newNode(6);
    root->right->right = newNode(7);

    root->left->left->left = newNode(8);
    root->left->left->right = newNode(9);

    root->left->right->left = newNode(10);
    root->left->right->right = newNode(11);

    morrisTraversalPreorder(root);

    printf("\n");
    preorder(root);

    return 0;
}
```

Java

```
// Java program to implement Morris preorder traversal

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {
    Node root;

    void morrisTraversalPreorder()
    {
        morrisTraversalPreorder(root);
    }

    // Preorder traversal without recursion and without stack
    void morrisTraversalPreorder(Node node) {
        while (node != null) {

            // If left child is null, print the current node data. Move to
            // right child.
            if (node.left == null) {
                System.out.print(node.data + " ");
                node = node.right;
            } else {

                // Find inorder predecessor
                Node current = node.left;
                while (current.right != null && current.right != node) {
                    current = current.right;
                }

                // If the right child of inorder predecessor already points to
                // this node
                if (current.right == node) {
                    current.right = null;
                    node = node.right;
                }

                // If right child doesn't point to this node, then print this
                // node and make right child point to this node
                else {
                    System.out.print(node.data + " ");
                    current.right = node;
                    node = node.left;
                }
            }
        }
    }

    void preorder()
    {
        preorder(root);
    }

    // Function for Standard preorder traversal
    void preorder(Node node) {
        if (node != null) {
            System.out.print(node.data + " ");
            preorder(node.left);
            preorder(node.right);
        }
    }

    // Driver programs to test above functions
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);
        tree.root.left.left.left = new Node(8);
        tree.root.left.left.right = new Node(9);
        tree.root.left.right.left = new Node(10);
        tree.root.left.right.right = new Node(11);
        tree.morrisTraversalPreorder();
        System.out.println("");
        tree.preorder();

    }
}

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

Output:

```
1 2 4 8 9 5 10 11 3 6 7
1 2 4 8 9 5 10 11 3 6 7
```

Limitations:
Morris traversal modifies the tree during the process. It establishes the right links while moving down the tree and resets the right links while moving up the tree. So the algorithm cannot be applied if write operations are not allowed.