Iterative Preorder Traversal

Given a Binary Tree, write an iterative function to print Preorder traversal of the given binary tree.

Refer this for recursive preorder traversal of Binary Tree. To convert an inherently recursive procedures to iterative, we need an explicit stack. Following is a simple stack based iterative process to print Preorder traversal.
1) Create an empty stack *nodeStack* and push root node to stack.
2) Do following while *nodeStack* is not empty.
….**a)** Pop an item from stack and print it.
….**b)** Push right child of popped item to stack
….**c)** Push left child of popped item to stack

Right child is pushed before left child to make sure that left subtree is processed first.

C++

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

using namespace std;

/* A binary tree node has data, left child and right child */
struct node
{
    int data;
    struct node* left;
    struct node* 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* node = new struct node;
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return(node);
}

// An iterative process to print preorder traversal of Binary tree
void iterativePreorder(node *root)
{
    // Base Case
    if (root == NULL)
       return;

    // Create an empty stack and push root to it
    stack<node *> nodeStack;
    nodeStack.push(root);

    /* Pop all items one by one. Do following for every popped item
       a) print it
       b) push its right child
       c) push its left child
    Note that right child is pushed first so that left is processed first */
    while (nodeStack.empty() == false)
    {
        // Pop the top item from stack and print it
        struct node *node = nodeStack.top();
        printf ("%d ", node->data);
        nodeStack.pop();

        // Push right and left children of the popped node to stack
        if (node->right)
            nodeStack.push(node->right);
        if (node->left)
            nodeStack.push(node->left);
    }
}

// Driver program to test above functions
int main()
{
    /* Constructed binary tree is
            10
          /   \
        8      2
      /  \    /
    3     5  2
  */
  struct node *root = newNode(10);
  root->left        = newNode(8);
  root->right       = newNode(2);
  root->left->left  = newNode(3);
  root->left->right = newNode(5);
  root->right->left = newNode(2);
  iterativePreorder(root);
  return 0;
}
```

Java

```
// Java program to implement iterative preorder traversal
import java.util.Stack;

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    Node root;

    void iterativePreorder()
    {
        iterativePreorder(root);
    }

    // An iterative process to print preorder traversal of Binary tree
    void iterativePreorder(Node node) {

        // Base Case
        if (node == null) {
            return;
        }

        // Create an empty stack and push root to it
        Stack<Node> nodeStack = new Stack<Node>();
        nodeStack.push(root);

        /* Pop all items one by one. Do following for every popped item
         a) print it
         a) print it
         b) push its right child
         c) push its left child
         Note that right child is pushed first so that left is processed first */
        while (nodeStack.empty() == false) {

            // Pop the top item from stack and print it
            Node mynode = nodeStack.peek();
            System.out.print(mynode.data + " ");
            nodeStack.pop();

            // Push right and left children of the popped node to stack
            if (mynode.right != null) {
                nodeStack.push(mynode.right);
            }
            if (mynode.left != null) {
                nodeStack.push(mynode.left);
            }
        }
    }

    // driver program to test above functions
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(8);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(2);
        tree.iterativePreorder();

    }
}

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

Python

```
# Python program to perform iterative preorder traversal

# A binary tree node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# An iterative process to print preorder traveral of BT
def iterativePreorder(root):

    # Base CAse
    if root is None:
        return

    # create an empty stack and push root to it
    nodeStack = []
    nodeStack.append(root)

    #  Pop all items one by one. Do following for every popped item
    #   a) print it
    #   b) push its right child
    #   c) push its left child
    # Note that right child is pushed first so that left
    # is processed first */
    while(len(nodeStack) > 0):

        # Pop the top item from stack and print it
        node = nodeStack.pop()
        print node.data,

        # Push right and left children of the popped node
        # to stack
        if node.right is not None:
            nodeStack.append(node.right)
        if node.left is not None:
            nodeStack.append(node.left)

# Driver program to test above function
root = Node(10)
root.left = Node(8)
root.right = Node(2)
root.left.left = Node(3)
root.left.right = Node(5)
root.right.left = Node(2)
iterativePreorder(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
10 8 3 5 2 2
```