```cpp
#include <iostream>
#include <string>
#include <utility>
using namespace std;

// Threaded binary tree node
struct Node
{
    int data;
    Node *left, *right;

    // true if the right pointer of a node points to its inorder successor
    bool isThreaded = false;
};

// Function to create a new binary tree node having a given key
Node* newNode(int key)
{
    Node* node = new Node;
    node->data = key;
    node->left = node->right = nullptr;

    return node;
}

// Utility function to return the leftmost node in a given binary tree
Node *leftMostNode(Node* root)
{
    Node* node = root;
    while (node && node->left) {
        node = node->left;
    }

    return node;
}

// Iterative function to perform inorder traversal on a threaded binary tree
void traverse(Node* root)
{
    // base case
    if (root == nullptr) {
        return;
    }

    // start from the leftmost node
    Node* curr = leftMostNode(root);
    while (curr)
    {
        // print the current node
        cout << curr->data << " ";

        // go to the inorder successor if the current node is threaded
        if (curr->isThreaded) {
            curr = curr->right;
        }
        // otherwise, visit the leftmost child in the right subtree
        else {
```

```
            curr = leftMostNode(curr->right);
        }
    }
}

// Function to convert a binary tree into a threaded binary tree
// using inorder traversal
void populateNext(Node* curr, Node* &prev)
{
    // base case: empty tree
    if (curr == nullptr) {
        return;
    }

    // recur for the left subtree
    populateNext(curr->left, prev);

    // if the current node is not the root node of a binary tree
    // and has a null right child
    if (prev && !prev->right)
    {
        // set the right child of the previous node to point to the current node
        prev->right = curr;

        // set thread flag to true
        prev->isThreaded = true;
    }

    // update previous node
    prev = curr;

    // recur for the right subtree
    populateNext(curr->right, prev);
}

// Convert a binary tree into a threaded binary tree
void convertToThreaded(Node* root)
{
    // stores previously visited node
    Node* prev = nullptr;
    populateNext(root, prev);
}

int main()
{
    /* Construct the following tree
              5
            /   \
           /     \
          2       7
         / \     / \
        /   \   /   \
        1    4 6     9
            /       / \
           /       /   \
          3       8    10
    */
```

```
    Node* root = newNode(5);
    root->left = newNode(2);
    root->right = newNode(7);
    root->left->left = newNode(1);
    root->left->right = newNode(4);
    root->right->left = newNode(6);
    root->right->right = newNode(9);
    root->left->right->left = newNode(3);
    root->right->right->left = newNode(8);
    root->right->right->right = newNode(10);

    convertToThreaded(root);
    traverse(root);

    return 0;
}
```