Threaded Binary Tree

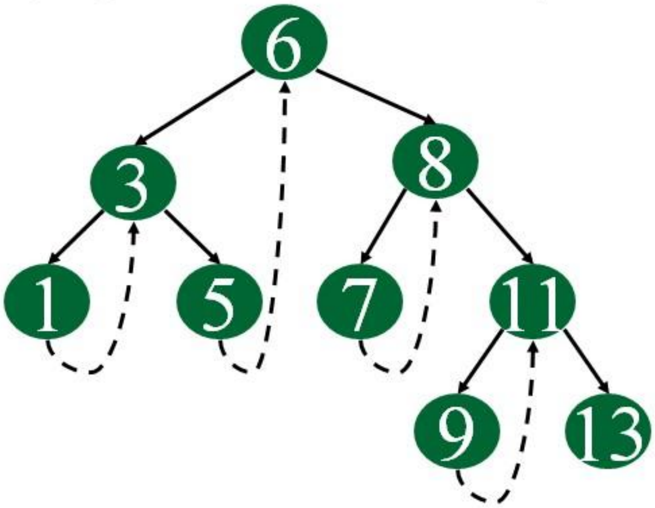
The *Inorder traversal of a Binary tree* can either be done using recursion or with the use of an auxiliary stack. **Threaded Binary Trees** are used to make the inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be *NULL* point to the inorder successor of the node (if it exists).

There are two types of threaded binary trees:

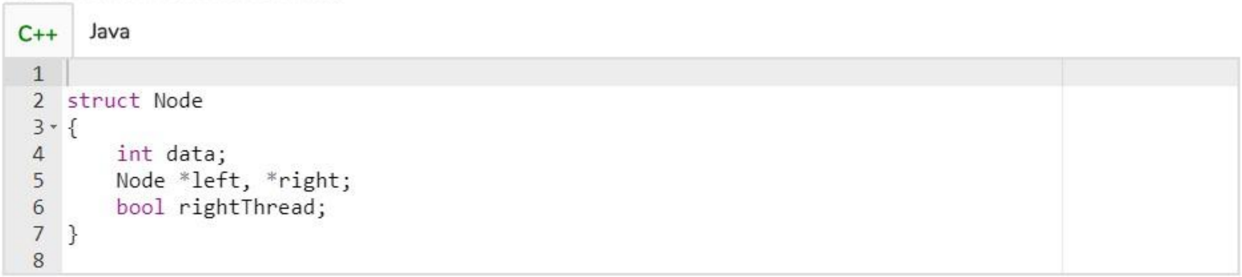
1. ***Single Threaded:*** Where a NULL right pointers is made to point to the inorder successor (if successor exists).
2. ***Double Threaded:*** Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.

Note: The threads are also useful for fast accessing ancestors of a node.

Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



Representation of a Threaded Node:



```

struct Node
{
    int data;
    Node *left, *right;
    bool rightThread;
}

```

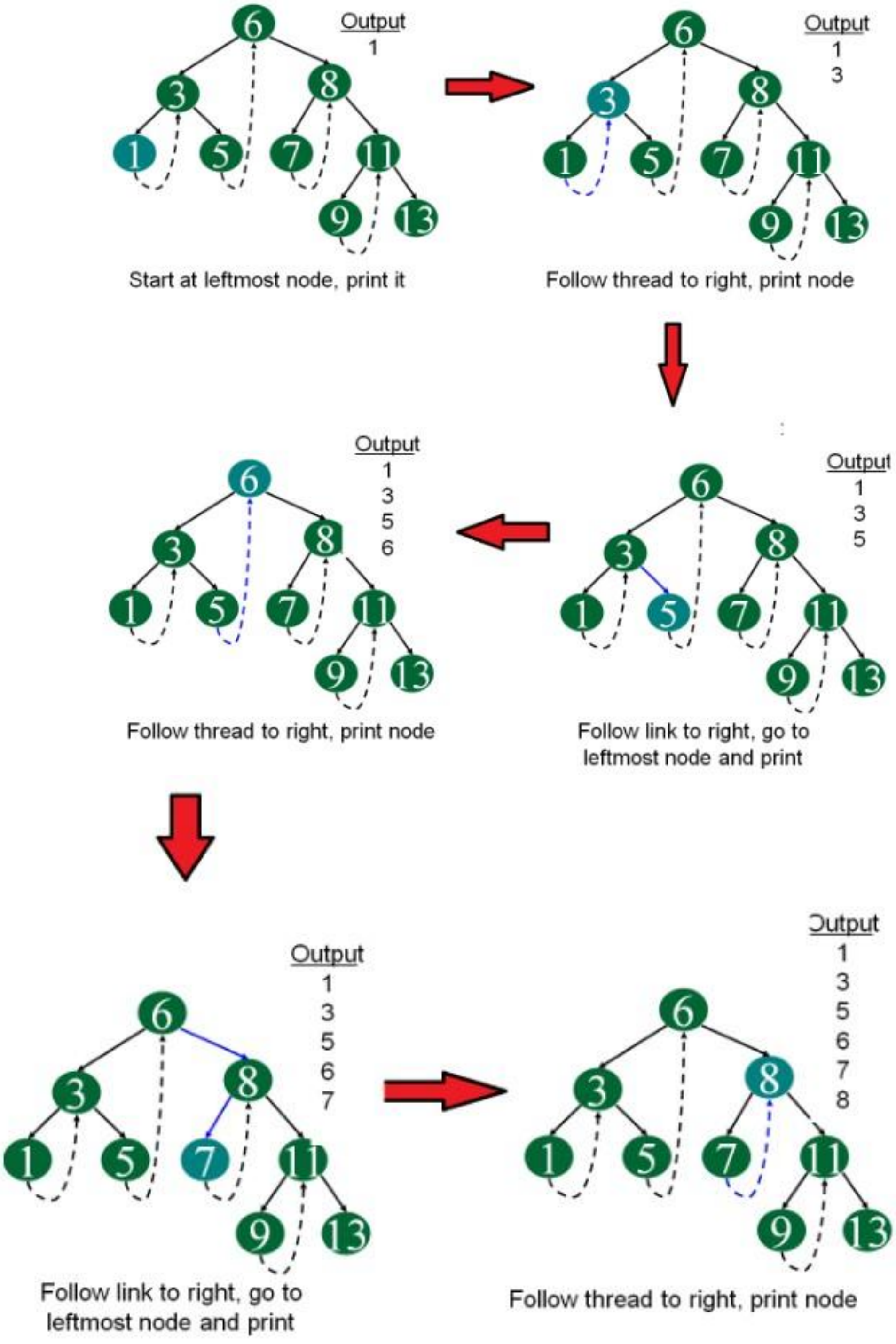
Since the right pointer in a Threaded Binary Tree is used for two purposes, the boolean variable *rightThread* is used to indicate whether the right pointer points to a right child or an inorder successor. Similarly, we can add leftThread for a double threaded binary tree.

Inorder Traversal in a Threaded Binary Tree: Below is the algorithm to perform inorder traversal in a Threaded Binary Tree using threads:

1. Start from the root node, go to the leftmost node and print the node.
2. Check if there is a thread towards the right for the current node.
    - If Yes, then follow the thread to the node and print the data of node linked with this thread.
    - Otherwise follow the link to the right subtree, find the leftmost node in the right subtree and print the leftmost node.

   Repeat the above process until the complete tree is traversed.

Following diagram demonstrates the inorder traversal in a Threaded Binary Tree:



Start at leftmost node, print it

Follow thread to right, print node

Follow link to right, go to leftmost node and print

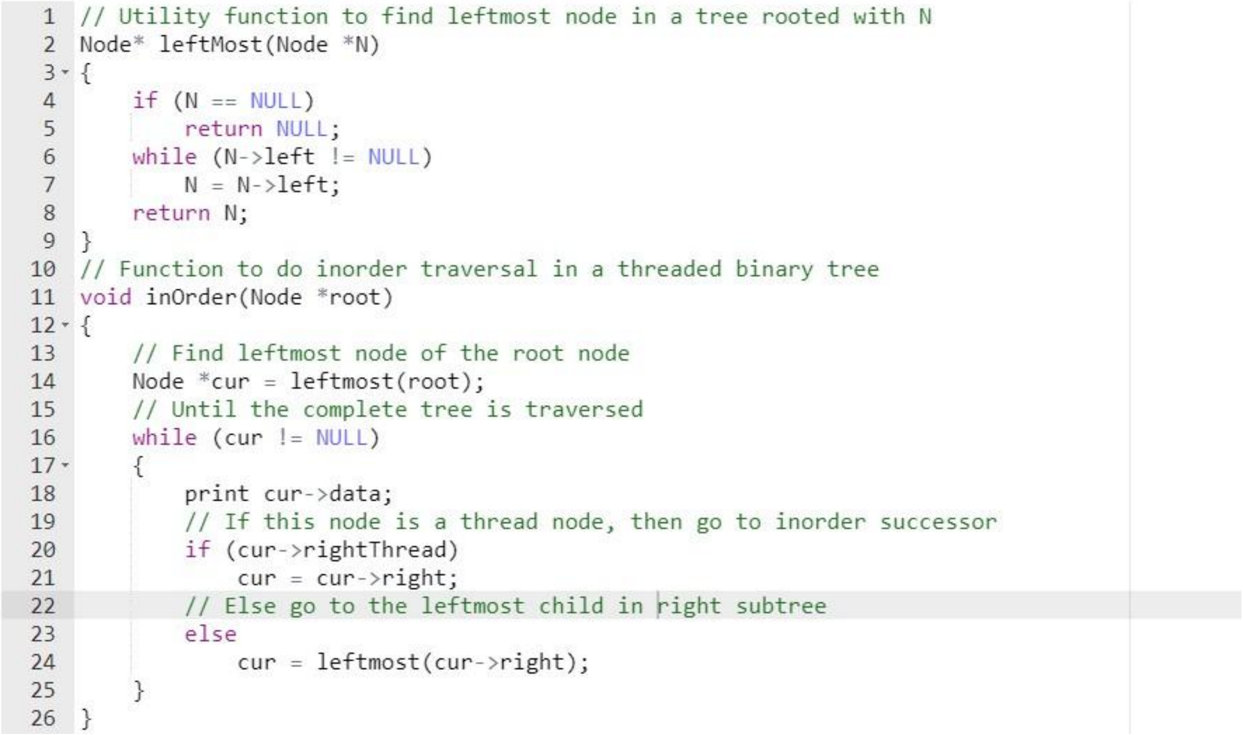
Follow thread to right, print node

Follow link to right, go to leftmost node and print

Follow thread to right, print node

continue same way for remaining node.....

Below functions implements the inorder traversal in a threaded binary tree:



```
// Utility function to find leftmost node in a tree rooted with N
Node* leftMost(Node *N)
{
    if (N == NULL)
        return NULL;
    while (N->left != NULL)
        N = N->left;
    return N;
}
// Function to do inorder traversal in a threaded binary tree
void inOrder(Node *root)
{
    // Find leftmost node of the root node
    Node *cur = leftmost(root);
    // Until the complete tree is traversed
    while (cur != NULL)
    {
        print cur->data;
        // If this node is a thread node, then go to inorder successor
        if (cur->rightThread)
            cur = cur->right;
        // Else go to the leftmost child in right subtree
        else
            cur = leftmost(cur->right);
    }
}
```