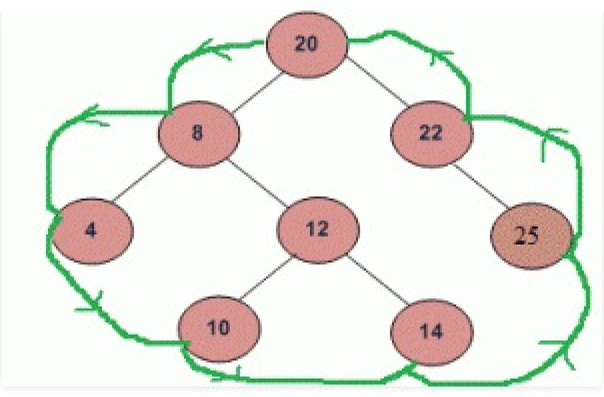
Boundary Traversal of binary tree

Given a binary tree, print boundary nodes of the binary tree Anti-Clockwise starting from the root. For example, boundary traversal of the following tree is "20 8 4 10 14 25 22"



We strongly recommend that you click here and practice it, before moving on to the solution.

We break the problem in 3 parts:
1. Print the left boundary in top-down manner.
2. Print all leaf nodes from left to right, which can again be sub-divided into two sub-parts:
.....**2.1** Print all leaf nodes of left sub-tree from left to right.
.....**2.2** Print all leaf nodes of right subtree from left to right.
3. Print the right boundary in bottom-up manner.

We need to take care of one thing that nodes are not printed again. e.g. The left most node is also the leaf node of the tree.

Based on the above cases, below is the implementation:

C++ Java Python

```
/* program for boundary traversal of a binary tree */
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node *left, *right;
};

// A simple function to print leaf nodes of a binary tree
void printLeaves(struct node* root)
{
    if ( root )
    {
        printLeaves(root->left);

        // Print it if it is a leaf node
        if ( !(root->left)  &&  !(root->right) )
            printf("%d ", root->data);

        printLeaves(root->right);
    }
}

// A function to print all left boundry nodes, except a leaf node.
// Print the nodes in TOP DOWN manner
void printBoundaryLeft(struct node* root)
{
    if (root)
    {
        if (root->left)
        {
            // to ensure top down order, print the node
            // before calling itself for left subtree
            printf("%d ", root->data);
            printBoundaryLeft(root->left);
        }
        else if( root->right )
        {
            printf("%d ", root->data);
            printBoundaryLeft(root->right);
        }
        // do nothing if it is a leaf node, this way we avoid
        // duplicates in output
    }
}

// A function to print all right boundry nodes, except a leaf node
// Print the nodes in BOTTOM UP manner
void printBoundaryRight(struct node* root)
{
    if (root)
    {
        if ( root->right )
        {
            // to ensure bottom up order, first call for right
            //  subtree, then print this node
            printBoundaryRight(root->right);
            printf("%d ", root->data);
        }
        else if ( root->left )
        {
            printBoundaryRight(root->left);
            printf("%d ", root->data);
        }
       // do nothing if it is a leaf node, this way we avoid
       // duplicates in output
    }
}


// A function to do boundary traversal of a given binary tree
void printBoundary (struct node* root)
{
    if (root)
    {
        printf("%d ",root->data);

        // Print the left boundary in top-down manner.
        printBoundaryLeft(root->left);

        // Print all leaf nodes
        printLeaves(root->left);
        printLeaves(root->right);

        // Print the right boundary in bottom-up manner
        printBoundaryRight(root->right);
    }
}

// A utility function to create a node
struct node* newNode( int data )
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root        = newNode(20);
    root->left               = newNode(8);
    root->left->left         = newNode(4);
    root->left->right        = newNode(12);
    root->left->right->left  = newNode(10);
    root->left->right->right = newNode(14);
    root->right              = newNode(22);
    root->right->right       = newNode(25);

    printBoundary( root );

    return 0;
}
```

Run on IDE

Output:

```
20 8 4 10 14 25 22
```

Time Complexity: O(n) where n is the number of nodes in binary tree.