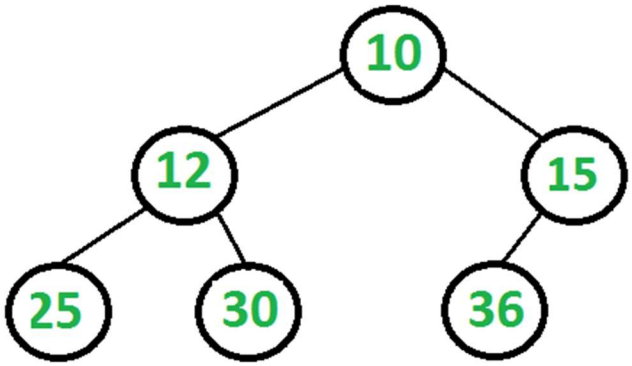
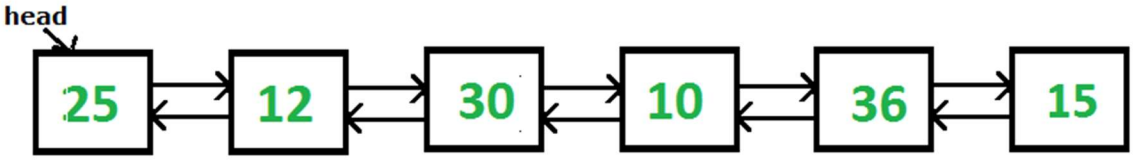
Convert Binary Tree to Doubly Linked List by keeping track of visited node

Given a Binary Tree, The task is to convert it to a **Doubly Linked List** keeping the same order.

- The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL.
- The order of nodes in DLL must be the same as in I**norder for the given Binary Tree**.
- The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.



The above tree should be in-place converted to following Doubly Linked List(DLL).



*The idea is to do in-order traversal of the binary tree. While doing inorder traversal, keep track of the previously visited node in a variable, say **prev**. For every visited node, make it next to the **prev** and set previous of this node as **prev**.*

Below is the implementation of the above approach:

C++

```
// A C++ program for in-place conversion of Binary Tree to
```

```
// DLL
#include <iostream>
using namespace std;

/* A binary tree node has data, and left and right pointers
 */
struct node {
    int data;
    node* left;
    node* right;
};

// A simple recursive function to convert a given Binary
// tree to Doubly Linked List root --> Root of Binary Tree
// head --> Pointer to head node of created doubly linked
// list
void BinaryTree2DoubleLinkedList(node* root, node** head)
{
    // Base case
    if (root == NULL)
        return;

    // Initialize previously visited node as NULL. This is
    // static so that the same value is accessible in all
    // recursive calls
    static node* prev = NULL;
```

```
    // Recursively convert left subtree
    BinaryTree2DoubleLinkedList(root->left, head);

    // Now convert this node
    if (prev == NULL)
        *head = root;
    else {
        root->left = prev;
        prev->right = root;
    }
    prev = root;

    // Finally convert right subtree
    BinaryTree2DoubleLinkedList(root->right, head);
}

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

/* Function to print nodes in a given doubly linked list */
```

```
void printList(node* node)
{
    while (node != NULL) {
        cout << node->data << " ";
        node = node->right;
    }
}

// Driver Code
int main()
{
    // Let us create the tree shown in above diagram
    node* root = newNode(10);
    root->left = newNode(12);
    root->right = newNode(15);
    root->left->left = newNode(25);
    root->left->right = newNode(30);
    root->right->left = newNode(36);

    // Convert to DLL
    node* head = NULL;
    BinaryTree2DoubleLinkedList(root, &head);

    // Print the converted list
    printList(head);

    return 0;
```

```
}
```

Output

```
25 12 30 10 36 15
```

Note: The use of static variables like above is not a recommended practice, here static is used for simplicity. Imagine if the same function is called for two or more trees. The old value of **prev** would be used in the next call for a different tree. To avoid such problems, we can use a double-pointer or a reference to a pointer.

Time Complexity: O(N), The above program does a simple inorder traversal, so time complexity is O(N) where N is the number of nodes in a given Binary tree.

Auxiliary Space: O(N), For recursion call stack.

Convert a given Binary Tree to Doubly Linked List iteratively using **Stack data structure**:

*Do iterative inorder traversal and maintain a **prev** pointer to point the last visited node then point **current node's perv** to prev and **prev's next** to current node.*

```
// A C++ program for in-place conversion of Binary Tree to

// DLL

#include <bits/stdc++.h>

using namespace std;


/* A binary tree node has data, and left and right pointers

*/

struct node {

        int data;

        node* left;

        node* right;

};
```

```
node * bToDLL(node *root)
{
	stack<pair<node*, int>> s;
	s.push({root, 0});
	vector<int> res;
	bool flag = true;
	node* head = NULL;
	node* prev = NULL;
	while(!s.empty()) {
		auto x = s.top();
		node* t = x.first;
		int state = x.second;
		s.pop();
		if(state == 3 or t == NULL) continue;
		s.push({t, state+1});
		if(state == 0) s.push({t->left, 0});
		else if(state == 1) {
			if(prev) prev->right = t;
			t->left = prev;
			prev = t;
			if(flag) {
				head = t;
				flag = false;
			}
		}
		else if(state == 2) s.push({t->right, 0});
	}
	return head;
}
```

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

/* Function to print nodes in a given doubly linked list */
void printList(node* node)
{
	while (node != NULL) {
		cout << node->data << " ";
		node = node->right;
	}
}

// Driver Code
int main()
{
	// Let us create the tree shown in above diagram
	node* root = newNode(10);
	root->left = newNode(12);
	root->right = newNode(15);
	root->left->left = newNode(25);
	root->left->right = newNode(30);
```

```
    root->right->left = newNode(36);

    // Convert to DLL
    node* head = bToDLL(root);

    // Print the converted list
    printList(head);

    return 0;
}
```

Output

```
25 12 30 10 36 15
```

Time complexity: O(N)
Auxiliary Space: O(N)