Data Structure & Algorithms

Lec 10: Introduction to Trees & Graphs

Fall 2017 - University of Windsor
Dr. Sherif Saad

Agenda

1. Binary Trees Implementation
2. Traversing Binary Tree
3. In-Class Activities

Binary Tree as ADT

A Binary Tree ADT is either empty, or it consists of a node called the root together with two binary trees called the left subtree and the right subtree of the root, in addition to the following operations.

1. Inserting an element into a tree
2. Deleting an element from a tree
3. Search for an element
4. Traversing the tree (Preorder, Inorder, Postorder)
5. Finding the size of the tree

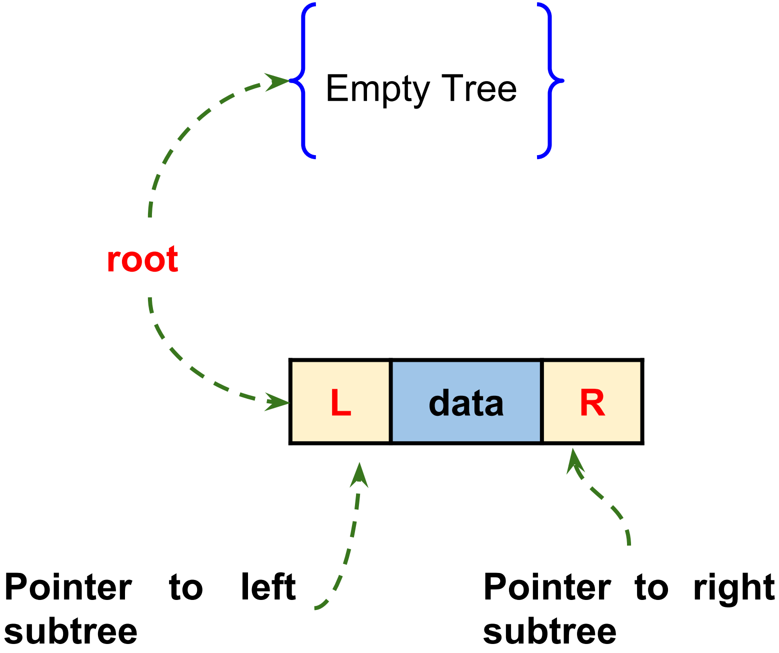
Binary Tree as ADT

A binary tree is either empty, or it consists of a node called the root.

Each node contains store data and two pointers. One points to the left subtree and the other points to the right subtree.

What is the left subtree?

What is the right subtree?



Traversing a Binary Tree

In-Order Traversal: the left subtree is visited first, then the root and later the right subtree.

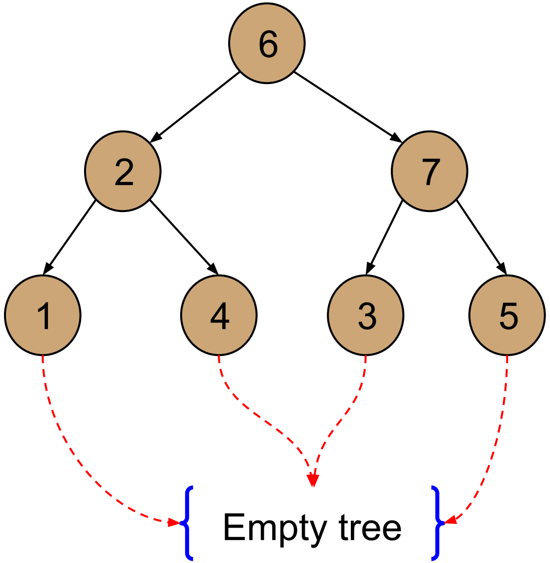
1, 2, 4, 6, 3, 7, 5

Pre-Order Traversal: the root node is visited first, then the left subtree and the right subtree

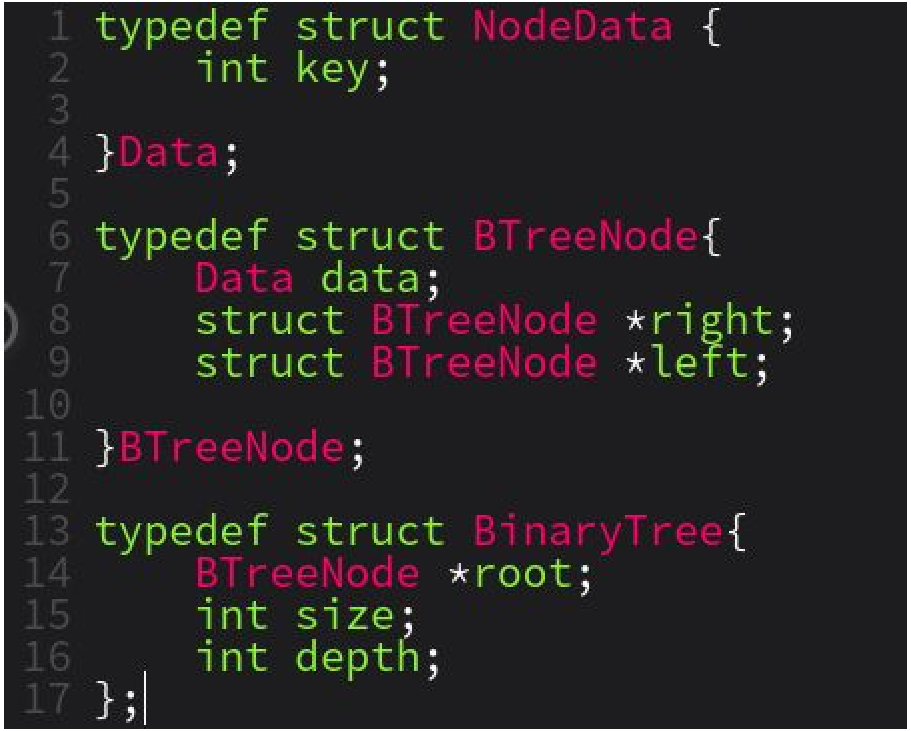
6, 2, 1, 4, 7, 3, 5

Post-Order Traversal: the left subtree is visited first, then the right subtree and finally the root

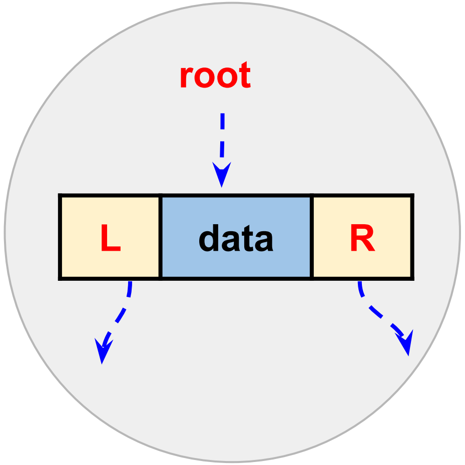
1, 4, 2, 3, 5, 7, 6



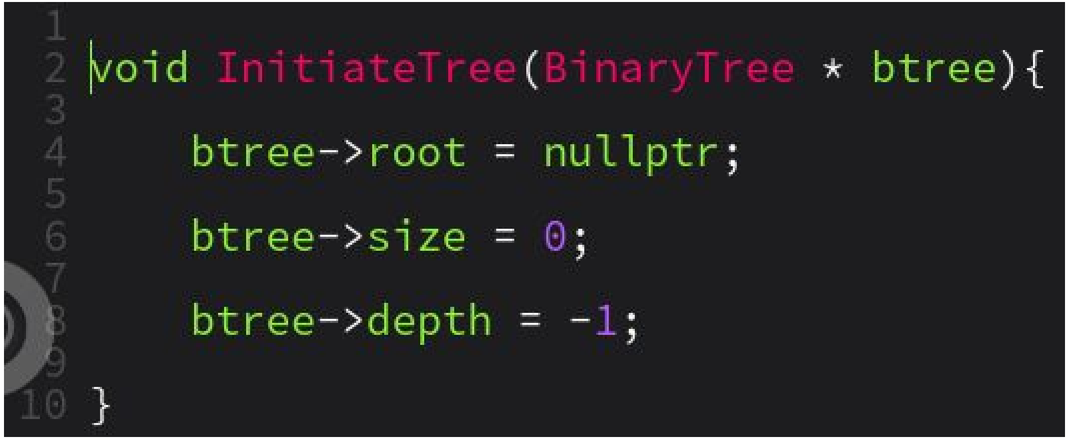
Binary Tree Design & Implementation



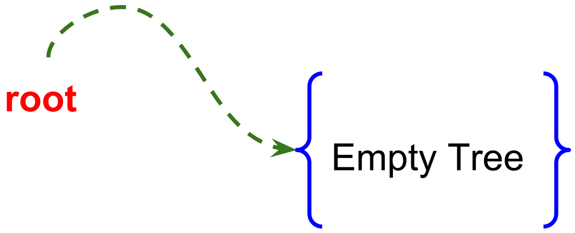
```
typedef struct NodeData {
    int key;

}Data;

typedef struct BTreeNode{
    Data data;
    struct BTreeNode *right;
    struct BTreeNode *left;

}BTreeNode;

typedef struct BinaryTree{
    BTreeNode *root;
    int size;
    int depth;
};
```



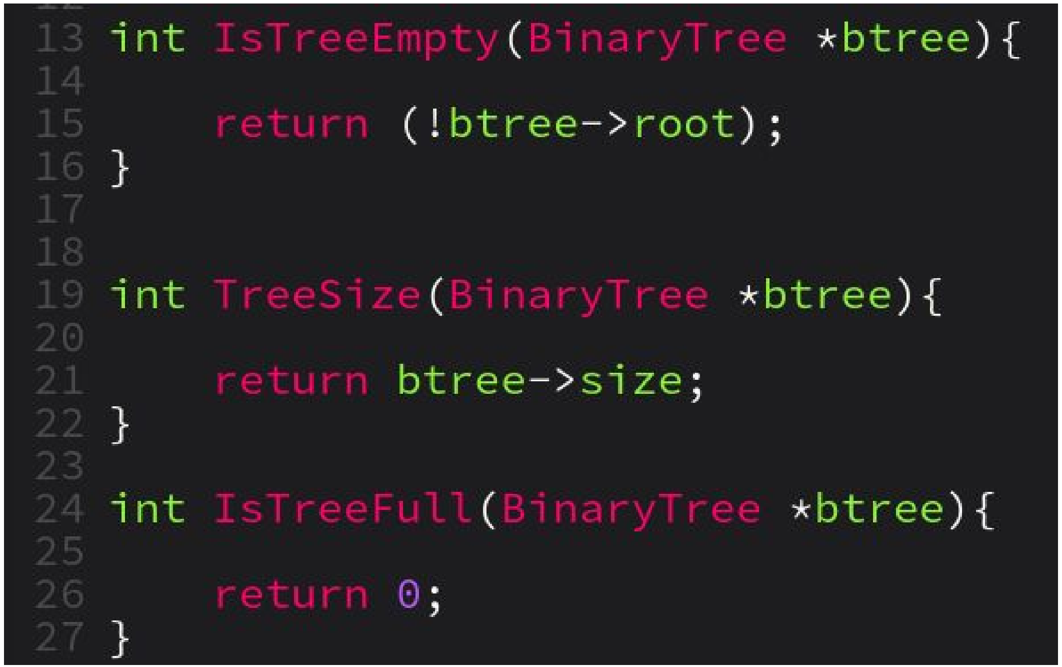
Create a Binary Tree



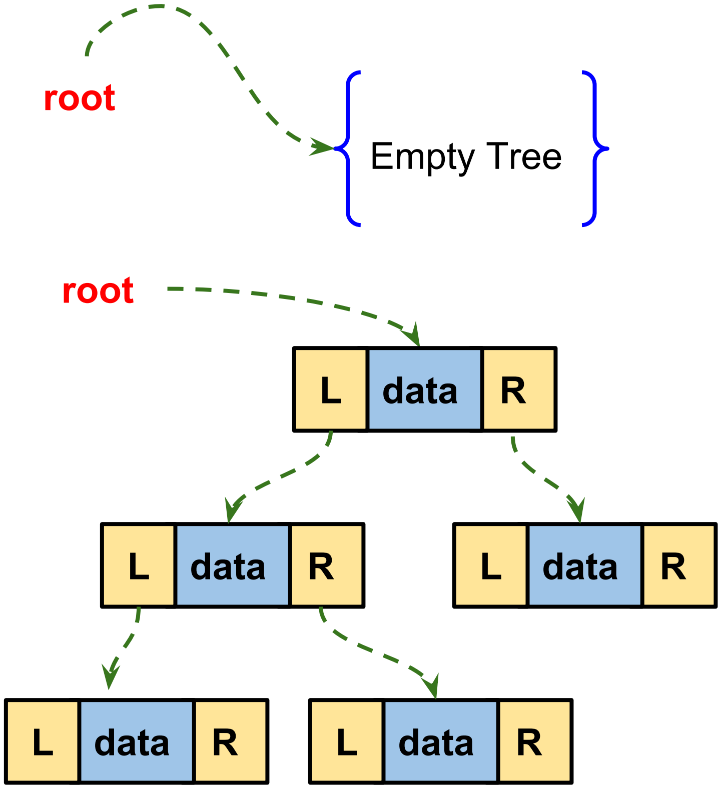
```
void InitiateTree(BinaryTree * btree){

    btree->root = nullptr;

    btree->size = 0;

    btree->depth = -1;

}
```



Binary Tree Implementation



```
int IsTreeEmpty(BinaryTree *btree){
    return (!btree->root);
}


int TreeSize(BinaryTree *btree){
    return btree->size;
}

int IsTreeFull(BinaryTree *btree){
    return 0;
}
```



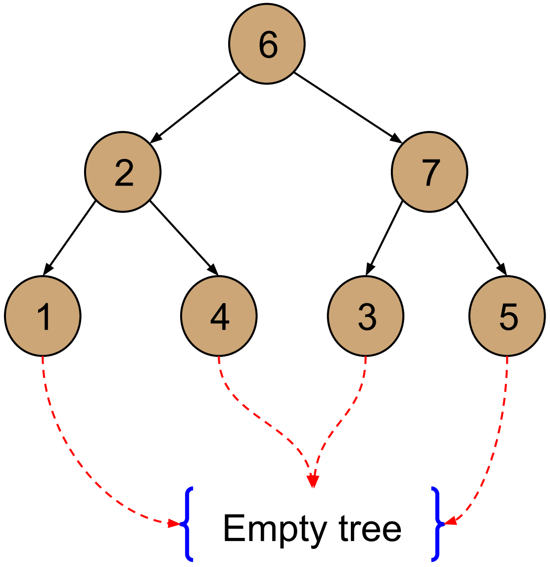
Binary Tree Inorder Traversal

The left subtree is visited first, then the root and later the right subtree.

We can implement tree traversal using recursion. We can also implement it iteratively

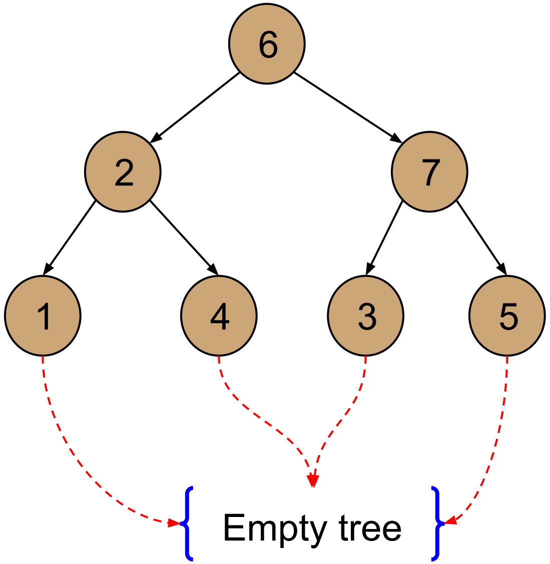
How could we do that using non-recursive approach?

We need to remember the current node so after we complete the left subtree we can go the right subtree

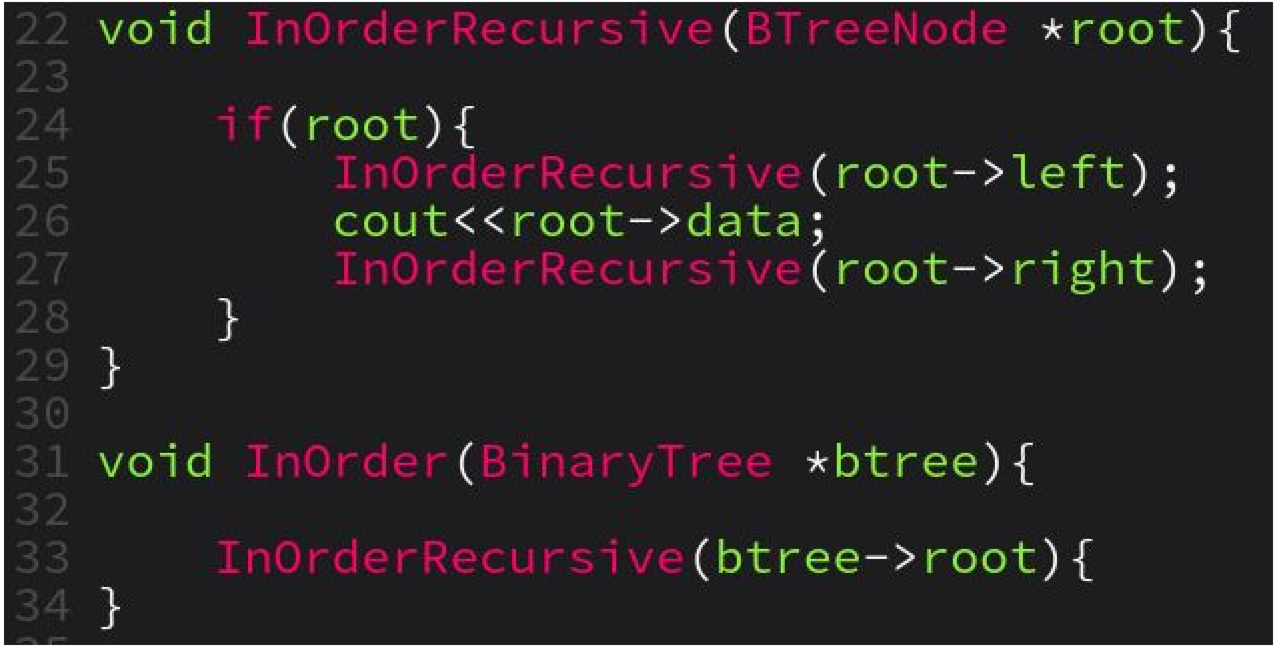


Binary Tree Inorder Traversal - Recursive

```
While all nodes are not visited:
    Recursively traverse the left subtree
    Visit root node
    Recursively traverse the right subtree
```



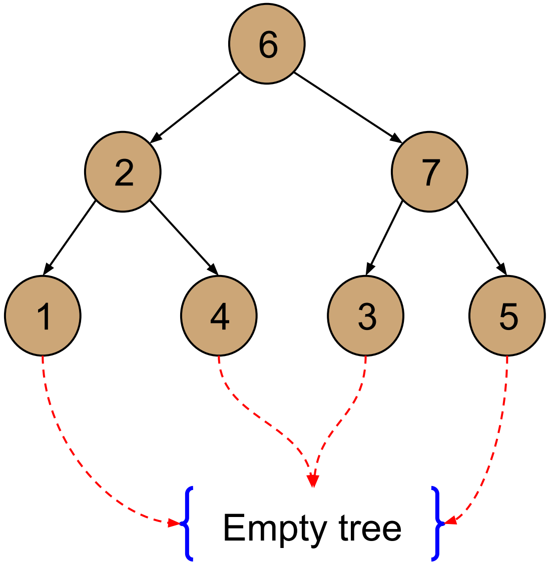
Binary Tree Inorder Traversal - Recursive



```
void InOrderRecursive(BTreeNode *root){

    if(root){
        InOrderRecursive(root->left);
        cout<<root->data;
        InOrderRecursive(root->right);
    }
}

void InOrder(BinaryTree *btree){

    InOrderRecursive(btree->root){
}
```

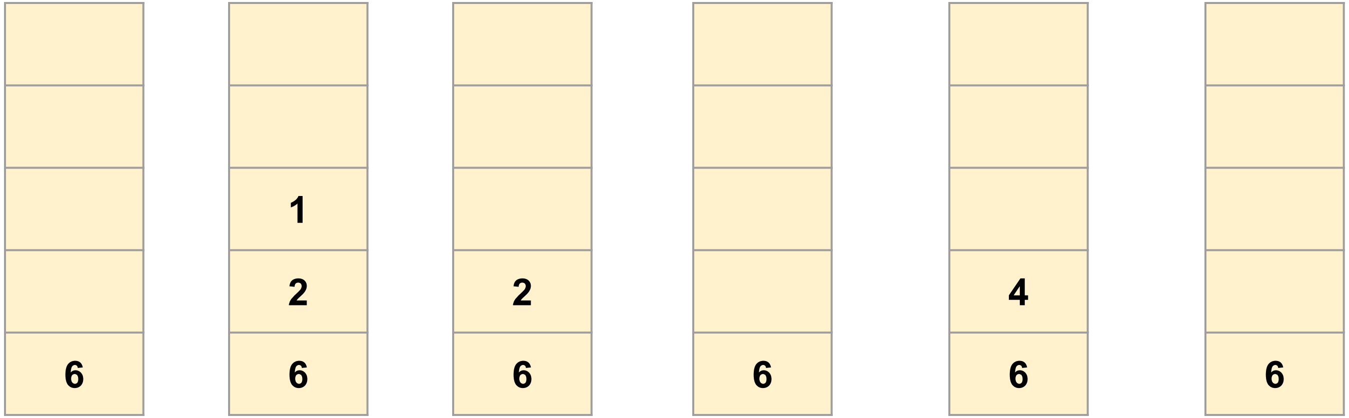
Binary Tree Inorder Traversal - Iterative

- A stack is required to keep track of the currently visited nodes.
- For each node, we push the node and its entire left subtree into the stack.
- Then we pop from the stack one node at a time, process the node and visit its right subtree.
- We only push none null nodes to the stack



Binary Tree Inorder Traversal - Iterative

1, 2, 4, 6, 3, 7, 5

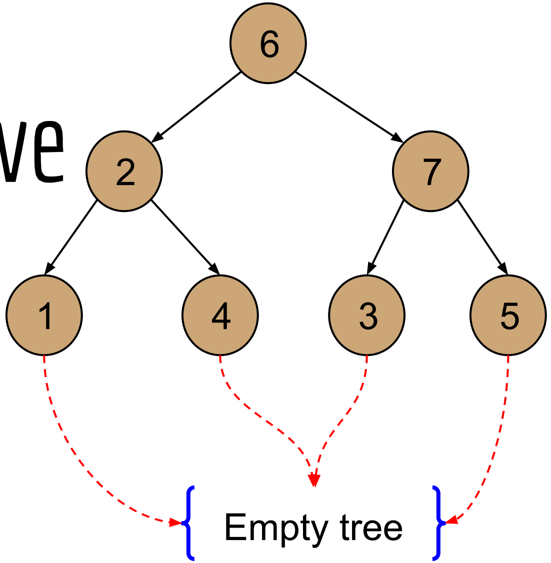


pop(1)
Print "1"
Switch to 1 right subtree.

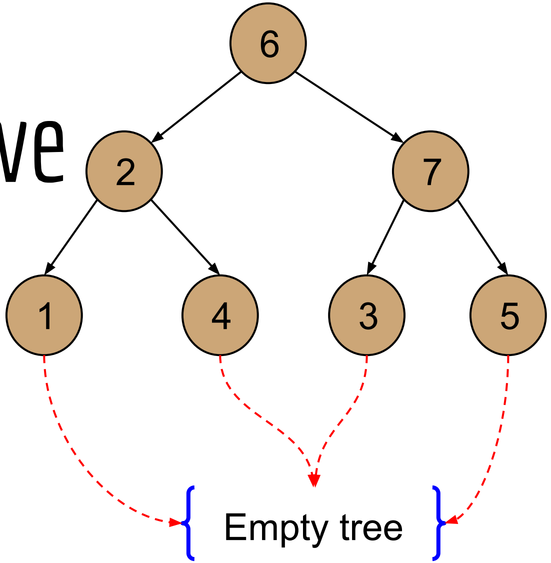
pop(2)
Print "2"
Switch to 2 right subtree.

push(4) and its left subtree

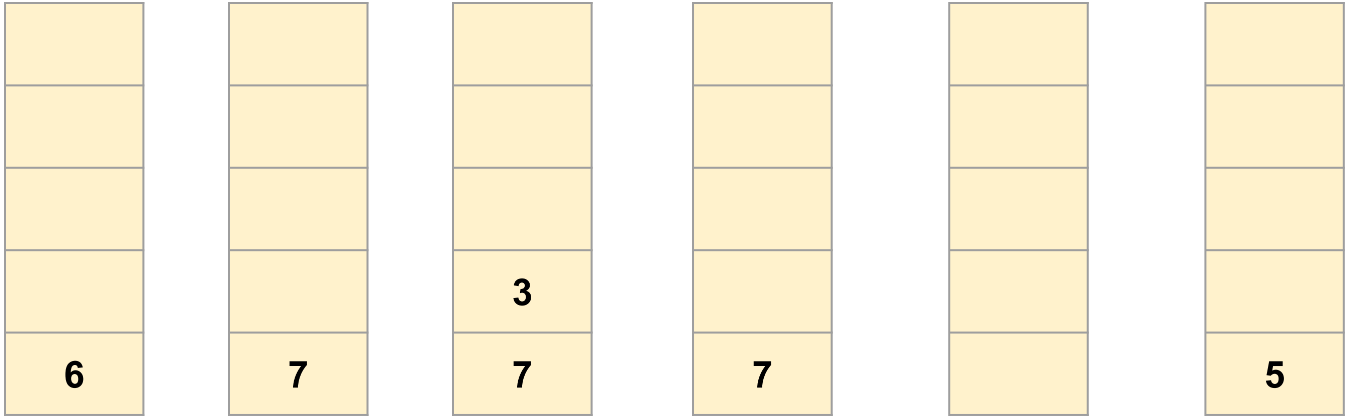
pop(4)
Print "4"
Switch to 4 right subtree.



Binary Tree Inorder Traversal - Iterative



1, 2, 4, 6, 3, 7, 5



pop(6)
Print "6"
Switch to 6 right subtree.

push(7) and its left subtree

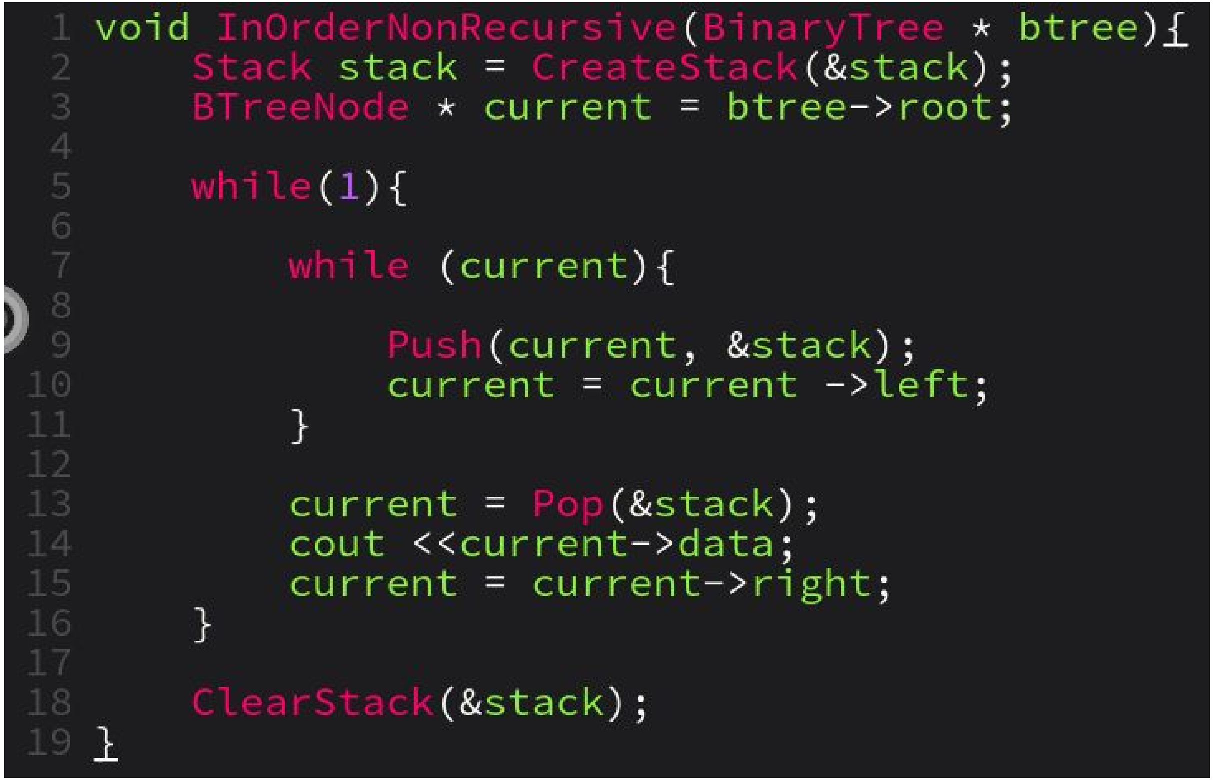
push(3) and its left subtree

pop(3)
Print "3"
Switch to 3 right subtree.

pop(7)
Print "7"
Switch to right subtree.

push(5) and its left subtree

Binary Tree Inorder Traversal - Iterative



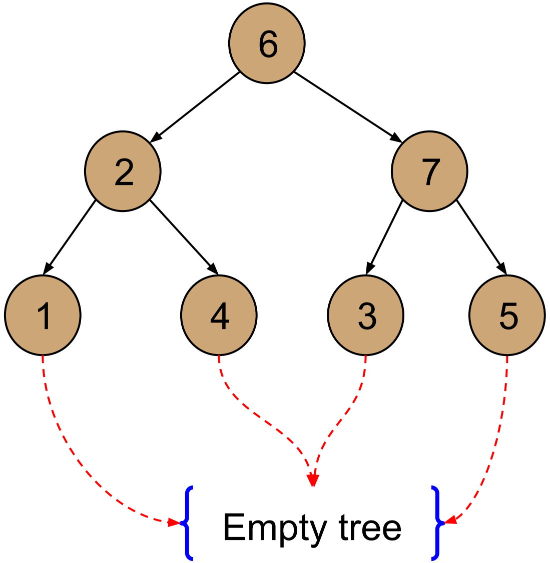
```
void InOrderNonRecursive(BinaryTree * btree){
    Stack stack = CreateStack(&stack);
    BTreeNode * current = btree->root;

    while(1){

        while (current){

            Push(current, &stack);
            current = current ->left;
        }

        current = Pop(&stack);
        cout <<current->data;
        current = current->right;
    }

    ClearStack(&stack);
}
```

Binary Tree Preorder Traversal - Recursive

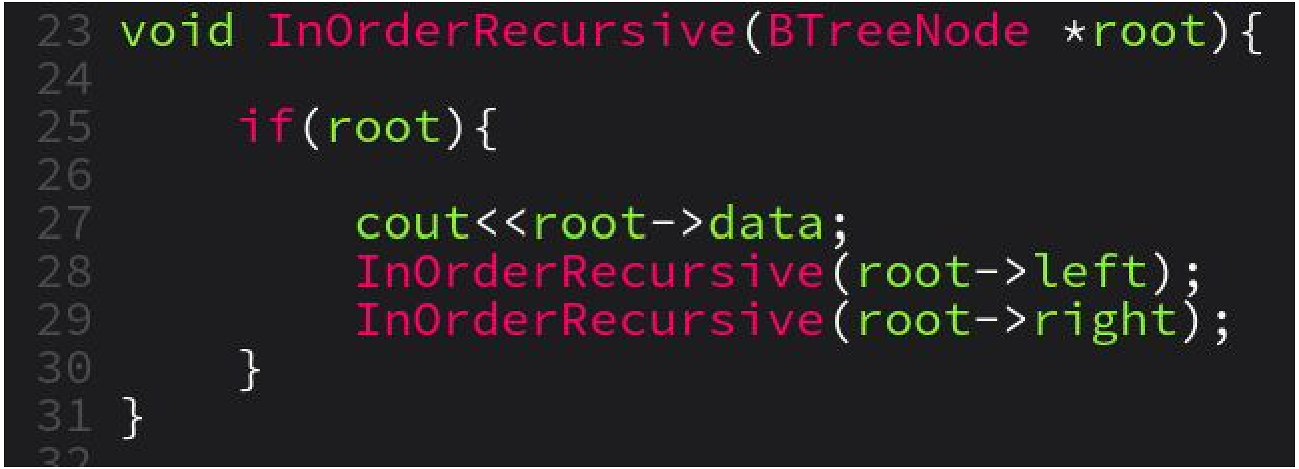
The root node is visited first, then the left subtree and the right subtree.

6, 2, 1, 4, 7, 3, 5

```
While all nodes are not visited:
    Visit root node
    Recursively traverse the left subtree
    Recursively traverse the right subtree
```

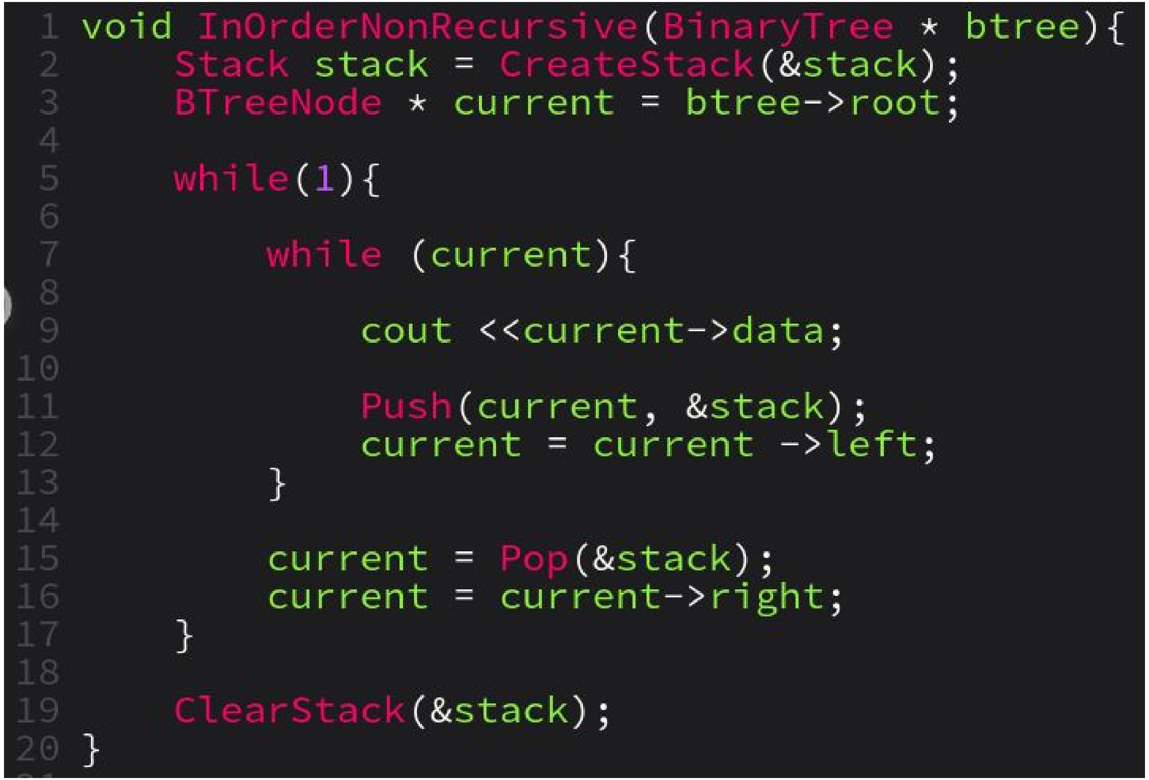


Binary Tree Preorder Traversal - Recursive



```
void InOrderRecursive(BTreeNode *root){

    if(root){

        cout<<root->data;
        InOrderRecursive(root->left);
        InOrderRecursive(root->right);
    }
}
```

Binary Tree Preorder Traversal - Iterative



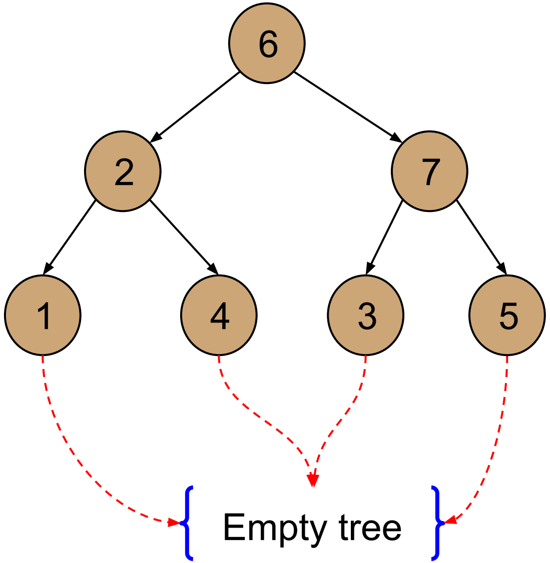
```
void InOrderNonRecursive(BinaryTree * btree){
    Stack stack = CreateStack(&stack);
    BTreeNode * current = btree->root;

    while(1){

        while (current){

            cout <<current->data;

            Push(current, &stack);
            current = current ->left;
        }

        current = Pop(&stack);
        current = current->right;
    }

    ClearStack(&stack);
}
```

Binary Tree Postorder Traversal - Recursive

The the left subtree, then the right subtree, finally the root

1, 4, 2, 3, 5, 7, 6

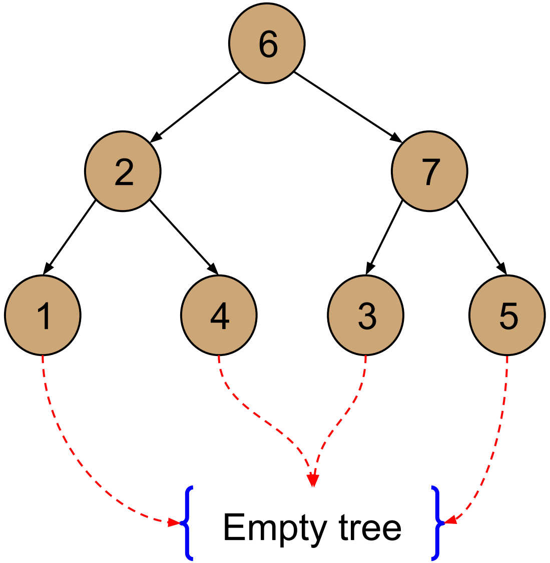
```
While all nodes are not visited:
    Recursively traverse the left subtree
    Recursively traverse the right subtree
    Visit root node
```



Binary Tree Postorder Traversal - Iterative

Write an algorithm to iteratively traverse a binary tree in postorder?

Hint: in postorder every node is visited twice

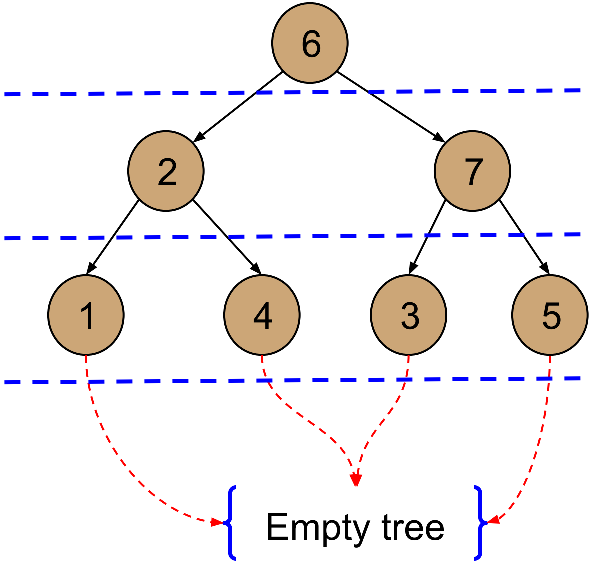


In-Class Activity

Write an algorithm to traverse a binary tree in a level order. Illustrate **your answers** with **sketches**

6, 2, 7, 1, 4, 3, 5

Note: a node in the tree has two pointers only. one to the left subtree and the other to the right subtree. Do not introduce additional pointers



In-Class Activity

Given an array of sorted numbers Give a trace of binary search algorithm by using suitable examples to show that the algorithm constructs a binary tree.