NANYANG TECHNOLOGICAL UNIVERSITY

CE1107/CZ1107: DATA STRUCTURES AND ALGORITHMS

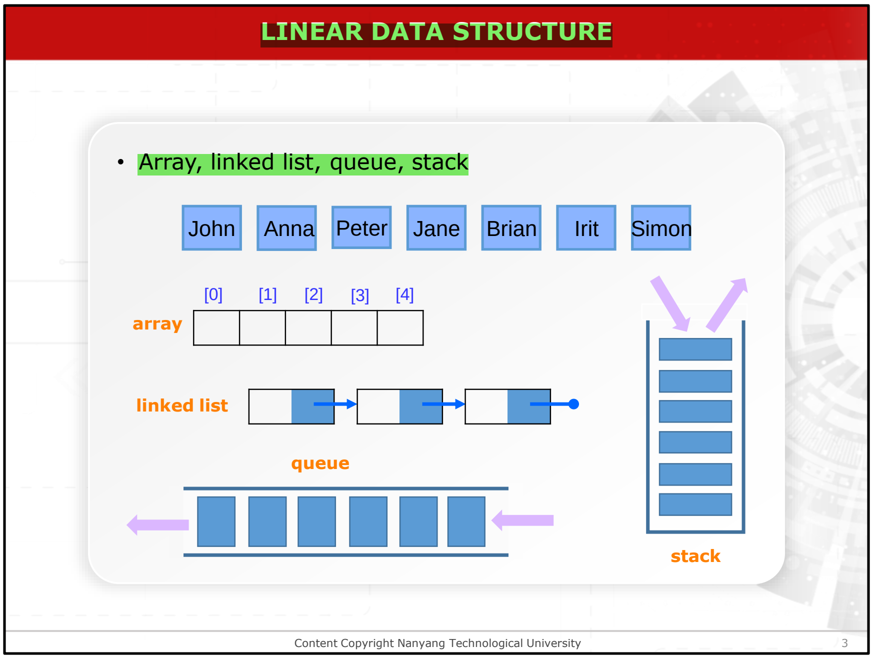
Binary Trees

College of Engineering
School of Computer Science and Engineering

This lecture is on Binary Trees

OUTLINE

- **Non-linear data structures**
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - Calculate height of every node
- Level-by-level traversal
- Preorder traversal with a stack

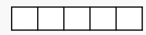


If it is a linear data structure, we can store data as;

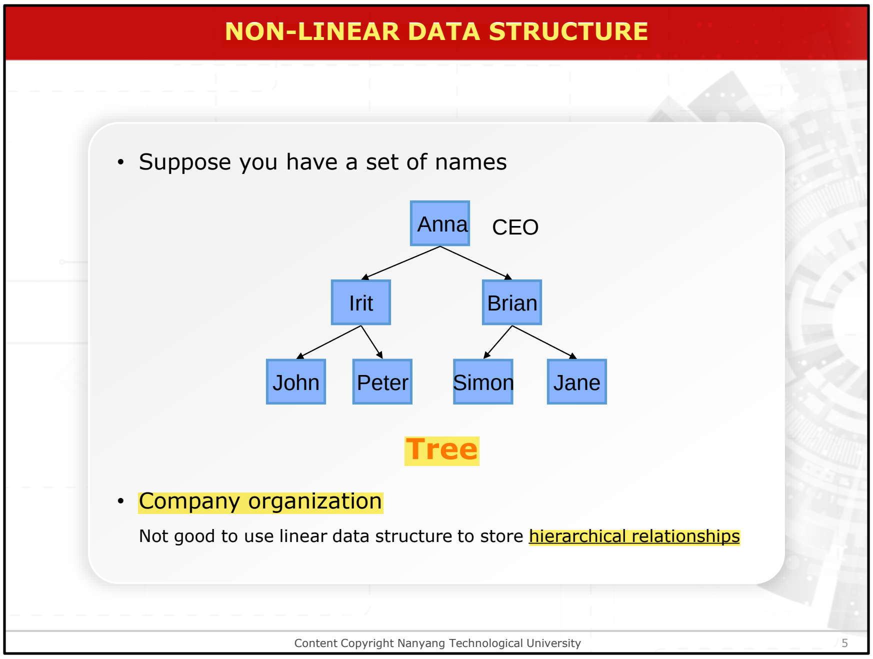
- an array
- linked list
- queue
- stack

DATA STRUCTURES SO FAR…

- Linear
  - Items all arranged one after another
  - Random access
    - Arrays
  - Sequential access
    - Linked list
  - Limited-access sequential
    - Stacks
    - Queues

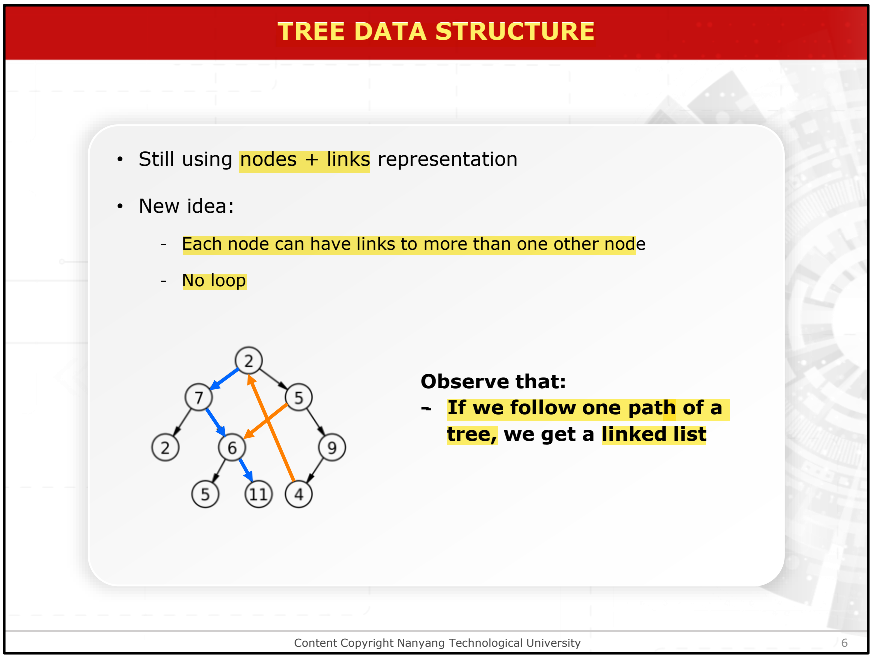


- Used them to store lists of numbers, lists of people, lists of moves, etc
  - Linear data



Non-linear Data structures

Suppose you have some names of few people, who represent a company. Then, to arrange their names according to their position in the company, you have to use a ‘tree’.



Tree data structure

The key points of tree data structure can be listed as below.

1. Each node can have links to more than one other node. - In linear data structure, for an example if we consider queue; a node can be linked with only one other node. But in non-linear data structure, this concept changes.

2. No loop - As per the example show in the slide, if you put a link from node. 5 to node 6, the result would be a loop between 2,7,6,5 nodes. Same result would be given if you form a link from node 4 to node 2.

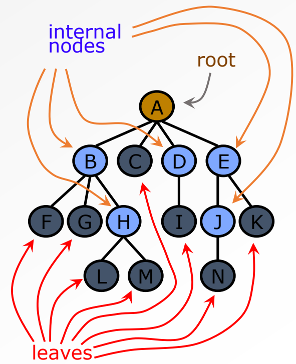
3. A tree data structure contains more than one linked list.

OUTLINE

- Non-linear data structures
- **Tree data structure**
  - **Binary trees**
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - Calculate height of every node
- Level-by-level traversal
- Preorder traversal with a stack

TREE DATA STRUCTURE

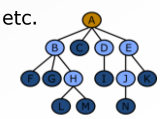
- A tree is composed of nodes
- Each node contains a value
- Types of nodes
  - **Root**: only one in a tree, has no parent.
  - Internal (non-leaf): Nodes with children are called **internal nodes**
  - Leaf: nodes without children are called **leaves**



WHY TREES?

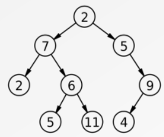
- Model layouts with hierarchical relationships between items
  - Chain of command in the army
  - Personnel structure in a company
  - (Binary tree structure is limited because each node can have **at most two children**)

- Tree structures also allow us to
  - Some problems require a tree structure: some games, most optimization problems, etc.
  - Allow us to do the following very quickly: (we'll see that in the following lectures)
    - **Search for a node with a given value**
    - **Add a given value to a list**
    - **Delete a given value from a list**

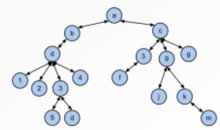


TREE DATA STRUCTURE

- Tree data structure looks like… a tree:
  - Only one root node (no nodes points to it)
  - Each node branches out to some number of nodes
  - Each node has only one "parent" node – the node pointing to it (except the root node)

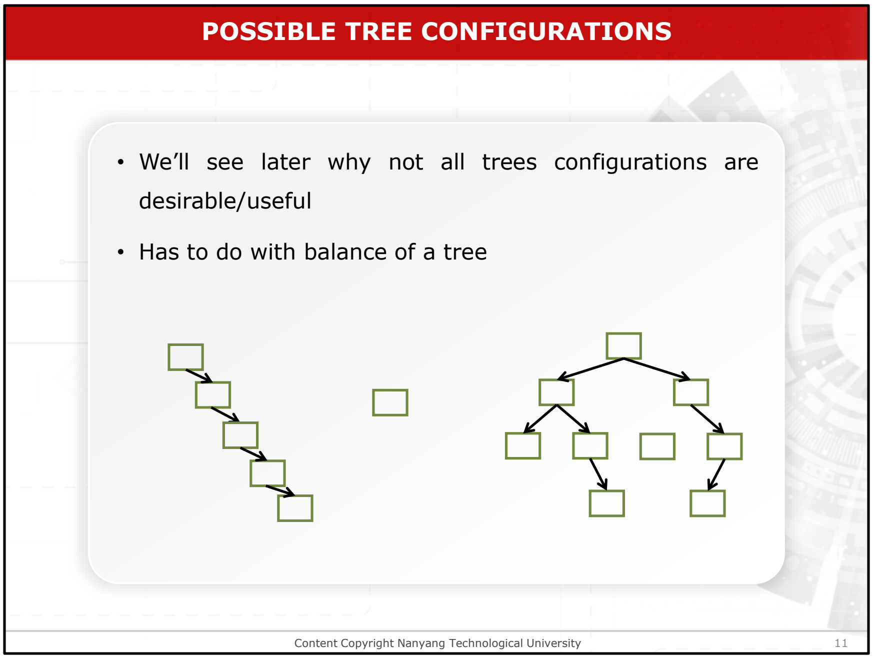


Binary tree



General tree

- General tree
  - Each node can have links to any number of other nodes
- **Binary tree (we'll work with this in our course)**
  - **Each node can have links to at most two other nodes**

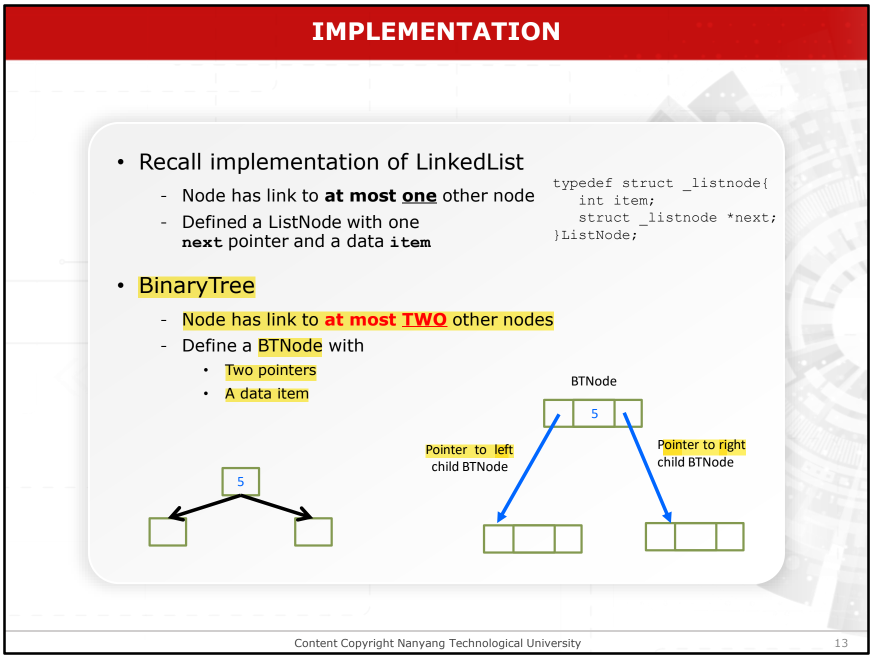


From the given examples in the slide;

- The first one can be recognizes as a tree, but it is an unbalanced tree.
- The second one has only one node, yet it can be defined as a tree
- The third one is also a tree.

OUTLINE

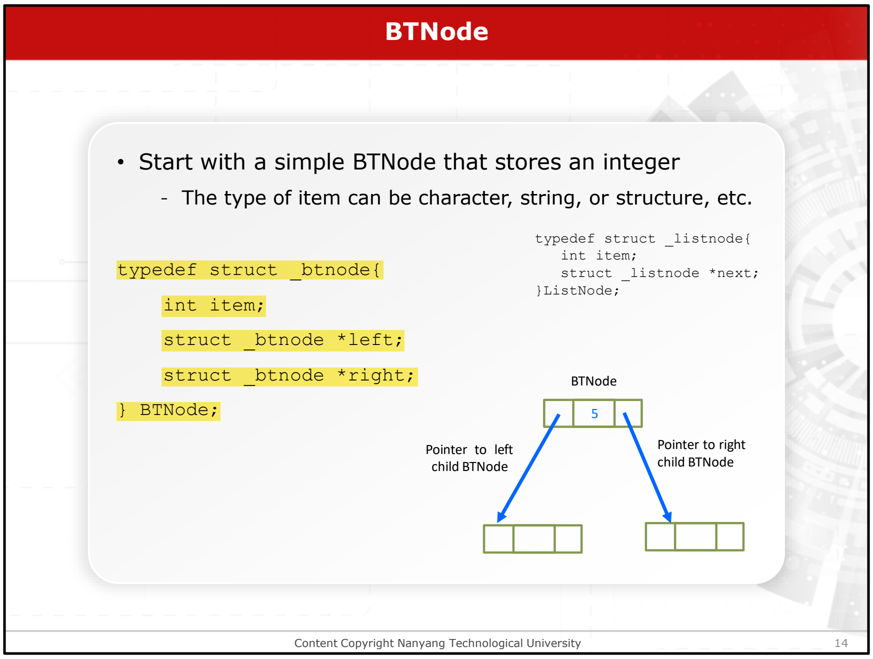
- Non-linear data structures
- Tree data structure
  - Binary trees
- **Implement binary tree nodes in C**
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - Calculate height of every node
- Level-by-level traversal
- Preorder traversal with a stack



The C structure we have implemented for list node is;

```
Typedef struct _listnode {
        int item;
        struct_list node *next;
} ListNode;
```

The same method can be used for tree structure as well. But binary tree node has links to at most two other nodes. Therefore, there should be two pointers.



The C structure for binary tree node

- The structure for binary tree can be defined as;

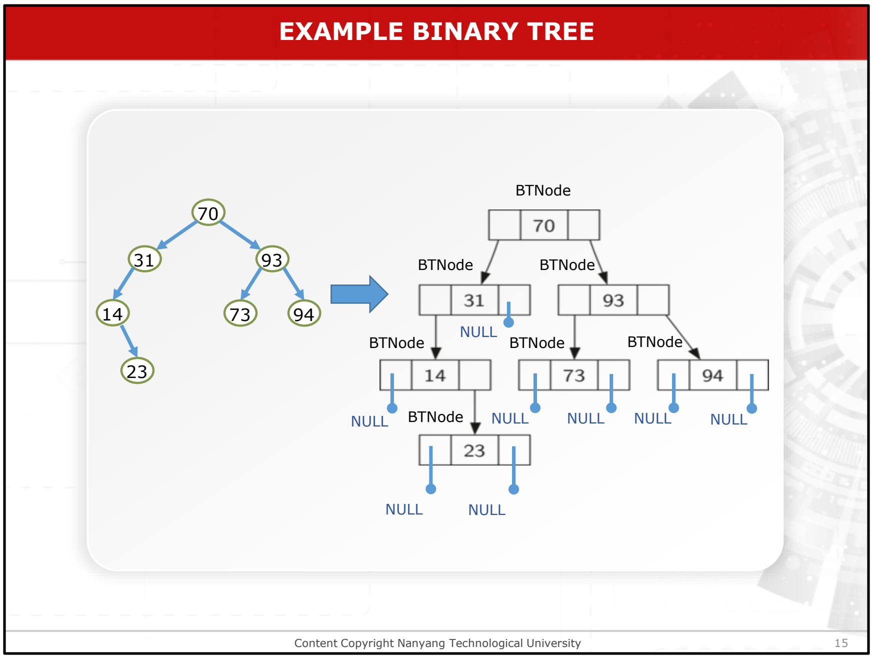
  Typedef struct_btnode {…}

- We have to define one integer variable to store values for the node.

  int item;

- Then we have to declare two pointers which pointers to the left and right child nodes.

  struct_btnode *left;
  struct_btnode *right;



We have constructed the node structure based on the given tree.

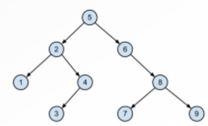
- Node 70 is the root node which has the right pointer points at node 93 and the left pointer points at node 31.
- Node 31 has only a left pointer which points at node 14, and node 14 has only a right pointer points at node 23.
- Node 93 has two pointers where left pointer points at node 73 and the right pointer points at node 94.

OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- **Binary Tree Traversal**
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - Calculate height of every node
- Level-by-level traversal
- Preorder traversal with a stack

TREE TRAVERSAL

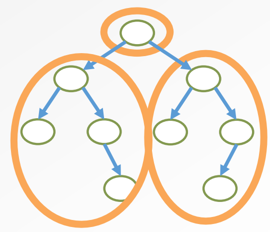
- Given a linear data structure and a particular item, very obvious what the "next" item is
  - Each node has an obvious "previous" and "next" node
- Trees are non-linear structures
  - How to extract data from a binary tree?
  - What is the traversal sequence? left/left/left, then left/left/right, then…?
- Need a systematic way to visit every node in the tree
  - Clearly defined steps
  - No repeated visits to nodes



TREE TRAVERSAL

- Why is this important?
  - Tree traversal is foundation for many functions
- Very common function template:

  Traverse tree
  - At each node, perform some operation

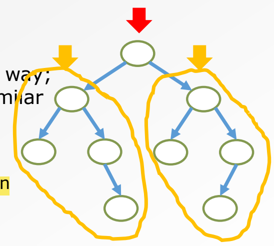


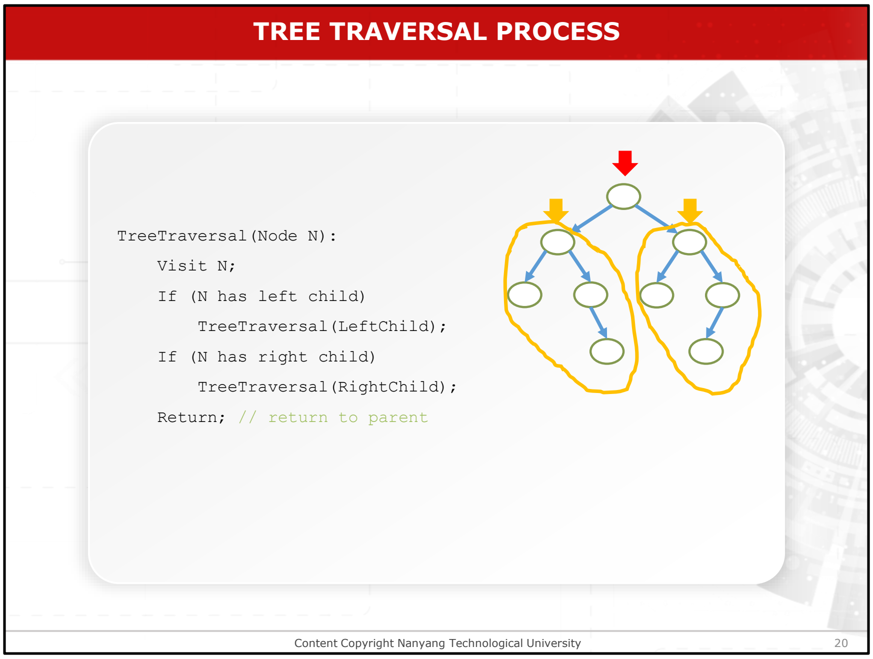
- Example task: count # of nodes in a tree

  At every node N, size of that subtree
  = size of N's left subtree
  + size of N's right subtree
  + N itself

TREE TRAVERSAL

- Tree traversal is recursive
  - Recursion: is the process of repeating items in a self-similar way; divide a problem into several similar sub-problems.
  - At each node
    - Visit the node and both children
- Initial case + repeating case
  - (Visit root) + (visit children)
- When combined, guarantees that all nodes will be visited once and only once





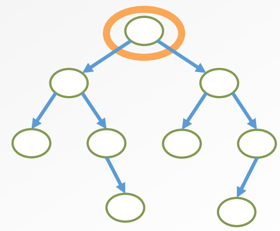
Tree Traversal Process

- The pseudo code of the tree traversal process:
- Since we try to count the number of nodes in the whole tree, we have to start with the root node. Therefore Node N is the root node.
- First we will visit the root node
- Then we check whether the root node has a leftchild. If it has, we then traverse the left subtree of the root which start with leftchild.
- Then we check the right child of the root in the same manner.

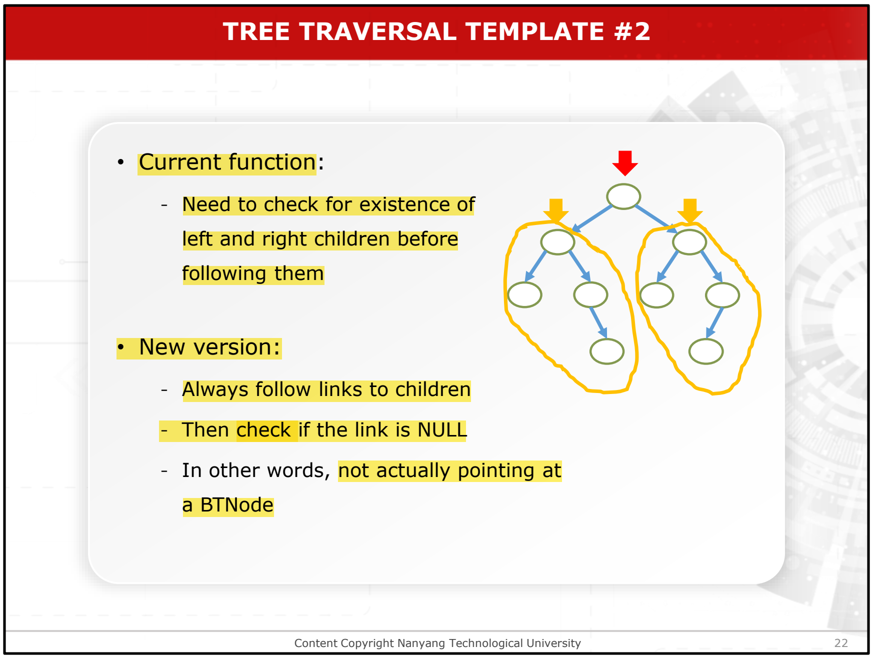
TREE TRAVERSAL TEMPLATE #1

Pseudocode

```
TreeTraversal(Node N):
    Visit N;
    If (N has left child)
        TreeTraversal(LeftChild);
    If (N has right child)
        TreeTraversal(RightChild);
    Return; // return to parent
```



In main(), call TreeTraversal(root)

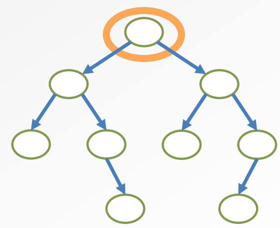


We can improve the discussed code by checking if a node is NULL or NOT NULL after visiting the node.

TREE TRAVERSAL TEMPLATE #2

Pseudocode

```
TreeTraversal2(Node N):
    If N==NULL return;
    Visit N;
    TreeTraversal2(LeftChild);
    TreeTraversal2(RightChild);
    Return; // return to parent
```



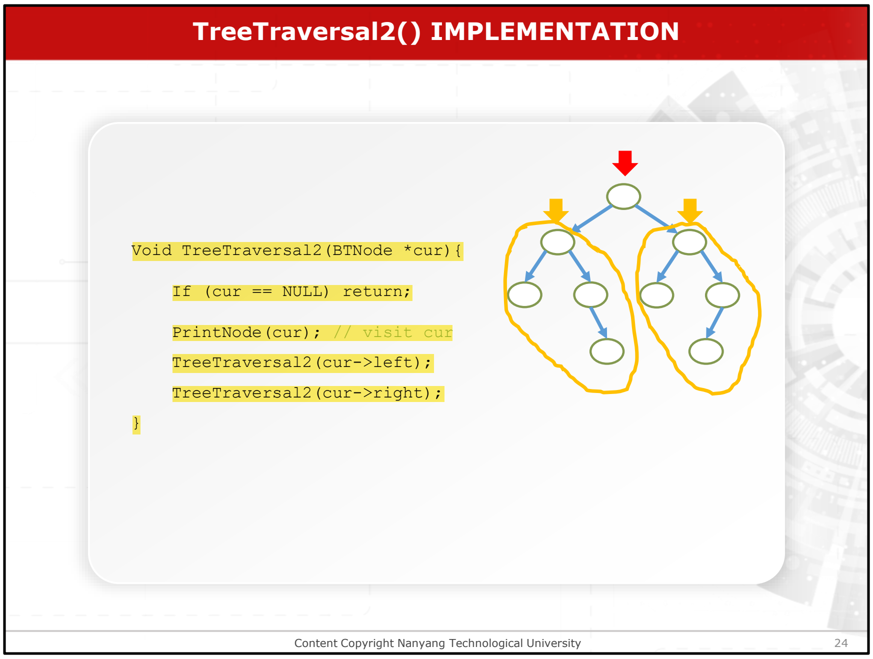
In main(), call TreeTraversal2(root)

Tree traversal template #2

Here, we points at a node and first check if the node is NULL or NOT NULL.

If the node is NULL, the code will return. Otherwise it visits N and pass the TreeTraversal (LeftChild) and TreeTraversal (RightChild) functions.

This code has only one ‘if’ statement, but the previous template we had two ‘if’ statements.



Tree Traversal2() implementation

The code for the 2nd template.

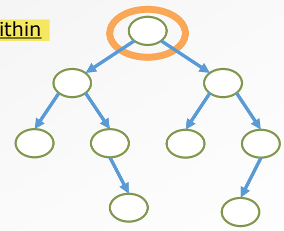
- First we define the TreeTraversal2() function where a pointer 'cur' will be passed with the BinaryTreeNode type.
  Void TreeTraversal2(NTNode *cur){…}

- Then we check if the pointer points at a NULL node. If yes, we will return. Otherwise we print the Node.
  if (cur==NULL) return;
  printNode cur;

Next, we continue on accessing cur->left and cur->right.
TreeTraversal2 (cur->left);
TreeTraversal2 (cur->right);

TREETRAVERSAL() FEATURES

- Recursive
  - TreeTraversal() is called from within its own body
  - initial call TreeTraversal(root)



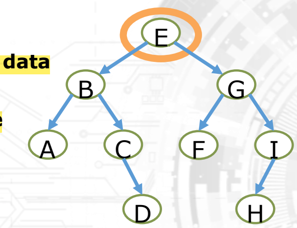
- Depth-first
  - The traversal goes as deep as possible before backtracking and going sideways
  - Not level-by-level! (that is called breadth-first)

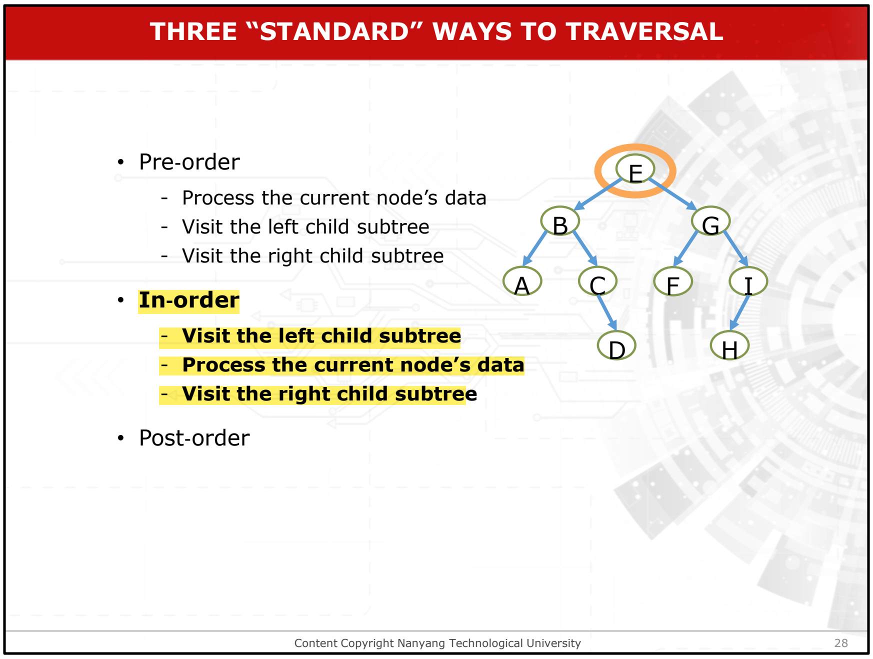
OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- **Tree traversal order**
  - **Pre-order**
  - **In-order**
  - **Post-order**
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - Calculate height of every node
- Level-by-level traversal
- Preorder traversal with a stack

THREE "STANDARD" WAYS TO TRAVERSAL

- Pre-order
  - **Process the current node's data**
  - **Visit the left child subtree**
  - **Visit the right child subtree**
- In-order
- Post-order





In-order traversal

- In in-order traversal we first visit the left child subtree.

    - For an example if we consider the given tree in the slide, the root node F is where we start traversing. Therefore at first we move the pointer from F to B. Now for B also, first we have to visit the left child subtree. Therefore we move the pointer from B to A.

- Now for A there are no left child subtree therefore we proceed to process the current node's data.

- Then we visit the right child subtree; but A does not have a right child. Therefore we move the pointer back to B.

- Since B's left child subtree is already completed, we can proceed to B's data.

- Then we will visit to right child subtree of B which is C. Now where have to visit C's left child subtree; but C does not have a left child subtree. Therefore we can proceed with C's data and move the pointer to it's right child subtree which is D.

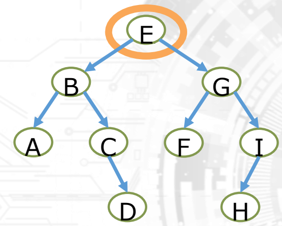
- For D there is no left child or right child sub tree, therefore we proceed with D’s data and return to it’s parent C.

- C is already visited therefore we return to B and then to F.

- Now we have already visited the left child subtree of F. Therefore we can proceed with F’s data.

THREE "STANDARD" WAYS TO TRAVERSAL

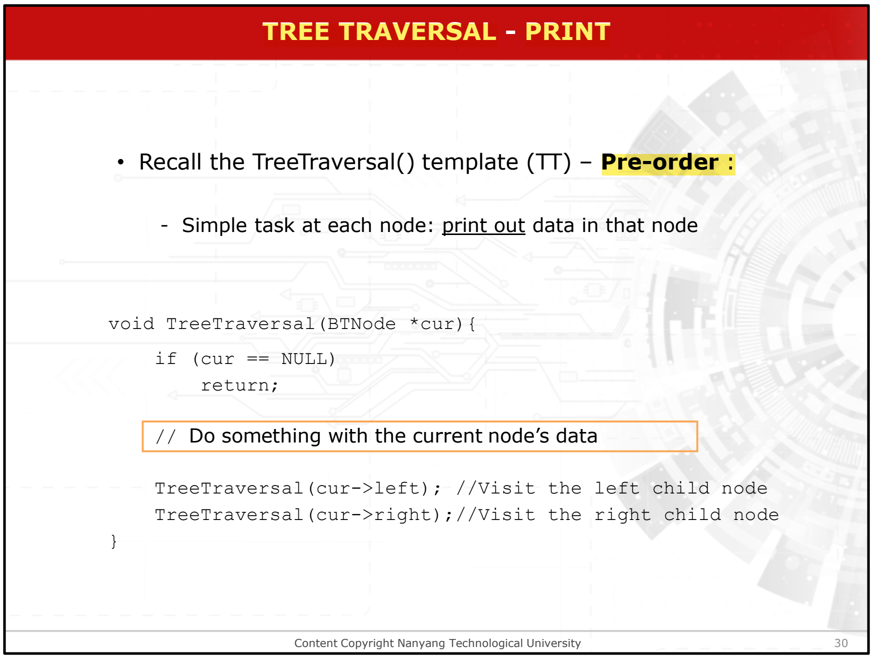
- Pre-order
  - Process the current node's data
  - Visit the left child subtree
  - Visit the right child subtree
- In-order
  - Visit the left child subtree
  - Process the current node's data
  - Visit the right child subtree
- **Post-order**
  - **Visit the left child subtree**
  - **Visit the right child subtree**
  - **Process the current node's data**



 29

Post-order traversal

In post-order traversal; first we visit the left child subtree, then we visit the right child subtree and finally process the current node's data.



Tree traversal-Print

For the pre-order traversal, we first check whether the current node is NULL or NOTNULL.

If the current node is NOT NULL we visit the current node then visit the left child subtree and finally visit the right child subtree.

TREE TRAVERSAL - PRINT

- Recall the TreeTraversal() template (TT) – **Pre-order** :
  - Simple task at each node: print out data in that node

```
void TreeTraversal(BTNode *cur){
    if (cur == NULL)
        return;

    printf("%c",cur->item);

    TreeTraversal(cur->left); //Visit the left child node
    TreeTraversal(cur->right);//Visit the right child node
}
```

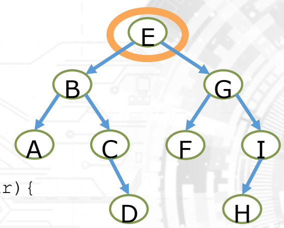

Once the pointer 'cur' points at the current node, we can print the data of the current node by issuing the given code.

print ('%c' , cur->item);

TREE TRAVERSAL PRE-ORDER: PRINT

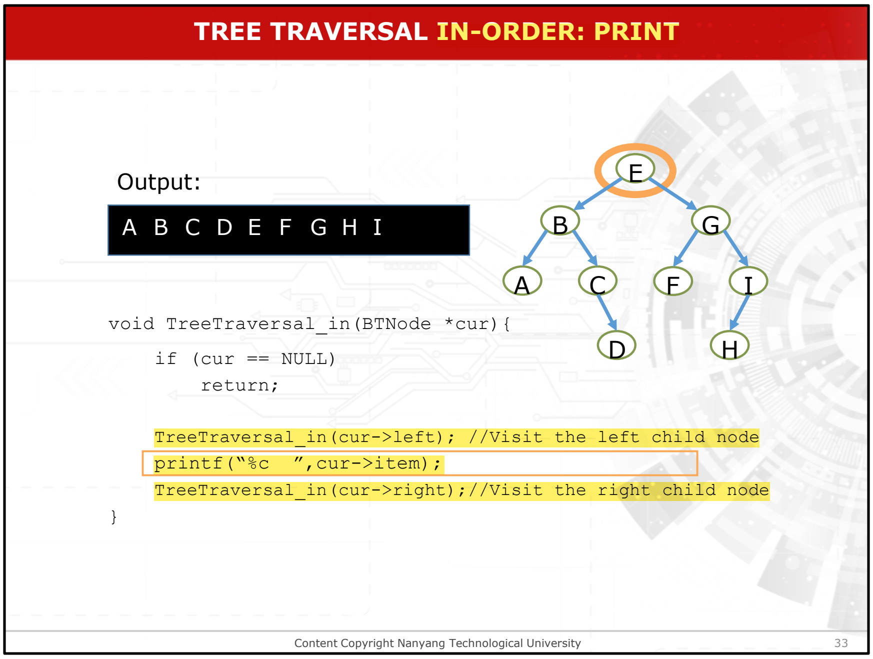
Output:

E B A C D G F I H




```
void TreeTraversal_pre(BTNode *cur){
    if (cur == NULL)
        return;

    printf("%c  ",cur->item);

    TreeTraversal_pre(cur->left); //Visit the left child node
    TreeTraversal_pre(cur->right);//Visit the right child node
}
```


The output of the code would be E, B, A, C, D, G, F, I and H.



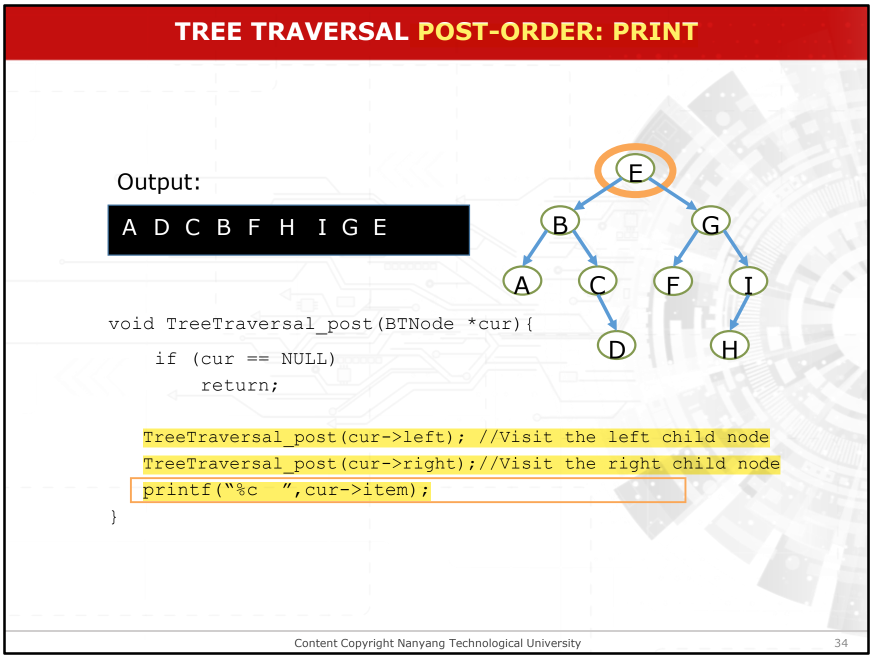
In in-order traversal, after checking whether the current node is NULL or NOT NULL, we check the left child subtree.

TreeTraversal_in(cur->left);

Then we print the current node value and check the right child subtree.

printf("%c", cur->item);
TreeTraversal_in(cur->right);

The output of the code would be A, B, C, D, F, G, H and I

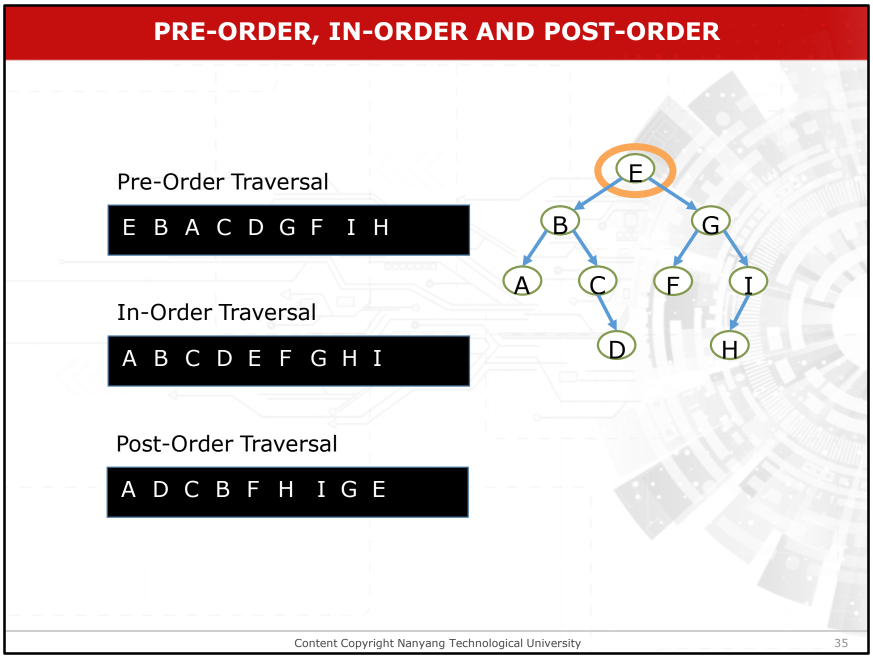


In post-order traversal, we can print the value of the current node after checking it's left and right child subtree.

The output would be A, D, C, B, F, H, I, G and E.

Root node is the last node to be printed in post-order traversal.

Different traversal types outputs different results.



In post-order traversal, we can print the value of the current node after checking it’s left and right child subtree.

The output would be A, D, C, B, F, H, I, G and E.

Root node is the last node to be printed in post-order traversal.

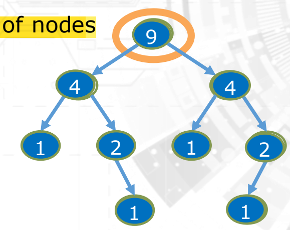
Different traversal types outputs different results.

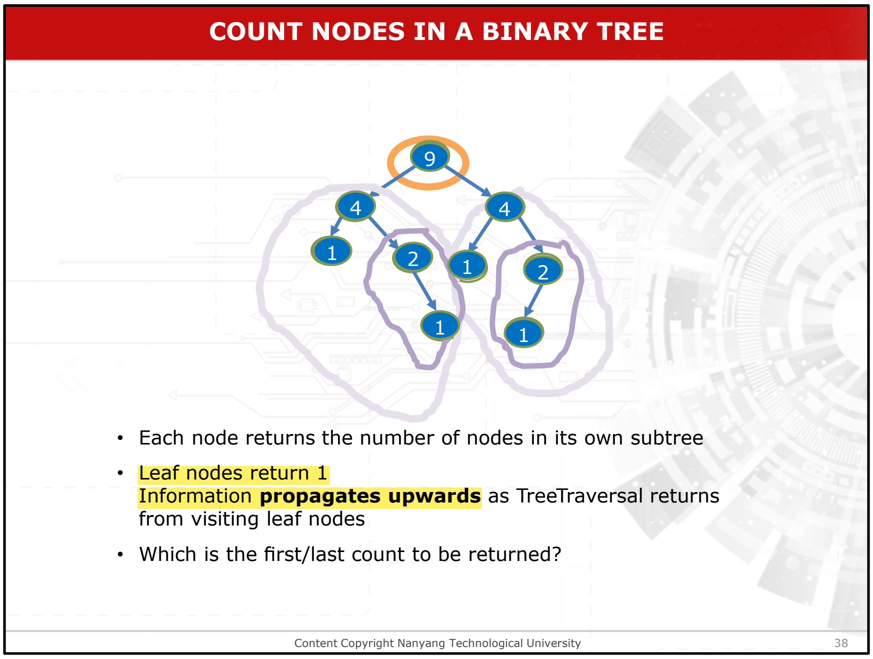
OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - **Count nodes in a binary tree**
  - Find grandchild nodes
  - Calculate height of every node
- Level-by-level traversal
- Preorder traversal with a stack

COUNT NODES IN A BINARY TREE

- Recursive definition:
  - Number of nodes in a tree
    = 1
    + number of nodes in left subtree
    + number of nodes in right subtree
- Each node returns the number of nodes in its subtree





Count nodes in a binary tree

Q: Which is the first count to be returned?

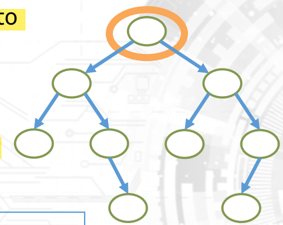
A: The most left leaf node provides the first count

Q: Which is the last count to be returned?

A: The root node provides the last count

countNode()

- Return the size of your subtree to your parent node
- Leaf nodes must return 1 to parent node
- Root node returns size of entire tree



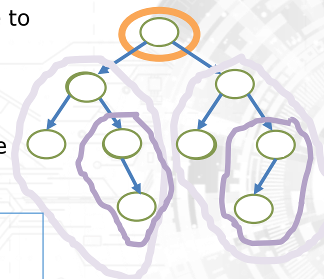
```
void TreeTraversal(BTNode *cur){
    if (cur == NULL)
       return;
    //may do something with cur;
    TreeTraversal(cur->left);
    TreeTraversal(cur->right);
    //may do something with cur;
}
```

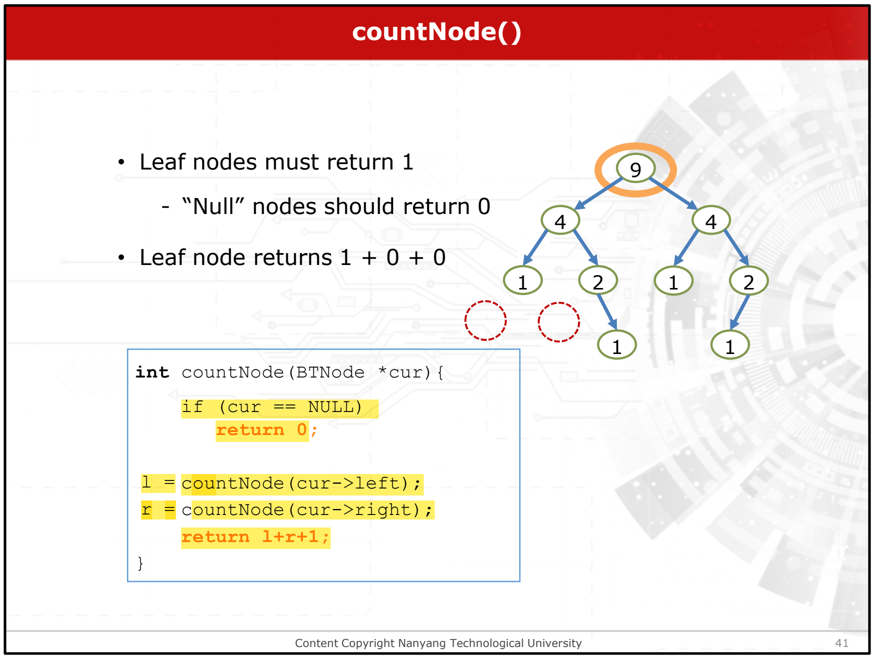

We use TreeTraversal Template to count the no of nodes in Binary Tree

countNode()

- Return the size of your subtree to your parent node
- Leaf nodes must return 1 to parent node
- Root node returns size of entire tree

```
int countNode(BTNode *cur){
    if (cur == NULL)
       return ???;

    countNode(cur->left);
    countNode(cur->right);
    ??? //sum and get total;
}
```





- As learned, we first have to check if the current node is NULL or NOT NULL, if NULL we should return 0;

```
if (cur==NULL)
return 0;
```

- Then we can define two variable as "l" and "r" where "l" gets the count of the nodes in left child subtrees and "r"gets the count of the nodes in the right child subtrees.

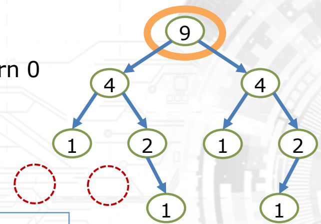
```
l =countNode(cur->left);
r= countNode(cur->right);
```

- After get the values for "l" and "r", we can get the sum of of "l" and "r", add 1 to the sum and return the value;

```
return 1+r+l ;
```

countNode()

- Leaf nodes must return 1
  - "Null" nodes should return 0
- Leaf node returns 1 + 0 + 0



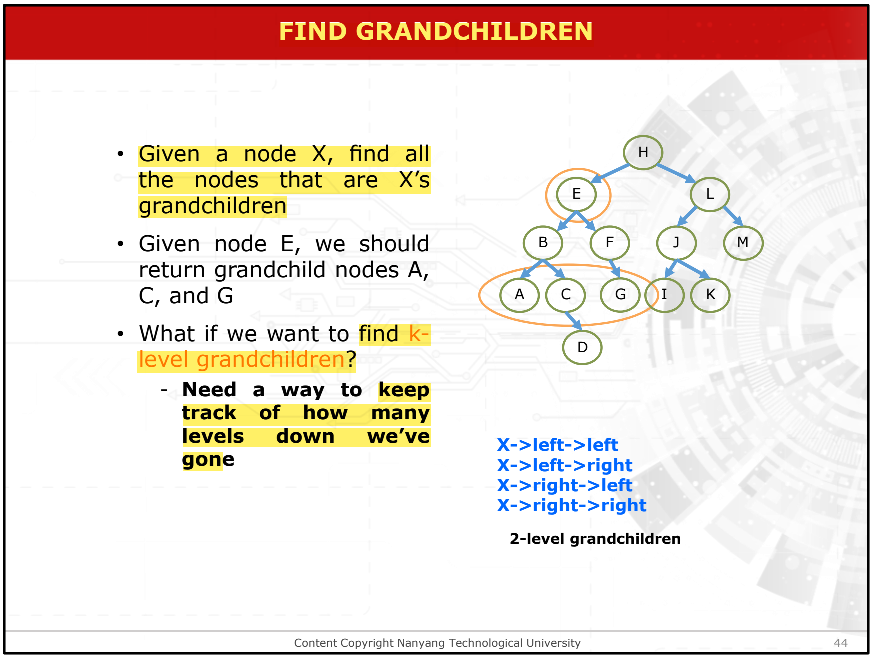

```
int countNode(BTNode *cur){
    if (cur == NULL)
       return 0;

    return (countNode(cur->left)
          + countNode(cur->right)
          + 1);
}
```


We can write the CountNode() function in even lesser complex way;

return (countNode (cur->left) + countNode(cur->right)+1);

OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - **Find grandchild nodes**
  - Calculate height of every node
- Level-by-level traversal
- Preorder traversal with a stack



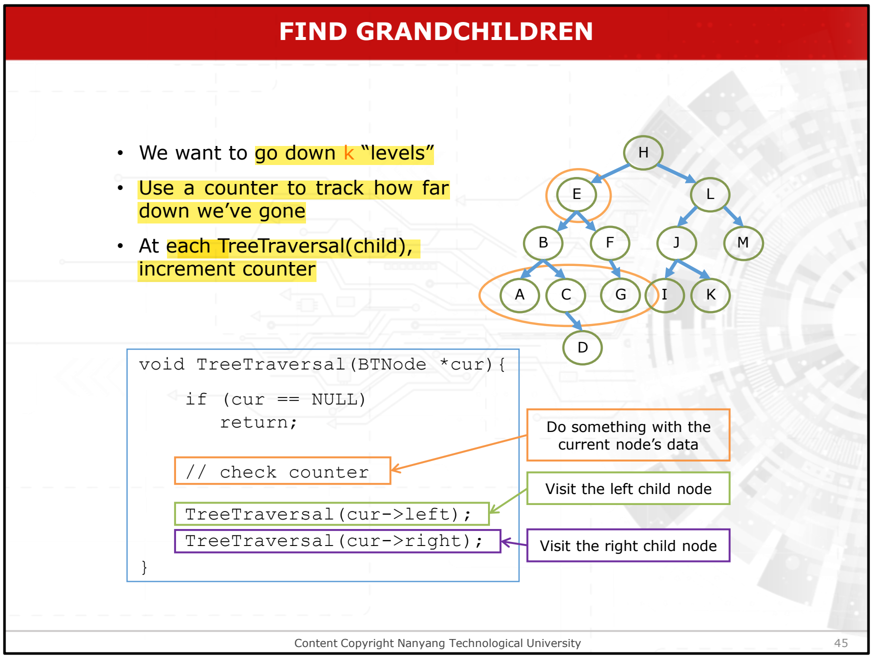
Find grandchildren

As per the given tree, E's grandchildren are A, C and G.

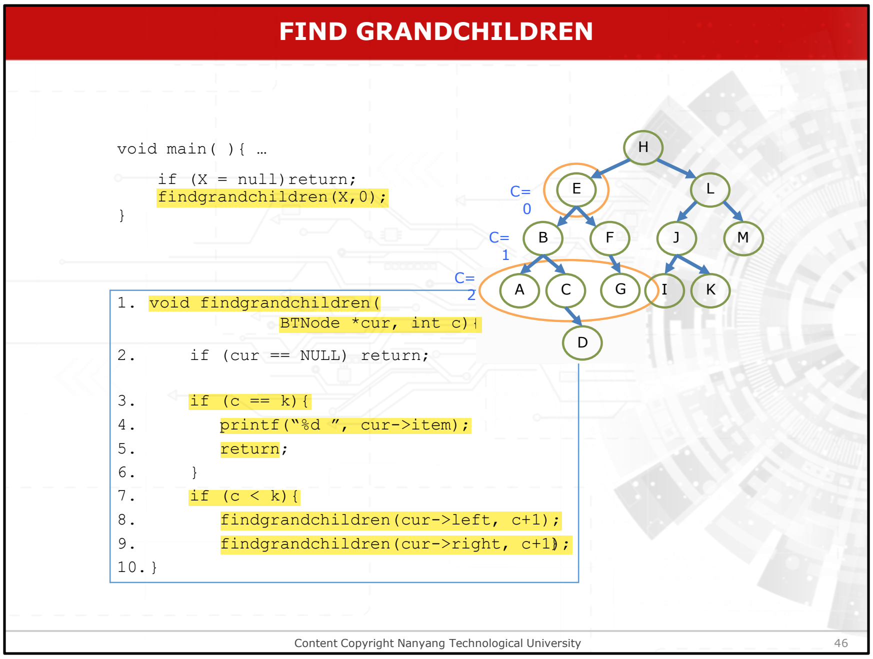
For the given X node, X's grandchildren can be listed as;

x->left-> left
x->left->right
x->right->left
x->right->right

But the above list is only for the 2^{nd} level grandchildren.



To find the K-level grandchildren of a tree we use the TreeTraversal template.



Find grandchildren

- First we declare the findgrandchildren() function. We should pass the pointer and a counter which we use to track how far we go down in the tree.

- Same as in the TreeTraversal template we check whether the currents node is NULL or NOT NULL.

```
if (cur==NULL) return;
```

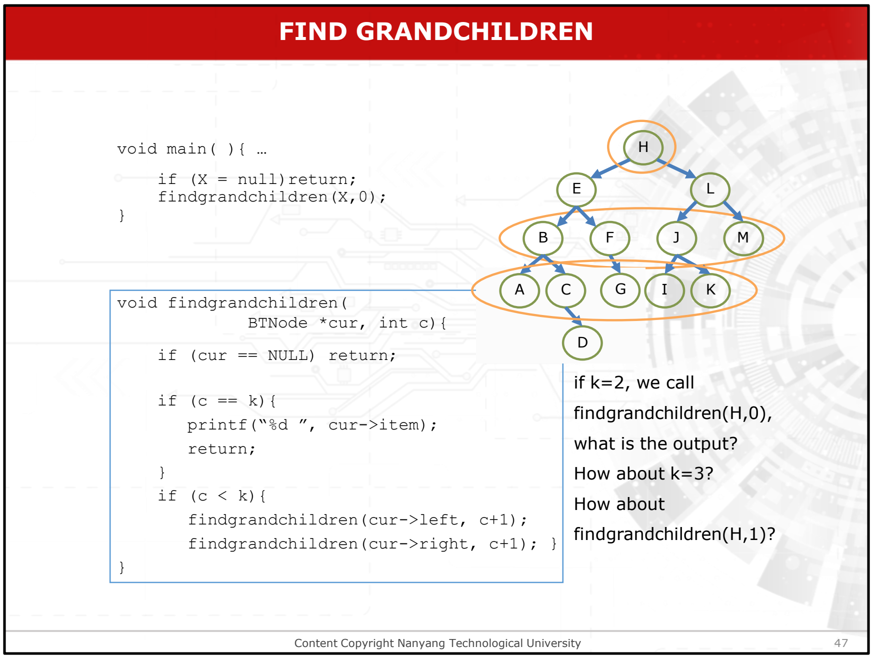
- If the counter is equal to K, we print the value of the current node.

```
if (C==K){
printf("%d",cur->item);
return;
}
```

- Otherwise (C<K) we go on traversing the left child subtree and the right child subtree. We have to increase C by one because we are going down of the tree by 1 level.

```
if (C<K){
findgrandchildren(cur->left, C+1);
```

```
findgrandchildren(cur->right, C+1);
}
```



- If K=2, findgrandchildren (H,0) what is the output? 2 steps down

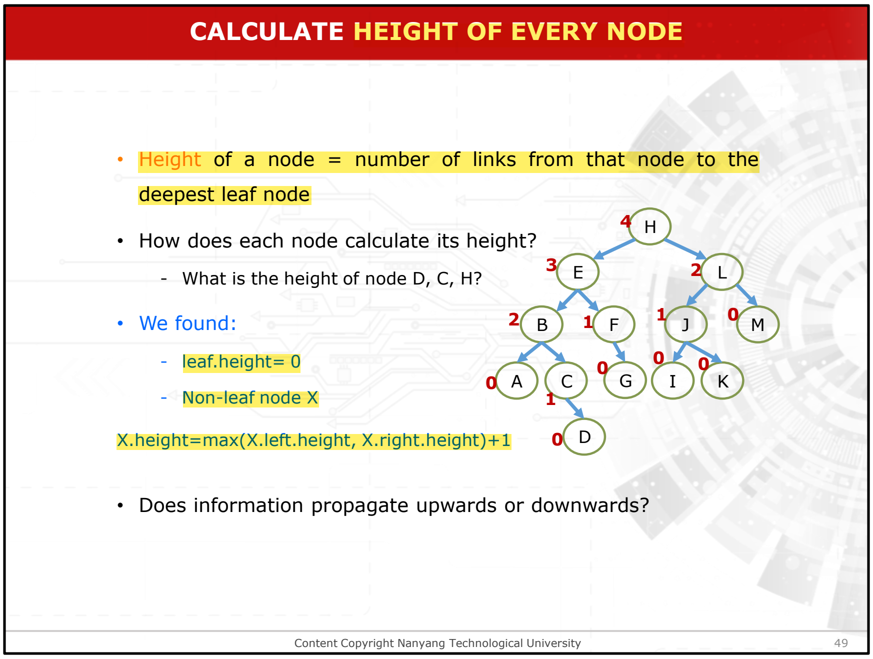
  B, F, J and M
- If K=3;

  A,C,G,I and K 3 steps down
- Findgrandchildren (H,1) when K=3? 2 steps down

  B, F, J and M

OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - **Calculate height of every node**
- Level-by-level traversal
- Preorder traversal with a stack



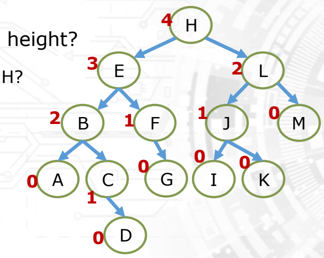
<u>Calculate height of every node</u>

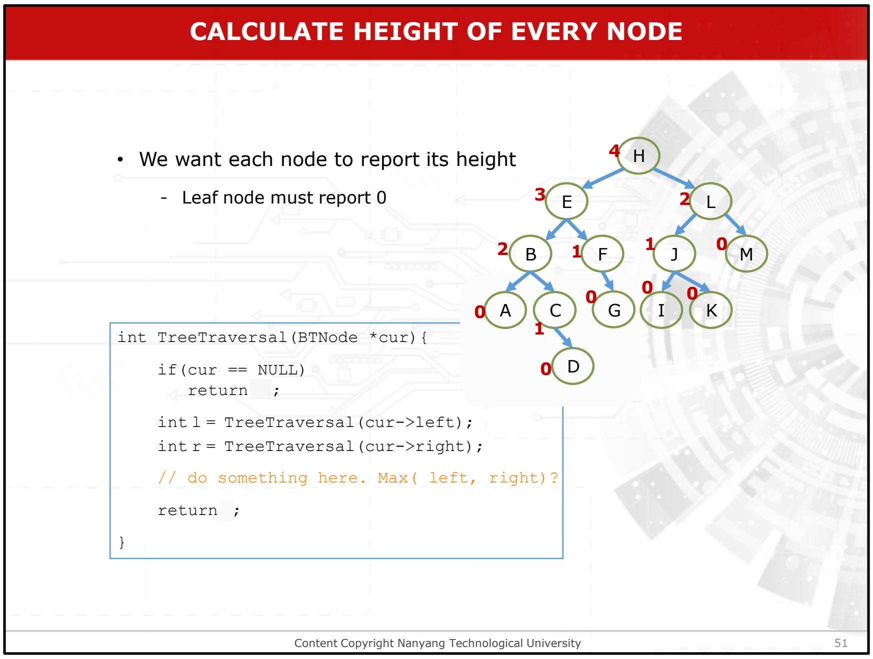
- For an example, if we want to find the height of H we have to check the height of it's children E and L.
- For E we have to check B and F. For B we have to check A and C.
- A is a lead node. Therefore A's height =0
- C's child D also a leaf node. Therefor D's height is 0.
- C's height=D's height +1=1
- For B we have to first find the max heightof it's children and then add 1 to it.
- B's height=max[B.left.height , B.right.height] +1
  = max[C,1]+1
  = 1+1
  = 2

- Likewise we can fill the height of each node as in the slide.
- This time the information propagates upwards.

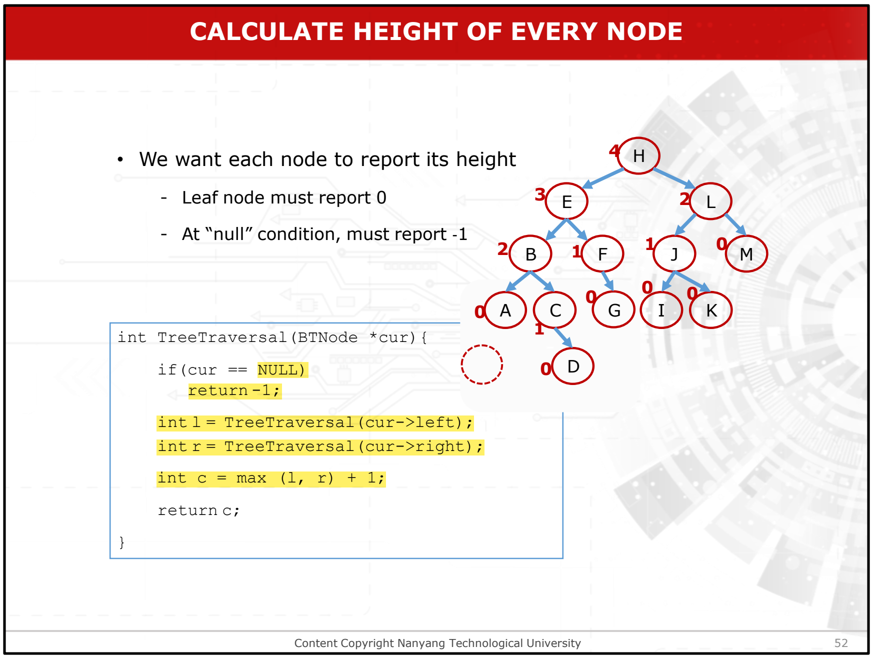
CALCULATE HEIGHT OF EVERY NODE

- Height of a node = number of links from that node to the deepest leaf node
- How does each node calculate its height?
  - What is the height of node D, C, H?
- Go through entire tree: calculate and store height of each node in the item field





We can use the TreeTraversal template for this example.



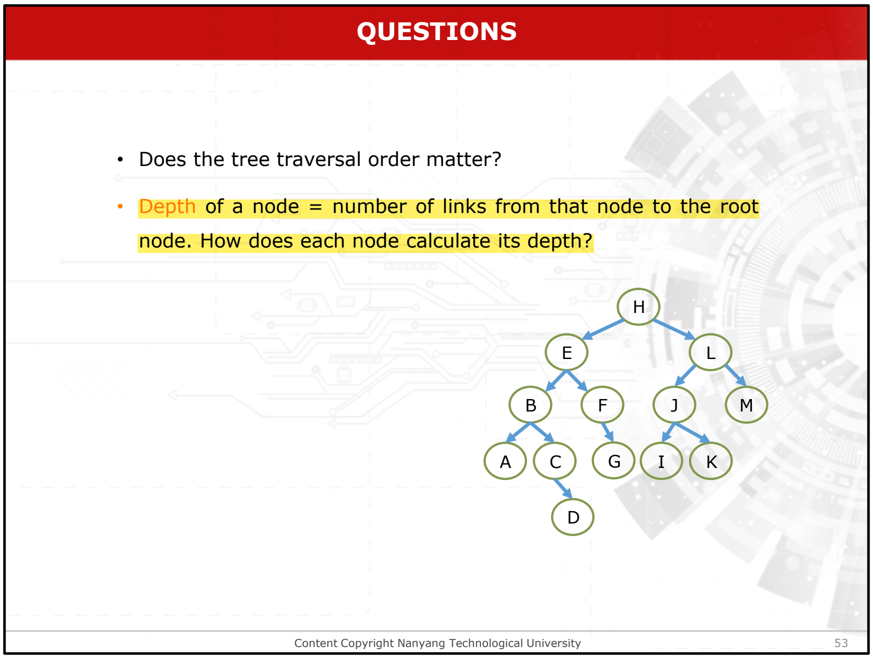
We define two integer variables as "l" and "r" where "l" gets the height of current node's left child subtree and "r" gets the height of current node's right child subtree. Then we can calculate C;

Int c=max(l,r) +1;

Q: At the beginning of the code, we check whether the current node is NULL or NOT NULL. If the current node is NULL should we return C?

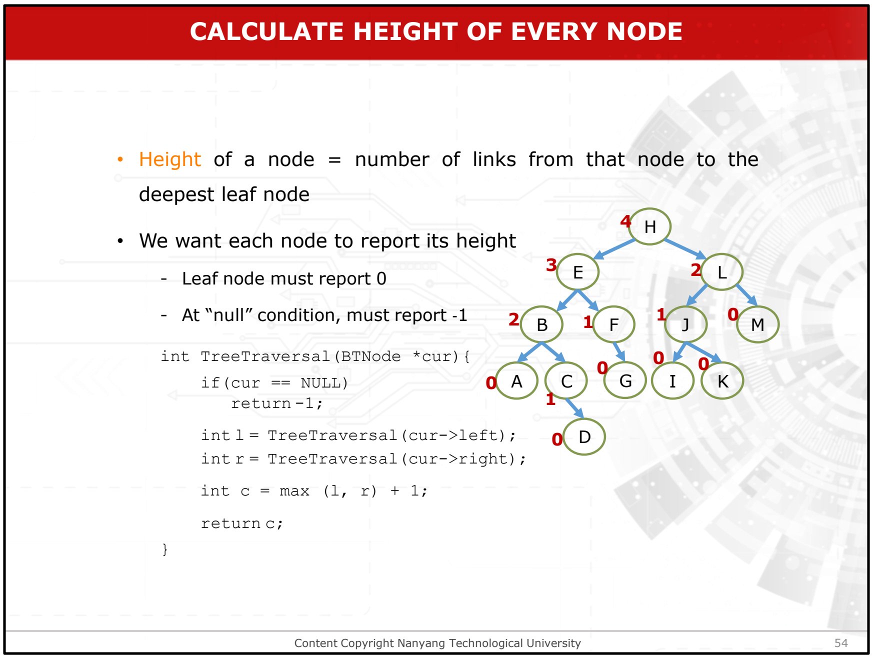
A: No, If we return) to a NULL node; as in the example A is 1 level up to the null node therefore according to the defined code A's height would be 0+1 which is 1.

This is not correct because A is a leaf node. Therefore for the NULL node we have to pass -1.



Post Order Traversal used Here:

- We have used the post-order traversal for this example.
- The tree traversal order should be selected
- Based on the problem
- Depth of a node can be calculated by considering root node’s depth as 0. Here we propagate the information downwards



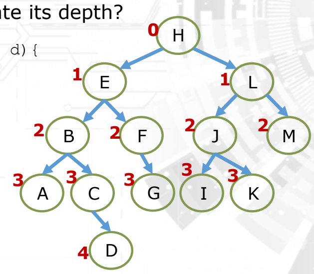
Post Order Traversal used Here:

- We have used the post-order traversal for this example.
- The tree traversal order should be selected based on the problem
- Depth of a node can be calculated by considering root node's depth as 0. Here we propagate the information downwards

QUESTIONS

- Does the tree traversal order matter?
- Height of a node = number of links from that node to the deepest leaf node
- Depth of a node = number of links from that node to the root node. How does each node calculate its depth?

```
void TreeTraversal(BTNode *cur, int d){
    if(cur == NULL)
       return;
    //print cur->item and d;
    TreeTraversal(cur->left, d+1);
    TreeTraversal(cur->right, d+1);
    return;
}
```



Here we propagate the information downwards

- Depth of a node can be calculated by considering root node's depth as 0.
- printf("Depth of Node %c is : %d ", cur->item, d);

OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - Calculate height of every node
- **Level-by-level traversal**
- Preorder traversal with a stack

LEVEL-BY-LEVEL: BREADTH-FIRST SEARCH

Depth-first search

begins at the root and explores as far as possible along each branch before backtracking

E.g. the post-order traversal

Breadth-first search

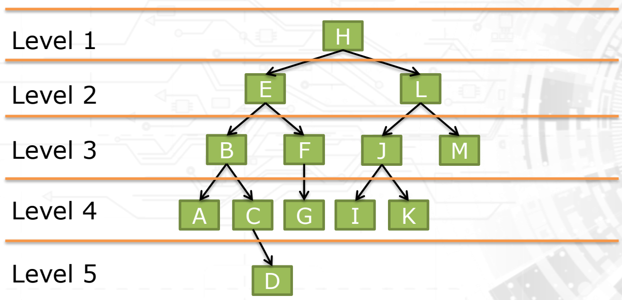
begins at a root node and inspects all its children nodes. Then for each of those children nodes in turn, it inspects their children nodes, and so on.

 57

There are two types of searching mechanisms.

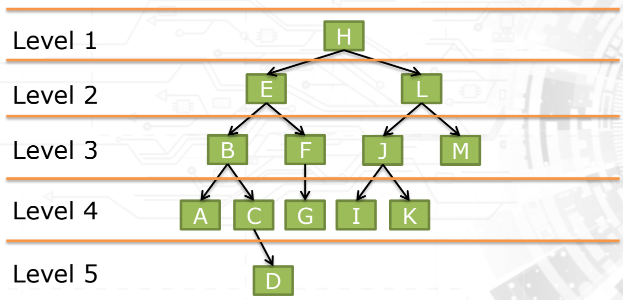
- Depth - first search
- Breadth – first search

LEVEL-BY-LEVEL TREE TRAVERSAL

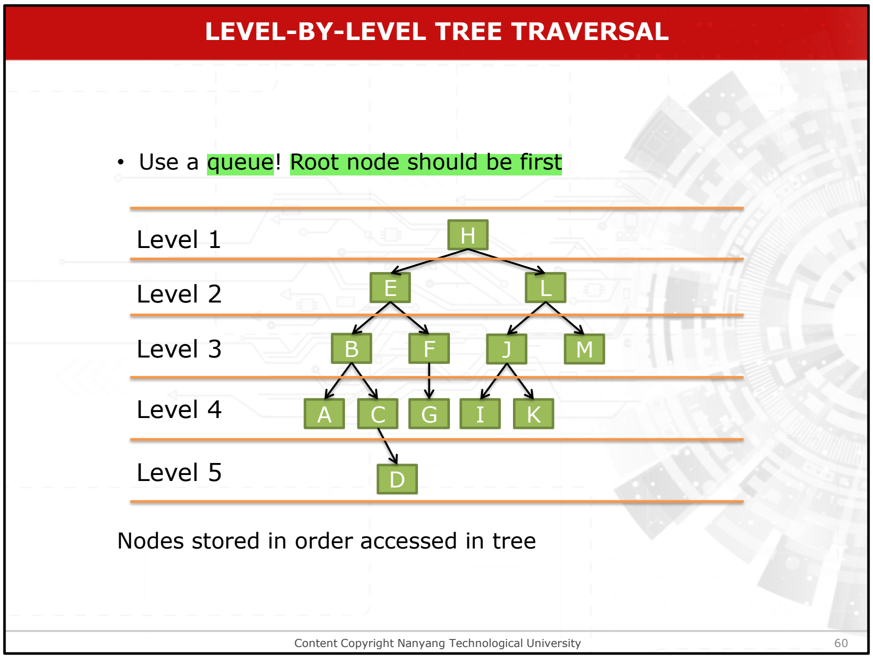


LEVEL-BY-LEVEL TREE TRAVERSAL

- Hint: Make use of another data structure



Nodes stored in order accessed in tree…

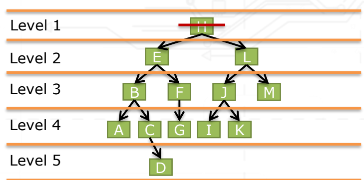


I have a queue which is capable of holding binary tree nodes

You need to change the internal node structure then your queue can hold binary tree nodes.

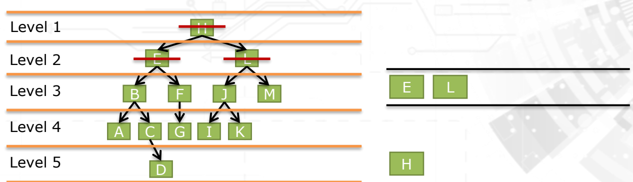
LEVEL-BY-LEVEL TREE TRAVERSAL

- Enqueue the root, H



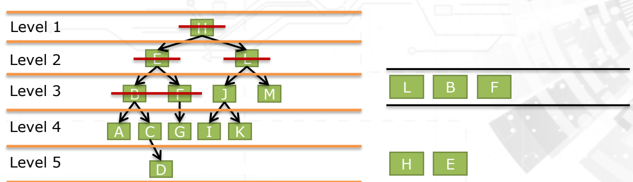
LEVEL-BY-LEVEL TREE TRAVERSAL

- Enqueue the root, H
- Dequeue H, and enqueue H's children



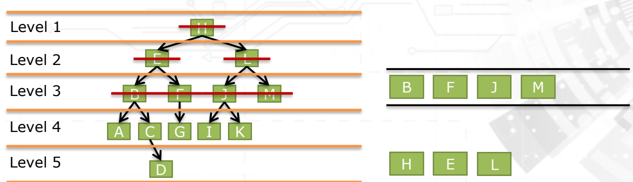
LEVEL-BY-LEVEL TREE TRAVERSAL

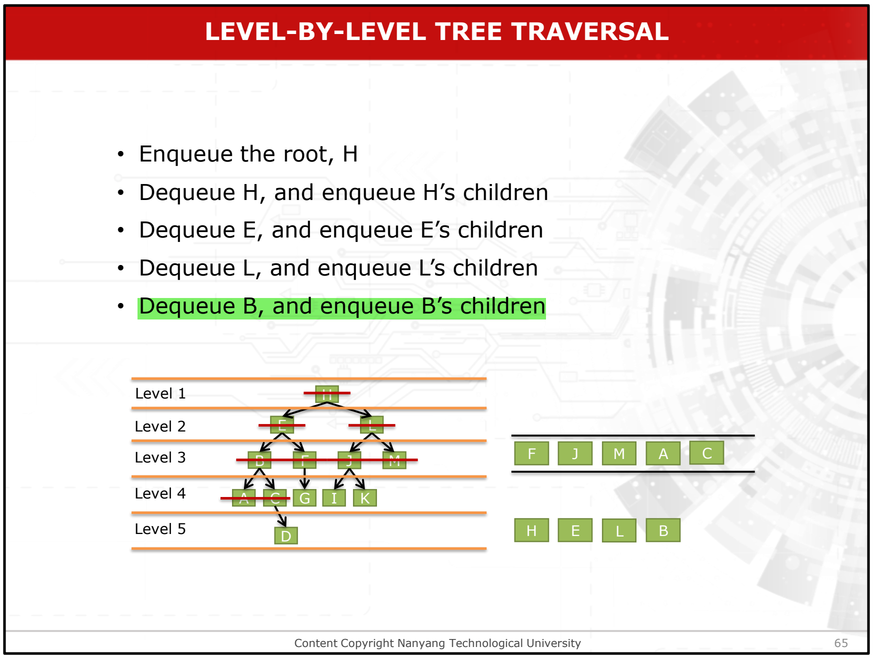
- Enqueue the root, H
- Dequeue H, and enqueue H's children
- Dequeue E, and enqueue E's children



LEVEL-BY-LEVEL TREE TRAVERSAL

- Enqueue the root, H
- Dequeue H, and enqueue H's children
- Dequeue E, and enqueue E's children
- Dequeue L, and enqueue L's children



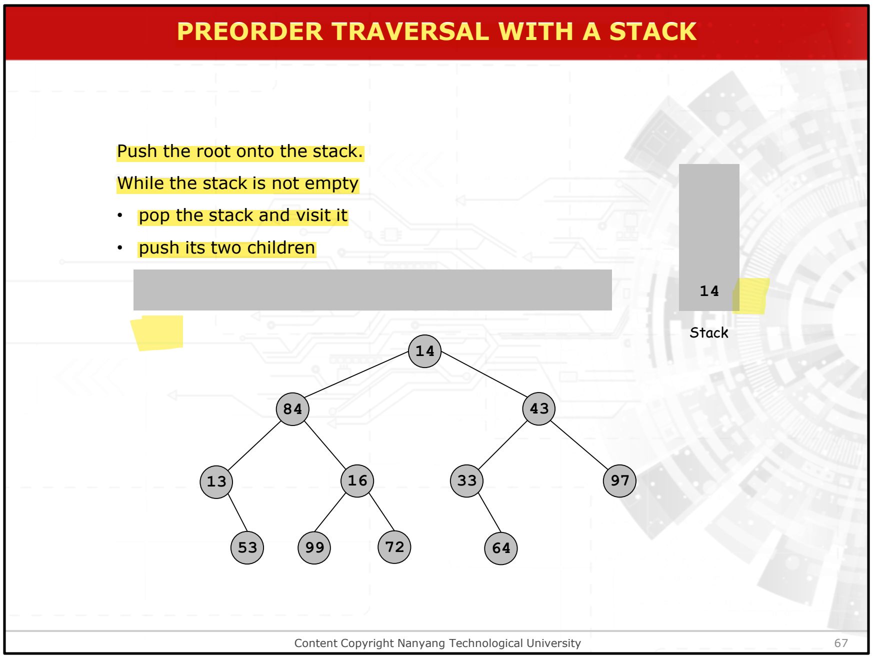


I don't touch level 3 nodes until I process level 2 nodes.

Similarly, I don 't process level 2 nodes until I finished processing level 1 nodes.

OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - Calculate height of every node
- Level-by-level traversal
- **Preorder traversal with a stack**



I have a stack which is capable of holding binary tree nodes

You need to change the internal node structure then your stack can hold binary tree nodes.

PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

While the stack is not empty

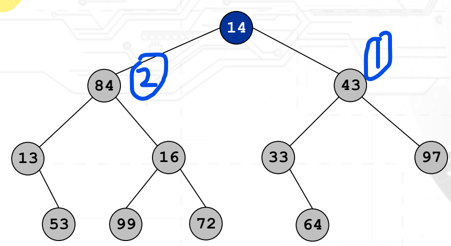
- pop the stack and visit it
- push its two children

14

2
1

84
43

Stack



PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

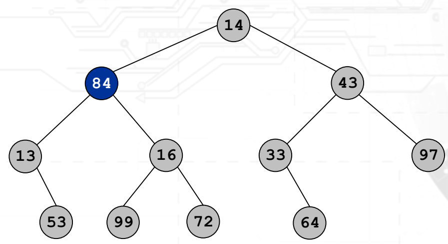
While the stack is not empty

- pop the stack and visit it
- push its two children

14 84

13
16
43

Stack



PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

While the stack is not empty

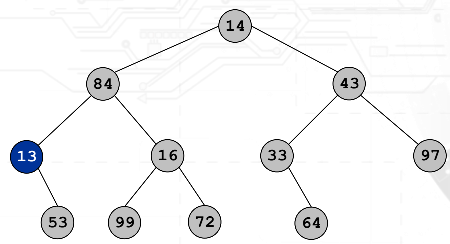
- pop the stack and visit it
- push its two children

14 84 13

Stack:

53
16
43

Stack



PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

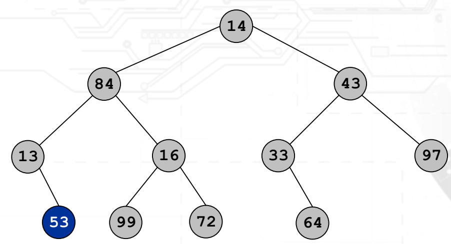
While the stack is not empty

- pop the stack and visit it
- push its two children

`14 84 13 53`

16
43

Stack



PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

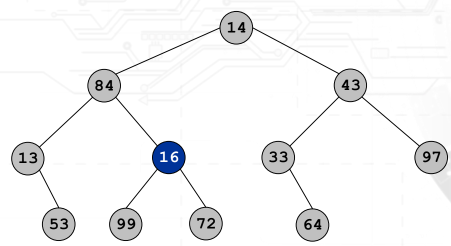
While the stack is not empty

- pop the stack and visit it
- push its two children

`14 84 13 53 16`

99
72
43

Stack



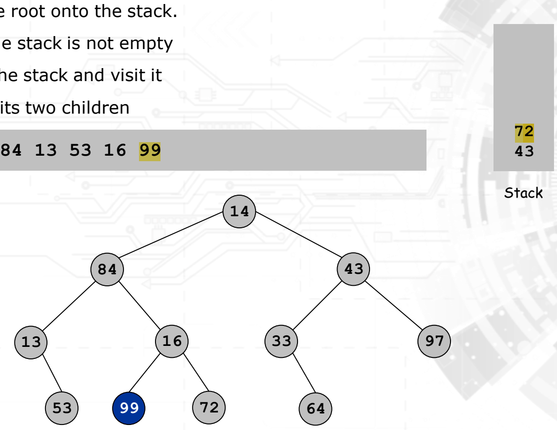
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

`14 84 13 53 16 99`



PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

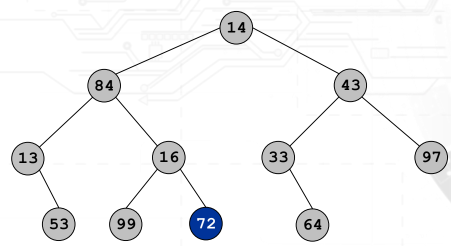
While the stack is not empty

- pop the stack and visit it
- push its two children

`14 84 13 53 16 99 72`

43

Stack



PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

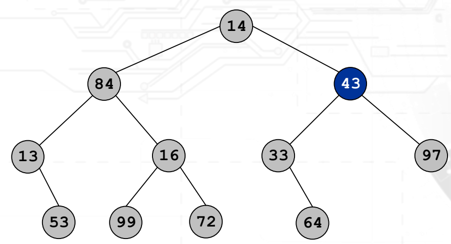
While the stack is not empty

- pop the stack and visit it
- push its two children

`14 84 13 53 16 99 72 43`

33
97

Stack



PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

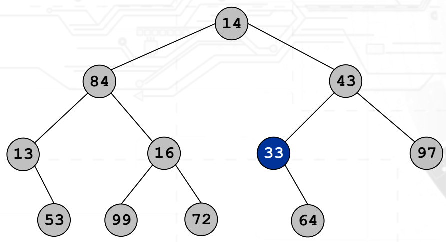
While the stack is not empty

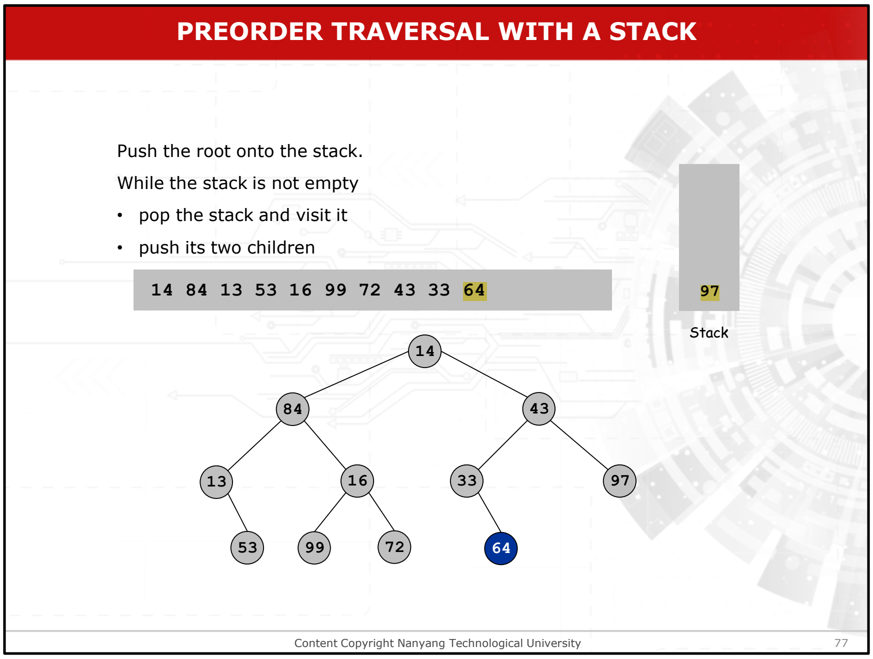
- pop the stack and visit it
- push its two children

`14 84 13 53 16 99 72 43 33`

64
97

Stack





PREORDER TRAVERSAL WITH A STACK
Push the root onto the stack.
While the stack is not empty
• pop the stack and visit it
• push its two children
14 84 13 53 16 99 72 43 33 64
97
Stack
14
84
43
13
16
33
97
53
99
72
64
Content Copyright Nanyang Technological University
77

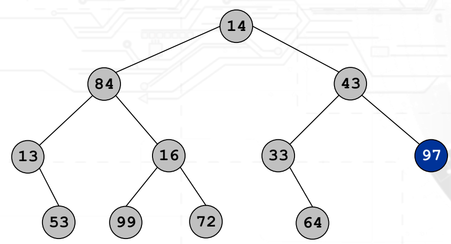
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

`14 84 13 53 16 99 72 43 33 64 97`



PREORDER TRAVERSAL WITH A STACK

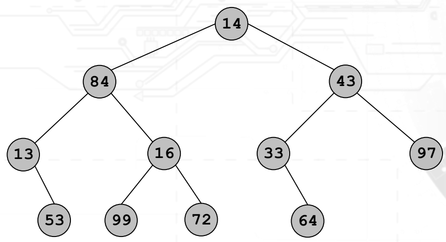
Push the root onto the stack.

While the stack is not empty

- pop the stack and visit it
- push its two children

14 84 13 53 16 99 72 43 33 64 97

Stack



YOU SHOULD BE ABLE TO

- Binary tree Traverse:
  - Pre-order
  - In-order
  - Post-order
- Write recursive binary tree functions using the TreeTraversal template as a starting point
- Based on the traversal of the binary tree, do a lot of things: print, count numbers, count height/depth, find grandchildren,…, etc.

http://nova.umuc.edu/~jark/idsv/lesson1.html