NANYANG TECHNOLOGICAL UNIVERSITY

CE1107/CZ1107: DATA STRUCTURES AND ALGORITHMS

Binary Trees

College of Engineering

School of Computer Science and Engineering

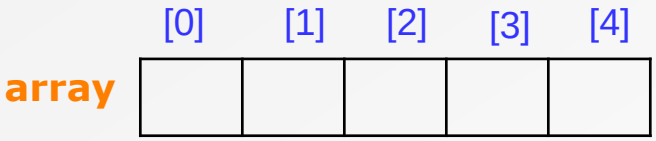
OUTLINE

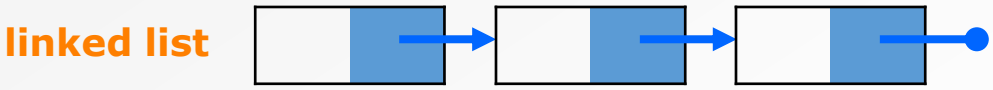
- **Non-linear data structures**
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - Calculate height of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

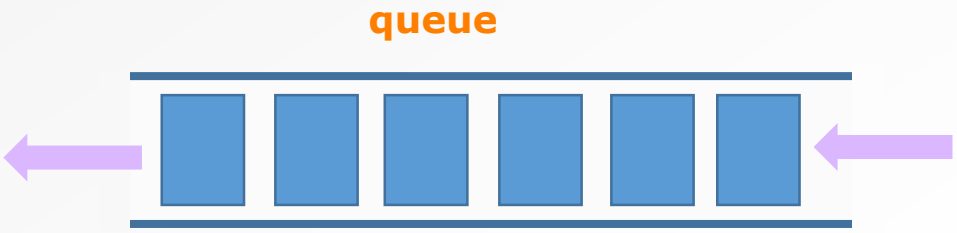
Content Copyright Nanyang Technological University

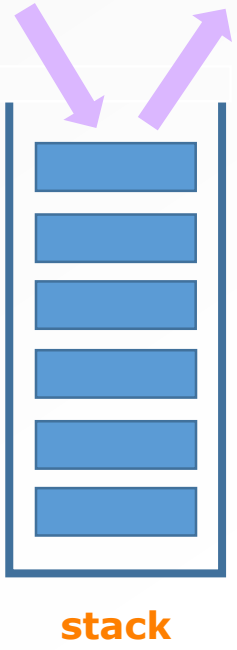
LINEAR DATA STRUCTURE

- Array, linked list, queue, stack





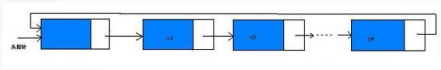




Content Copyright Nanyang Technological University

DATA STRUCTURES SO FAR...

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

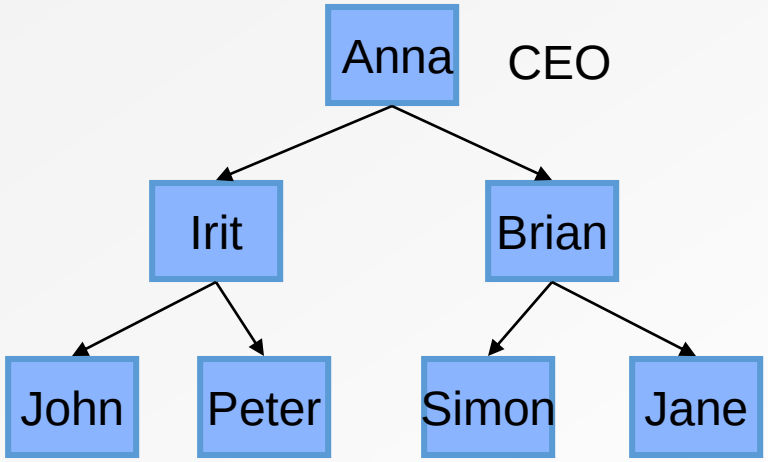


- Used them to store lists of numbers, lists of people, lists of moves, etc
  - Linear data

Content Copyright Nanyang Technological University

NON-LINEAR DATA STRUCTURE

- Suppose you have a set of names



Tree

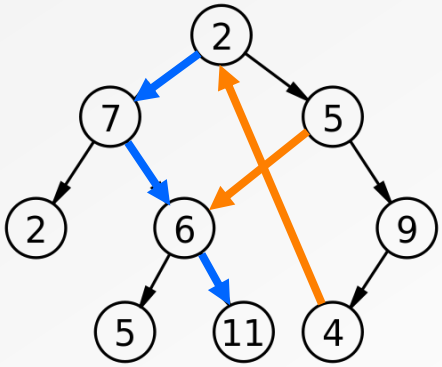
- Company organization

  Not good to use linear data structure to store hierarchical relationships

Content Copyright Nanyang Technological University

TREE DATA STRUCTURE

- Still using nodes + links representation
- New idea:
  - Each node can have links to more than one other node
  - No loop



Observe that:

- **If we follow one path of a tree, we get a linked list**

Content Copyright Nanyang Technological University

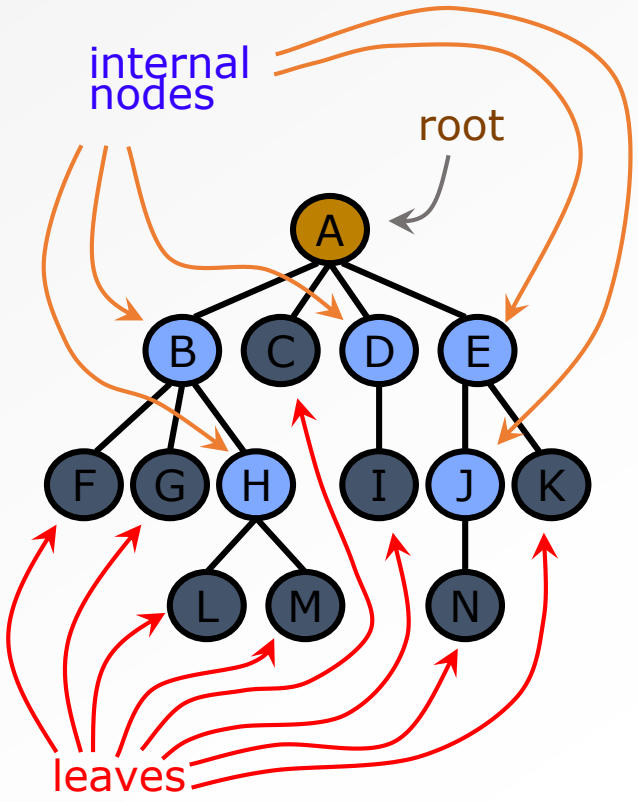
OUTLINE

- 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

Content Copyright Nanyang Technological University

TREE DATA STRUCTURE

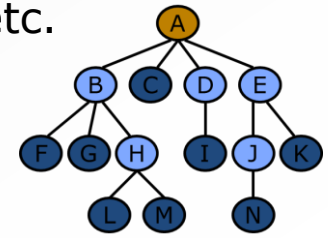
- A tree is composed of nodes
- Each node contains a value
- Types of nodes
  - **Root**: only one in a tree, has no parent.
  - Internal (non-leaf):
    Nodes with children are called **internal nodes**
  - Leaf:
    nodes without children are called **leaves**



Content Copyright Nanyang Technological University

WHY TREES?

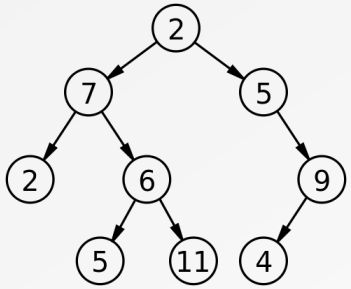
- Model layouts with hierarchical relationships between items
  - Chain of command in the army
  - Personnel structure in a company
  - (Binary tree structure is limited because each node can have **at most two children**)
- Tree structures also allow us to
  - Some problems require a tree structure: 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**



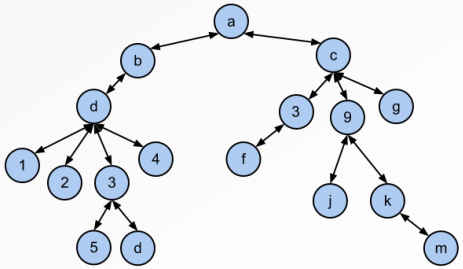
Content Copyright Nanyang Technological University

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



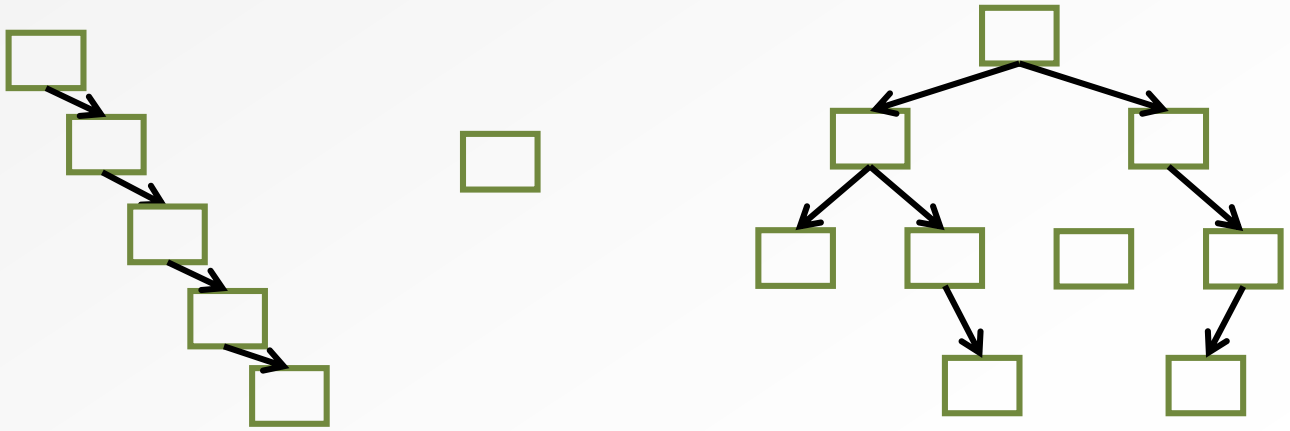
General tree

- 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**

Content Copyright Nanyang Technological University

POSSIBLE TREE CONFIGURATIONS

- We'll see later why not all trees configurations are desirable/useful
- Has to do with balance of a tree



Content Copyright Nanyang Technological University

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

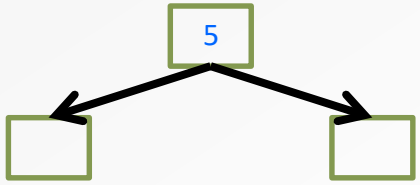
Content Copyright Nanyang Technological University

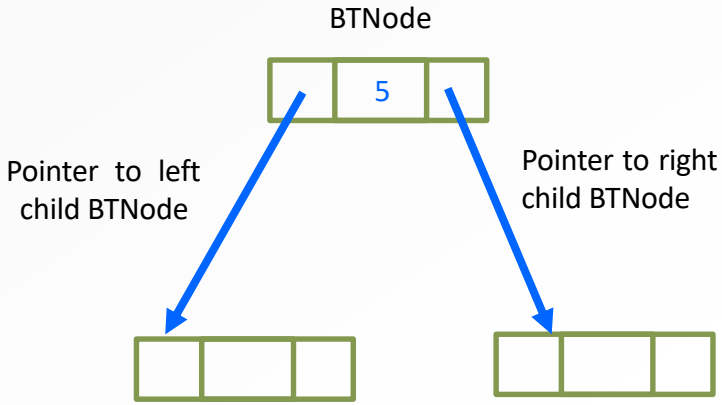
IMPLEMENTATION

- Recall implementation of LinkedList
  - Node has link to **at most <u>one</u>** other node
  - Defined a ListNode with one **`next`** pointer and a data **`item`**

```
typedef struct _listnode{
   int item;
   struct _listnode *next;
}ListNode;
```

- BinaryTree
  - Node has link to **at most <u>TWO</u>** other nodes
  - Define a BTNode with
    - Two pointers
    - A data item





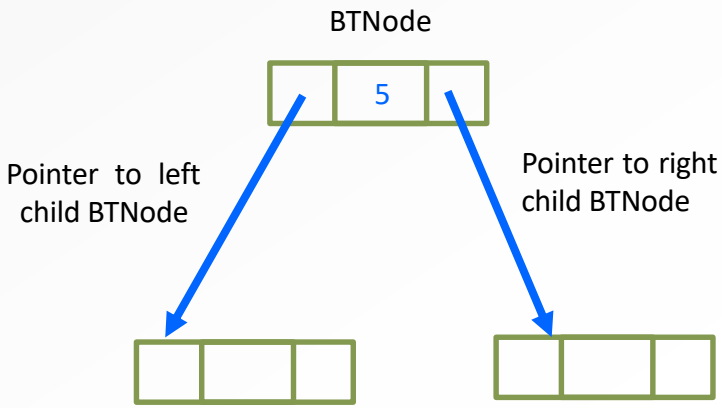
Content Copyright Nanyang Technological University

BTNode

- Start with a simple BTNode that stores an integer
  - The type of item can be character, string, or structure, etc.

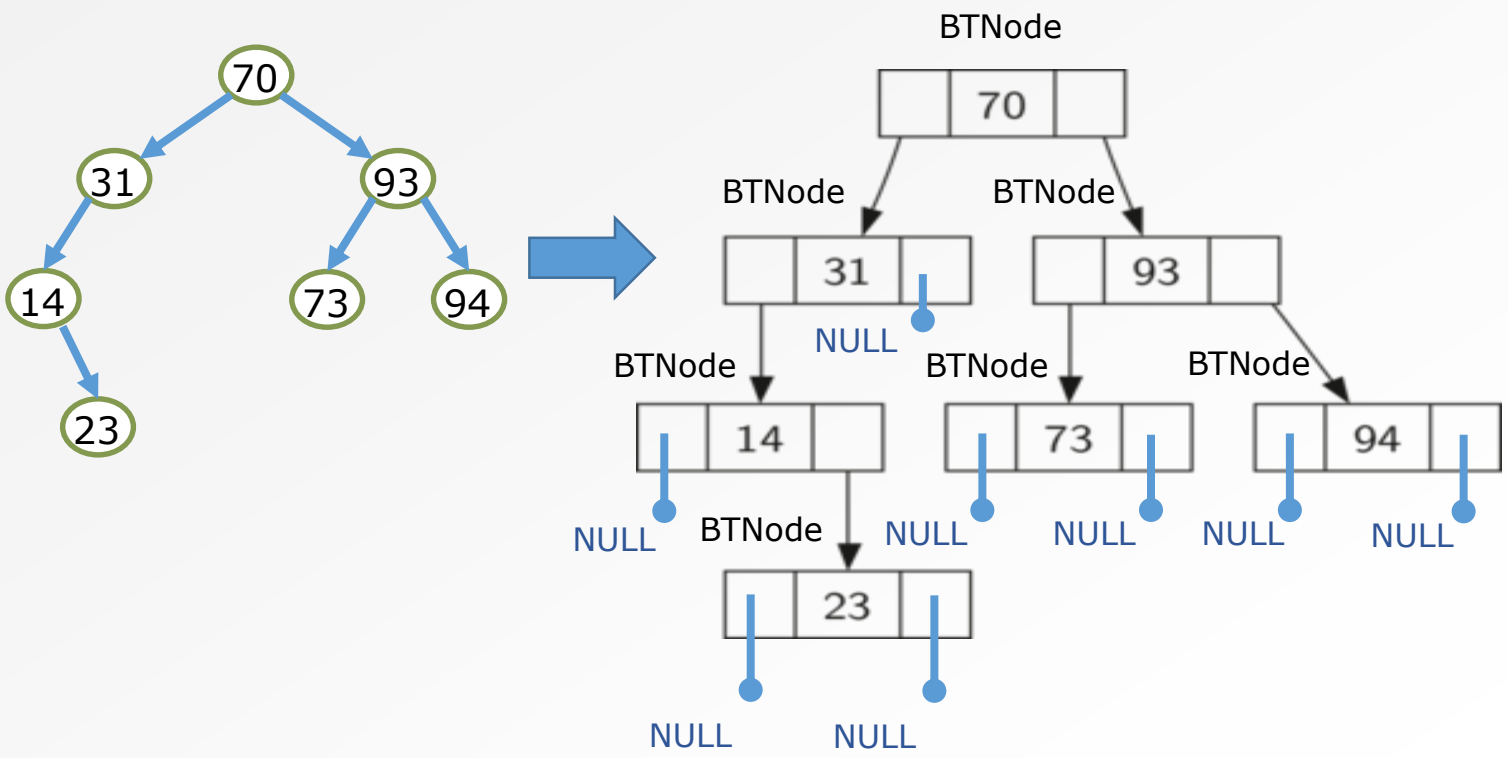
```
typedef struct _btnode{
    int item;
    struct _btnode *left;
    struct _btnode *right;
} BTNode;
```

```
typedef struct _listnode{
   int item;
   struct _listnode *next;
}ListNode;
```



Content Copyright Nanyang Technological University

EXAMPLE BINARY TREE



Content Copyright Nanyang Technological University

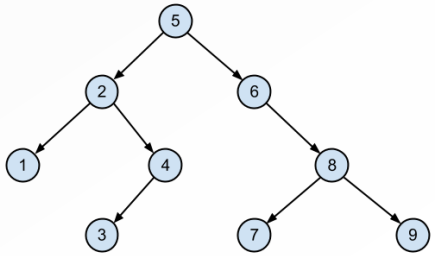
OUTLINE

- 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

Content Copyright Nanyang Technological University

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

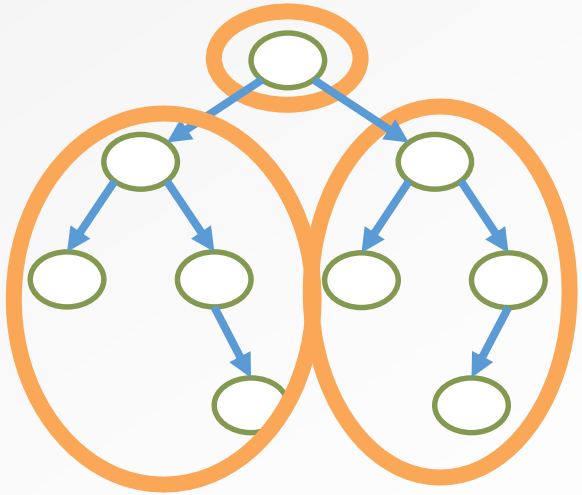


Content Copyright Nanyang Technological University

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



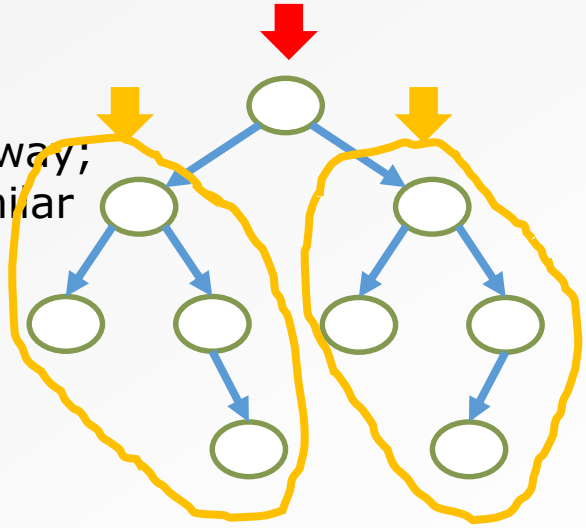
- Example task: count # of nodes in a tree

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

Content Copyright Nanyang Technological University

TREE TRAVERSAL

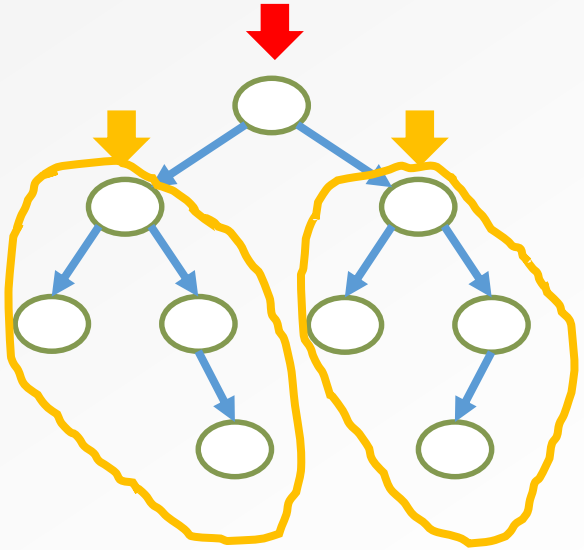
- Tree traversal is recursive
  - Recursion: is the process of repeating items in a self-similar way; divide a problem into several similar sub-problems.
  - At each node
    - Visit the node and both children
- Initial case + repeating case
  - (Visit root) + (visit children)
- When combined, guarantees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



Content Copyright Nanyang Technological University

TREE TRAVERSAL PROCESS

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



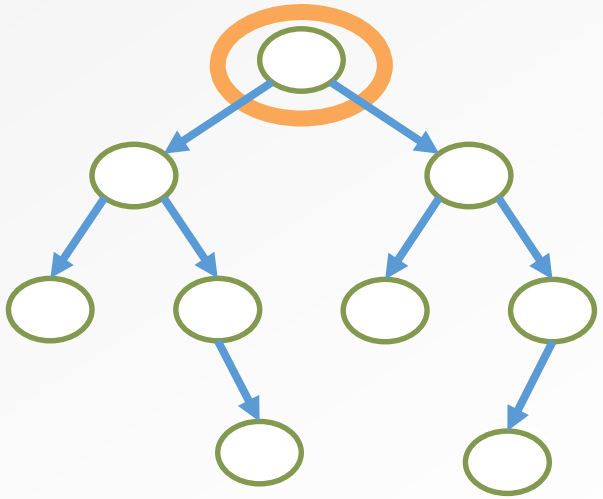
Content Copyright Nanyang Technological University

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
```

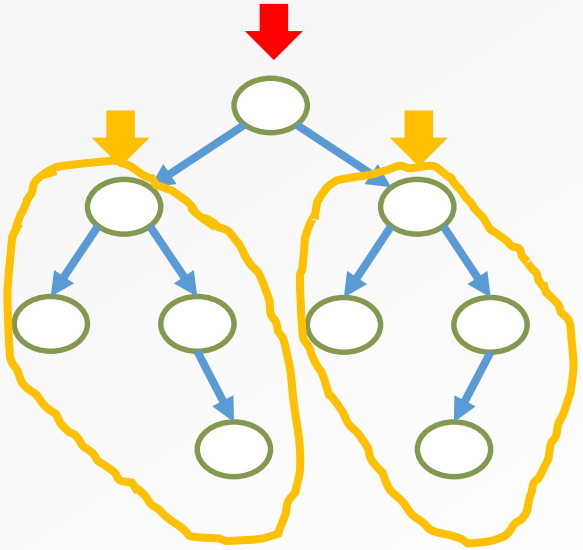
In main(), call TreeTraversal(root)



Content Copyright Nanyang Technological University

TREE TRAVERSAL TEMPLATE #2

- Current function:
  - Need to check for existence of left and right children before following them



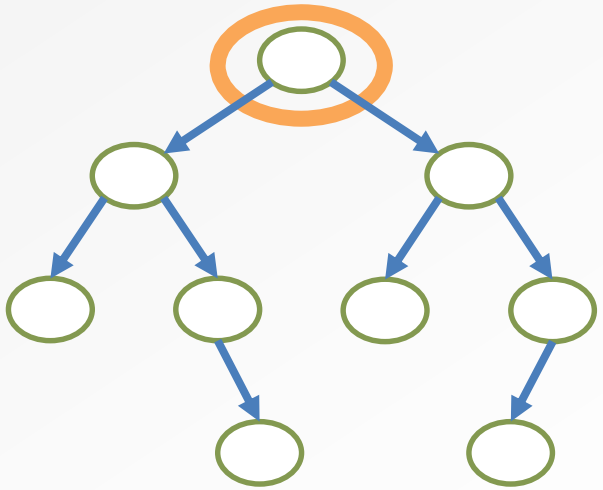
- New version:
  - Always follow links to children
  - Then check if the link is NULL
  - In other words, not actually pointing at a BTNode

Content Copyright Nanyang Technological University

TREE TRAVERSAL TEMPLATE #2

Pseudocode

```
TreeTraversal2(Node N):

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

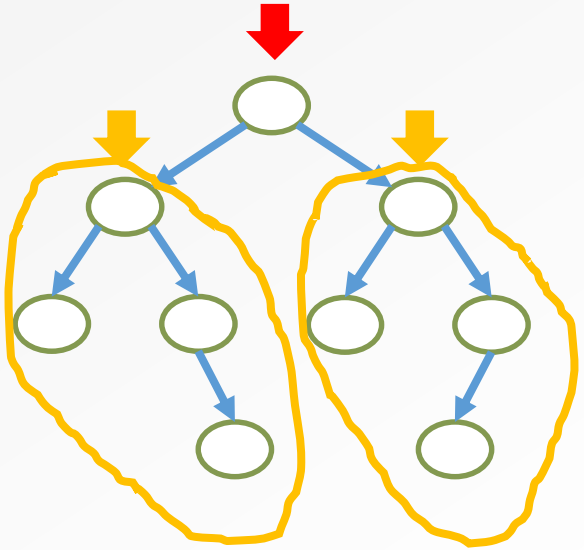


In main(), call TreeTraversal2(root)

Content Copyright Nanyang Technological University

TreeTraversal2() IMPLEMENTATION

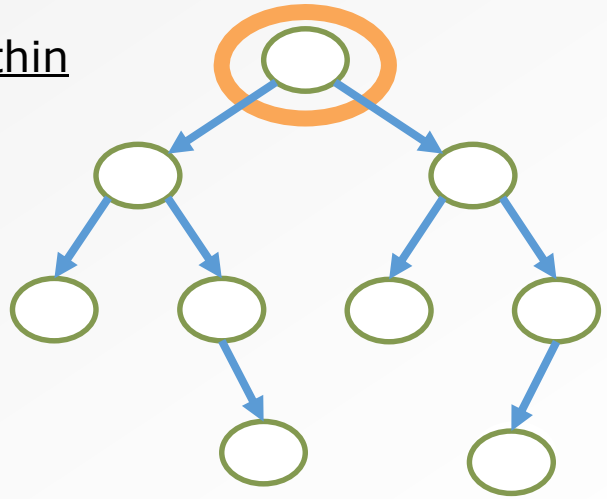
```
Void TreeTraversal2(BTNode *cur){
    If (cur == NULL) return;
    PrintNode(cur); // visit cur
    TreeTraversal2(cur->left);
    TreeTraversal2(cur->right);
}
```



Content Copyright Nanyang Technological University

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)

Content Copyright Nanyang Technological University

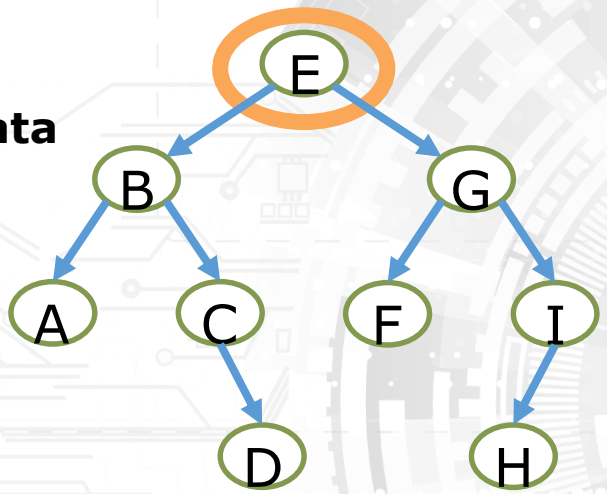
OUTLINE

- 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

Content Copyright Nanyang Technological University

THREE "STANDARD" WAYS TO TRAVERSAL

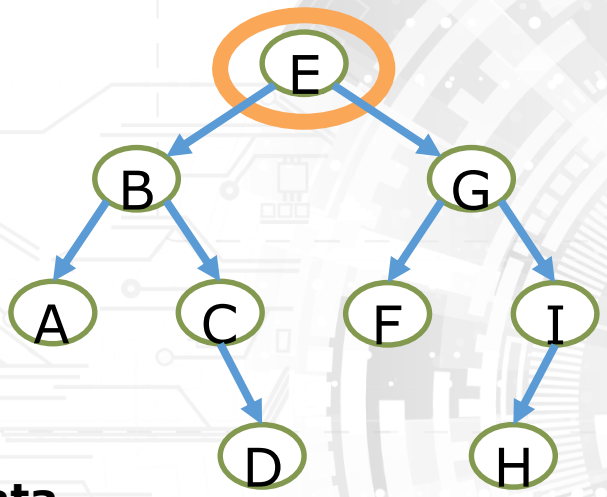
- Pre-order
  - **Process the current node's data**
  - **Visit the left child subtree**
  - **Visit the right child subtree**
- In-order
- Post-order



Content Copyright Nanyang Technological University

THREE "STANDARD" WAYS TO TRAVERSAL

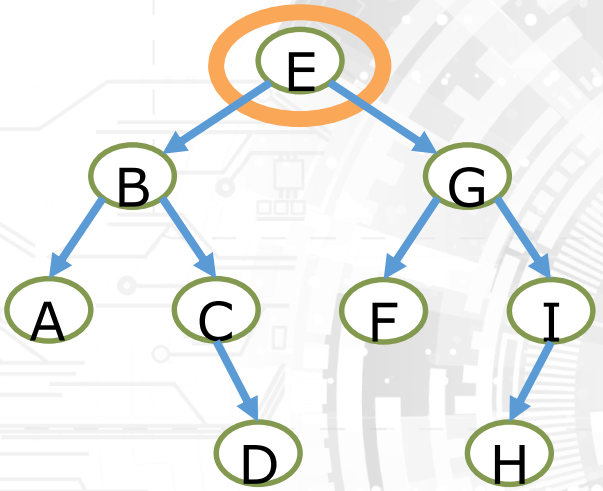
- Pre-order
  - Process the current node's data
  - Visit the left child subtree
  - Visit the right child subtree
- **In-order**
  - **Visit the left child subtree**
  - **Process the current node's data**
  - **Visit the right child subtree**
- Post-order



Content Copyright Nanyang Technological University

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**



Content Copyright Nanyang Technological University

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;

    // Do something with the current node's data

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

Content Copyright Nanyang Technological University

TREE TRAVERSAL - PRINT

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

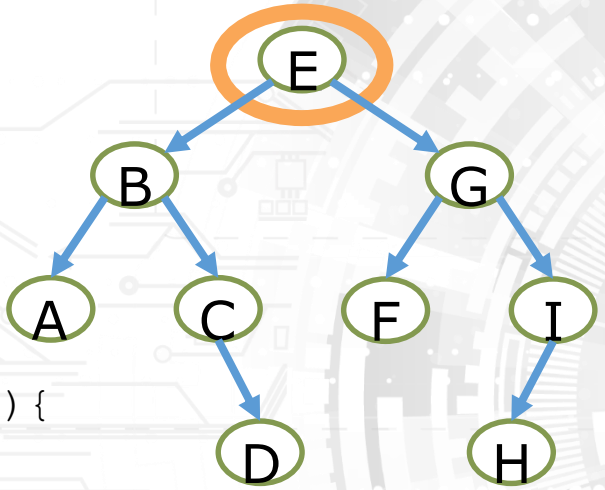
```
void TreeTraversal(BTNode *cur){
    if (cur == NULL)
        return;

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

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

Content Copyright Nanyang Technological University

TREE TRAVERSAL PRE-ORDER: PRINT

Output:

```
E B A C D G F I H
```



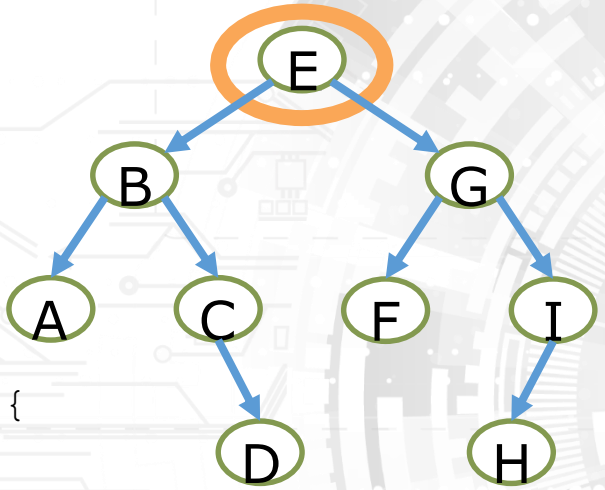

```
void TreeTraversal_pre(BTNode *cur){
    if (cur == NULL)
        return;

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

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

Content Copyright Nanyang Technological University

TREE TRAVERSAL IN-ORDER: PRINT

Output:

```
A B C D E F G H I
```



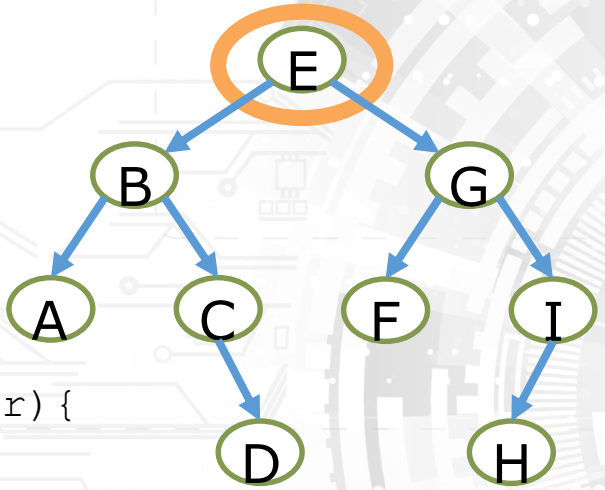

```
void TreeTraversal_in(BTNode *cur){
    if (cur == NULL)
        return;

    TreeTraversal_in(cur->left); //Visit the left child node
    printf("%c  ",cur->item);
    TreeTraversal_in(cur->right);//Visit the right child node
}
```

Content Copyright Nanyang Technological University

TREE TRAVERSAL POST-ORDER: PRINT

Output:

```
A D C B F H I G E
```




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

    TreeTraversal_post(cur->left); //Visit the left child node
    TreeTraversal_post(cur->right);//Visit the right child node
    printf("%c  ",cur->item);
}
```

Content Copyright Nanyang Technological University

PRE-ORDER, IN-ORDER AND POST-ORDER

Pre-Order Traversal

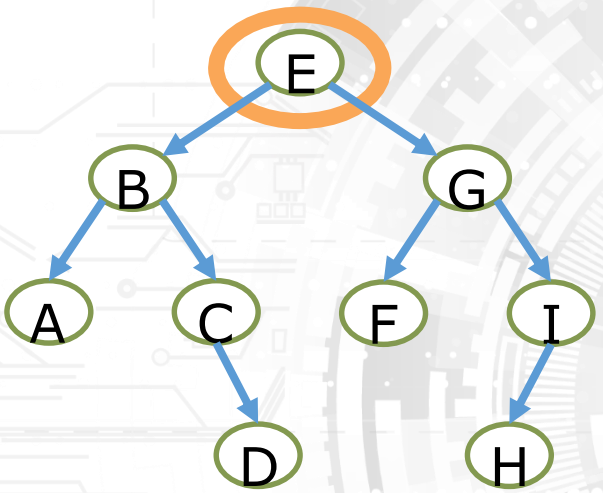
E B A C D G F I H

In-Order Traversal

A B C D E F G H I

Post-Order Traversal

A D C B F H I G E



Content Copyright Nanyang Technological University

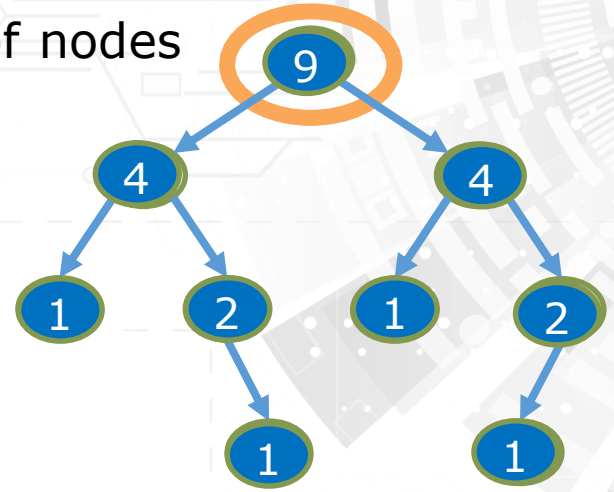
OUTLINE

- 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

Content Copyright Nanyang Technological University

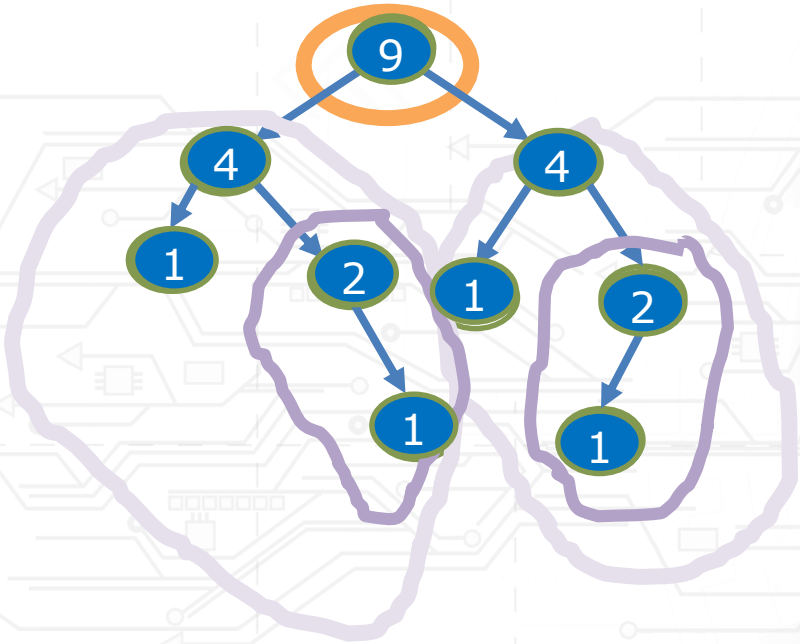
COUNT NODES IN A BINARY TREE

- 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



Content Copyright Nanyang Technological University

COUNT NODES IN A BINARY TREE

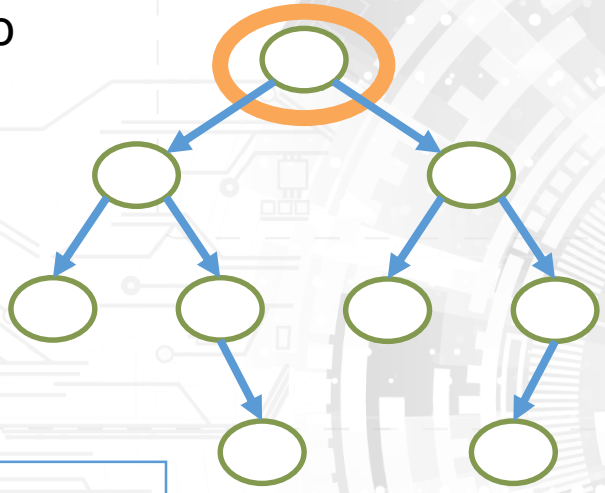


- Each node returns the number of nodes in its own subtree
- Leaf nodes return 1
  Information **propagates upwards** as TreeTraversal returns from visiting leaf nodes
- Which is the first/last count to be returned?

Content Copyright Nanyang Technological University

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

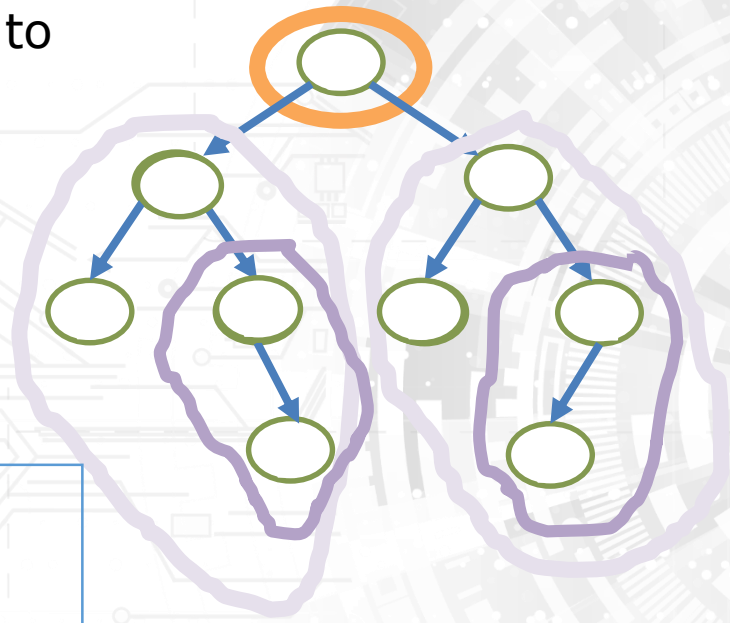


```
void TreeTraversal(BTNode *cur){
    if (cur == NULL)
       return;
    //may do something with cur;
    TreeTraversal(cur->left);
    TreeTraversal(cur->right);
    //may do something with cur;
}
```

Content Copyright Nanyang Technological University

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

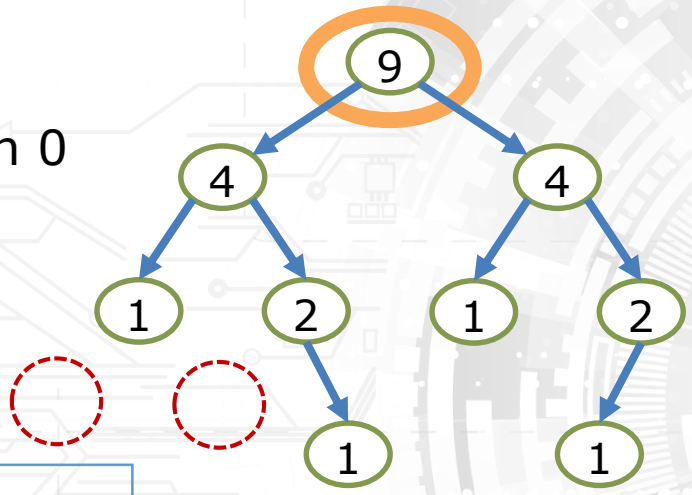


```
int countNode(BTNode *cur){
    if (cur == NULL)
        return ???;

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

Content Copyright Nanyang Technological University

countNode()

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

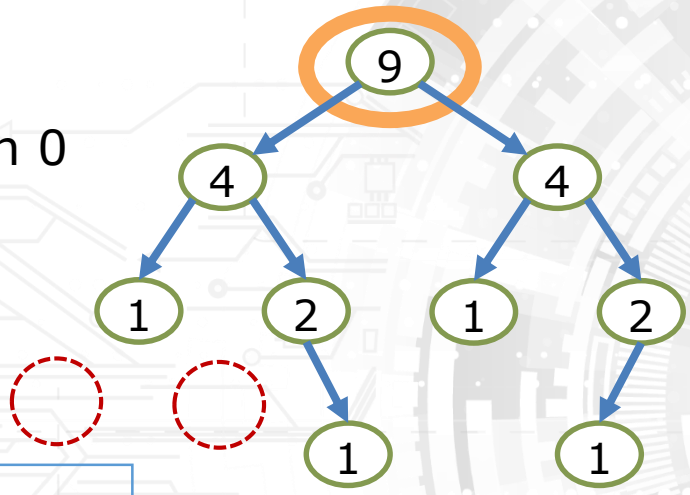



```
int countNode(BTNode *cur){
    if (cur == NULL)
        return 0;

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

Content Copyright Nanyang Technological University

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);
}
```

Content Copyright Nanyang Technological University

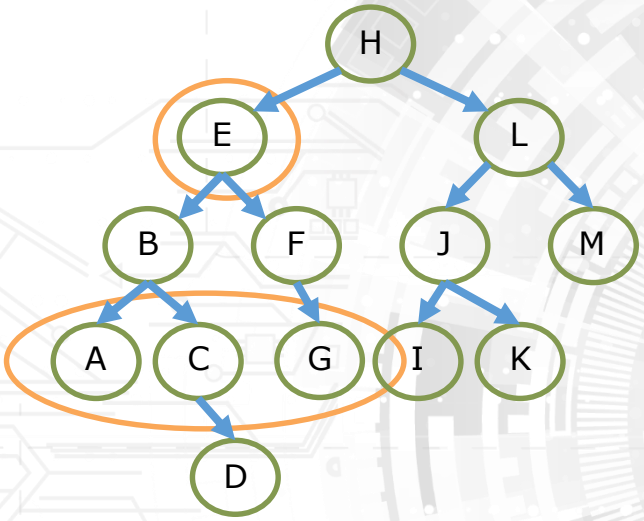
OUTLINE

- 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

Content Copyright Nanyang Technological University

FIND GRANDCHILDREN

- Given a node X, find all the nodes that are X's grandchildren
- Given node E, we should return grandchild nodes A, C, and G
- What if we want to find k-level grandchildren?
  - **Need a way to keep track of how many levels down we've gone**



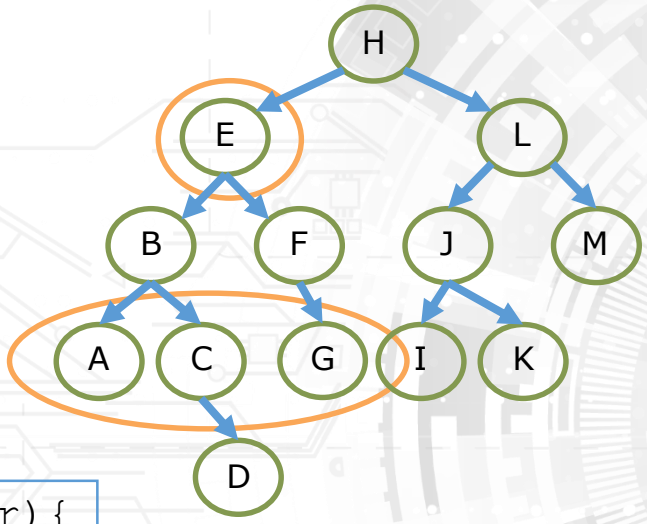
X->left->left
X->left->right
X->right->left
X->right->right

2-level grandchildren

Content Copyright Nanyang Technological University

FIND GRANDCHILDREN

- We want to go down k "levels"
- Use a counter to track how far down we've gone
- At each TreeTraversal(child), increment counter




```
void TreeTraversal(BTNode *cur){

    if (cur == NULL)
        return;

    // check counter

    TreeTraversal(cur->left);
    TreeTraversal(cur->right);
}
```

Do something with the current node's data

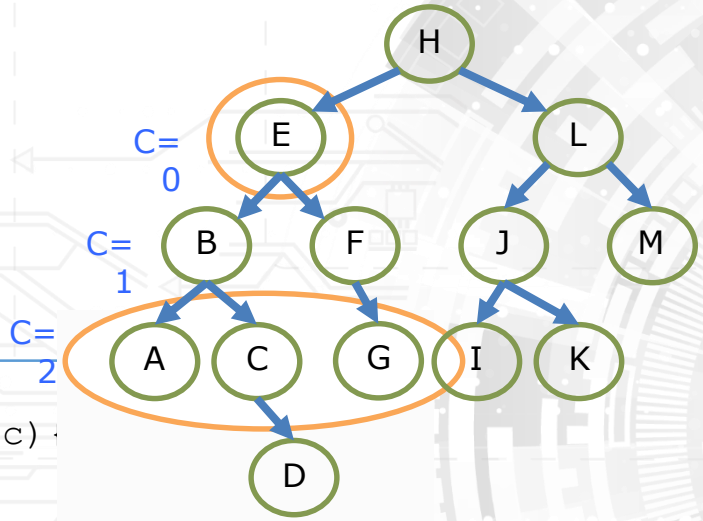
Visit the left child node

Visit the right child node

Content Copyright Nanyang Technological University

FIND GRANDCHILDREN

```
void main( ){ …
    if (X = null)return;
    findgrandchildren(X,0);
}
```

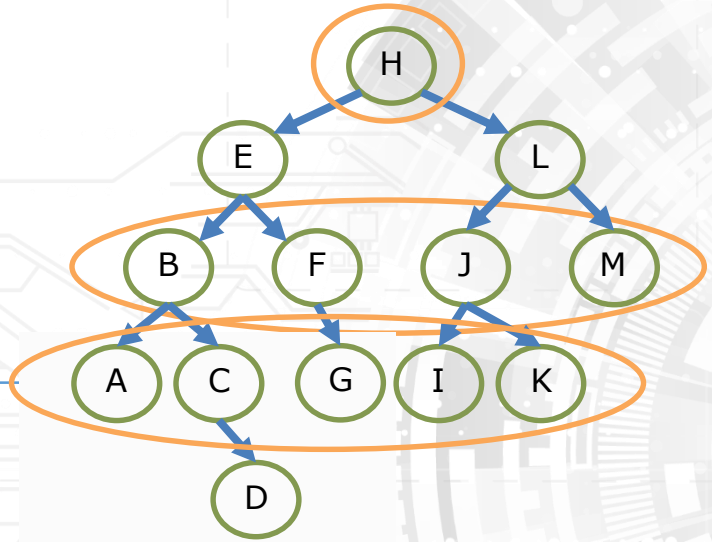



```
1.  void findgrandchildren(
                BTNode *cur, int c){
2.      if (cur == NULL) return;

3.      if (c == k){
4.         printf("%d ", cur->item);
5.         return;
6.      }
7.      if (c < k){
8.         findgrandchildren(cur->left, c+1);
9.         findgrandchildren(cur->right, c+1);
10.}
```

Content Copyright Nanyang Technological University

FIND GRANDCHILDREN

```
void main( ){ …
    if (X = null)return;
    findgrandchildren(X,0);
}
```




```
void findgrandchildren(
              BTNode *cur, int c){
    if (cur == NULL) return;

    if (c == k){
       printf("%d ", cur->item);
       return;
    }
    if (c < k){
       findgrandchildren(cur->left, c+1);
       findgrandchildren(cur->right, c+1); }
}
```

if k=2, we call findgrandchildren(H,0), what is the output?

How about k=3?

How about findgrandchildren(H,1)?

Content Copyright Nanyang Technological University

OUTLINE

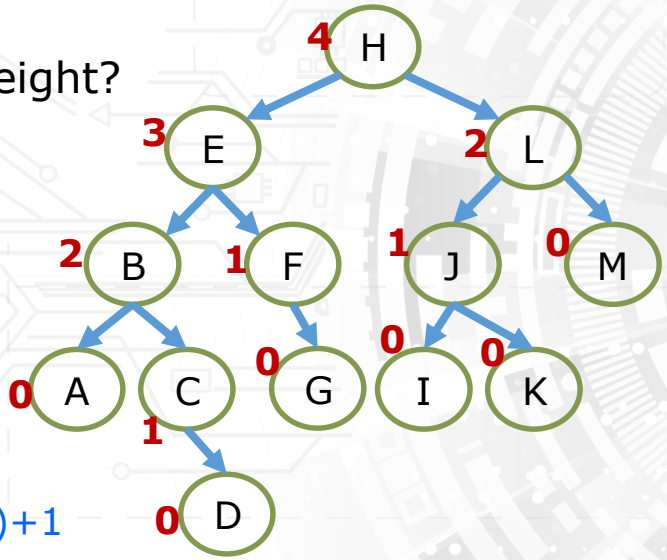
- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary Tree Traversal
- Tree traversal order
  - Pre-order
  - In-order
  - Post-order
- Application examples
  - Count nodes in a binary tree
  - Find grandchild nodes
  - **Calculate height of every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

Content Copyright Nanyang Technological University

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?
- We found:
  - leaf.height= 0
  - Non-leaf node X

X.height=max(X.left.height, X.right.height)+1

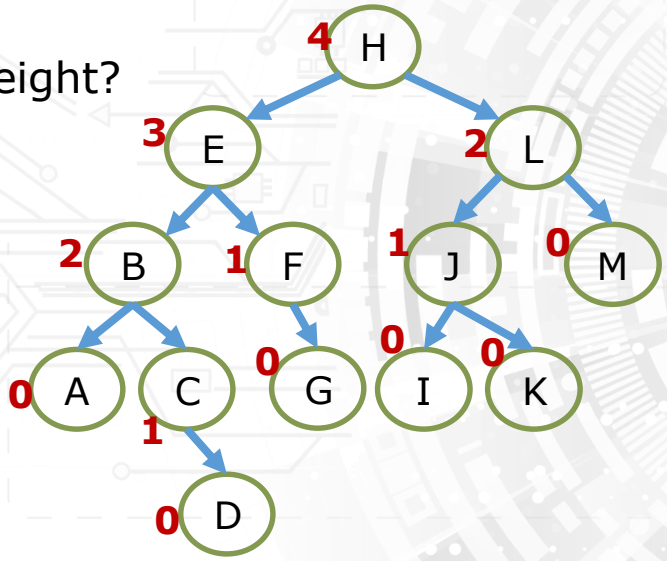


- Does information propagate upwards or downwards?

Content Copyright Nanyang Technological University

CALCULATE HEIGHT OF EVERY NODE

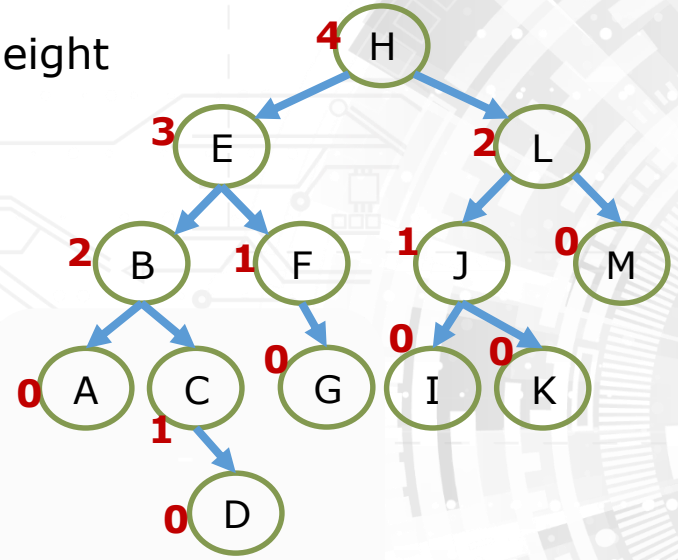
- Height of a node = number of links from that node to the deepest leaf node
- How does each node calculate its height?
  - What is the height of node D, C, H?
- Go through entire tree: calculate and store height of each node in the item field



Content Copyright Nanyang Technological University

CALCULATE HEIGHT OF EVERY NODE

- We want each node to report its height
  - Leaf node must report 0

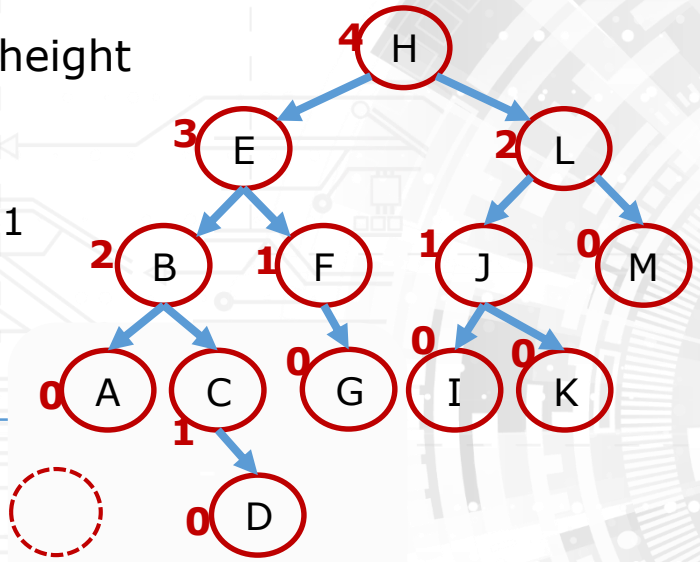



```
int TreeTraversal(BTNode *cur){

    if(cur == NULL)
       return   ;

    int l = TreeTraversal(cur->left);
    int r = TreeTraversal(cur->right);

    // do something here. Max( left, right)?

    return  ;

}
```

Content Copyright Nanyang Technological University

CALCULATE HEIGHT OF EVERY NODE

- We want each node to report its height
  - Leaf node must report 0
  - At "null" condition, must report -1

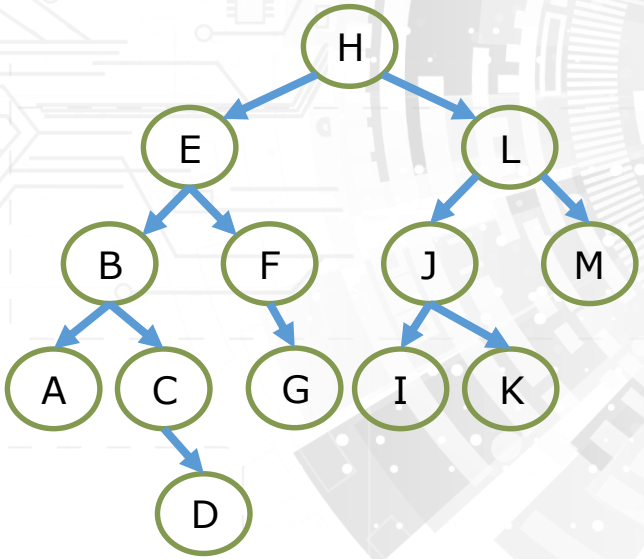



```
int TreeTraversal(BTNode *cur){

    if(cur == NULL)
       return -1;

    int l = TreeTraversal(cur->left);
    int r = TreeTraversal(cur->right);

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

    return c;

}
```

Content Copyright Nanyang Technological University

QUESTIONS

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

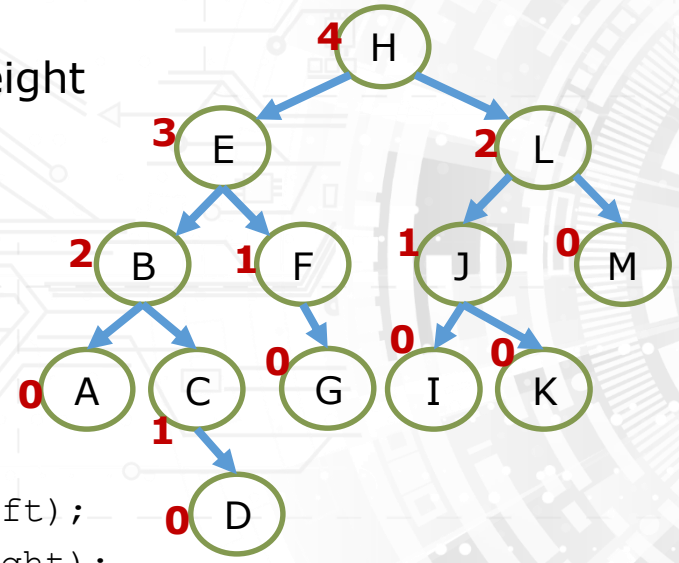


Content Copyright Nanyang Technological University

CALCULATE HEIGHT OF EVERY NODE

- Height of a node = number of links from that node to the deepest leaf node
- We want each node to report its height
  - Leaf node must report 0
  - At "null" condition, must report -1

```
int TreeTraversal(BTNode *cur){
    if(cur == NULL)
        return -1;

    int l = TreeTraversal(cur->left);
    int r = TreeTraversal(cur->right);

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

    return c;
}
```

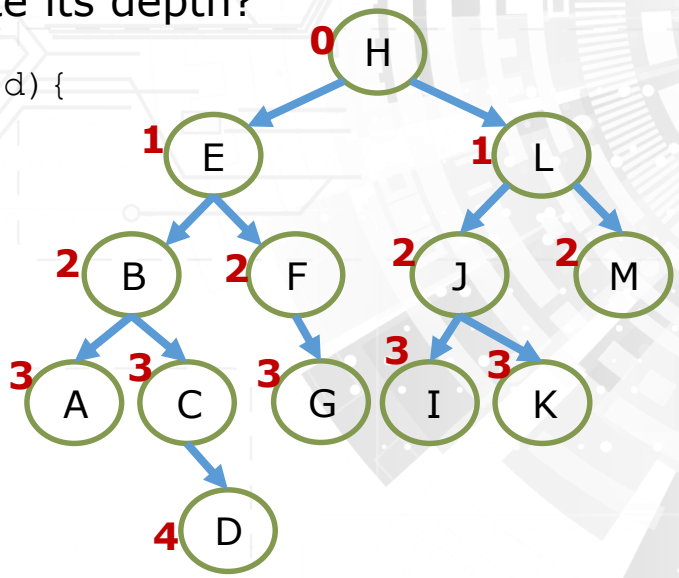


Content Copyright Nanyang Technological University

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 ;

}
```



Content Copyright Nanyang Technological University

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

Content Copyright Nanyang Technological University

LEVEL-BY-LEVEL: BREADTH-FIRST SEARCH

Depth-first search

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

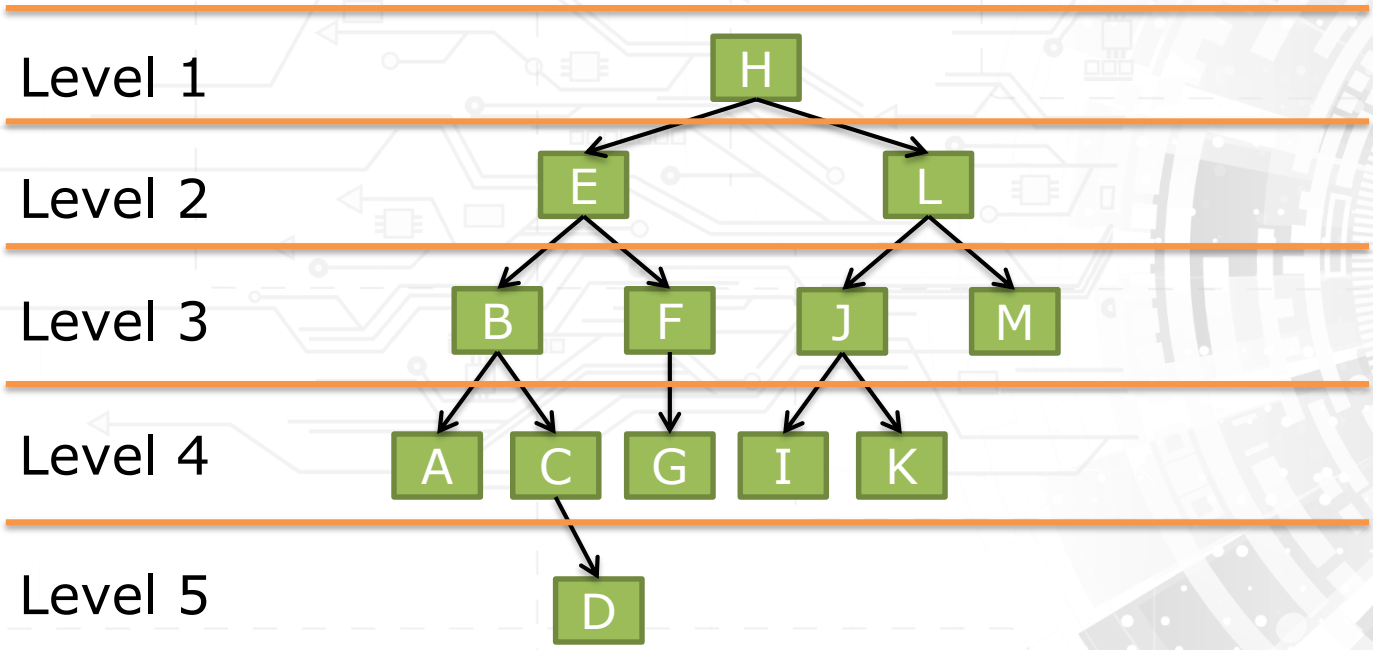
E.g. the post-order traversal

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.

Content Copyright Nanyang Technological University

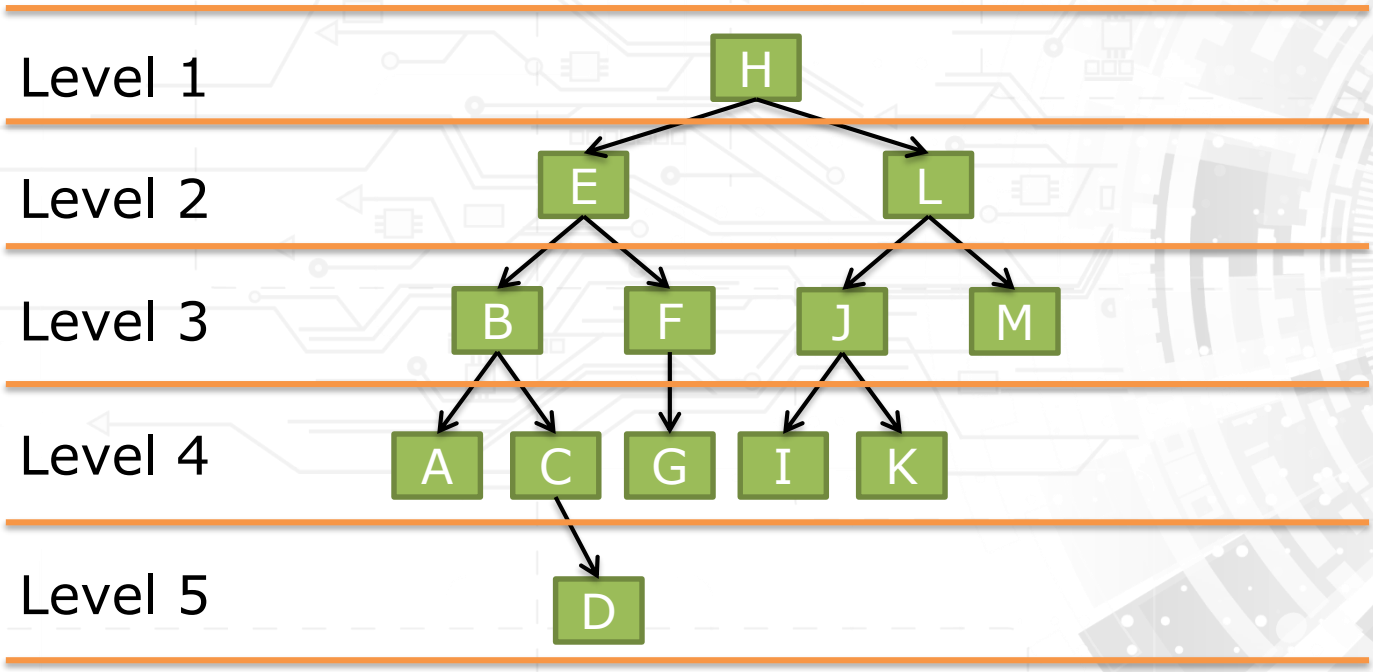
LEVEL-BY-LEVEL TREE TRAVERSAL



Content Copyright Nanyang Technological University

LEVEL-BY-LEVEL TREE TRAVERSAL

- Hint: Make use of another data structure

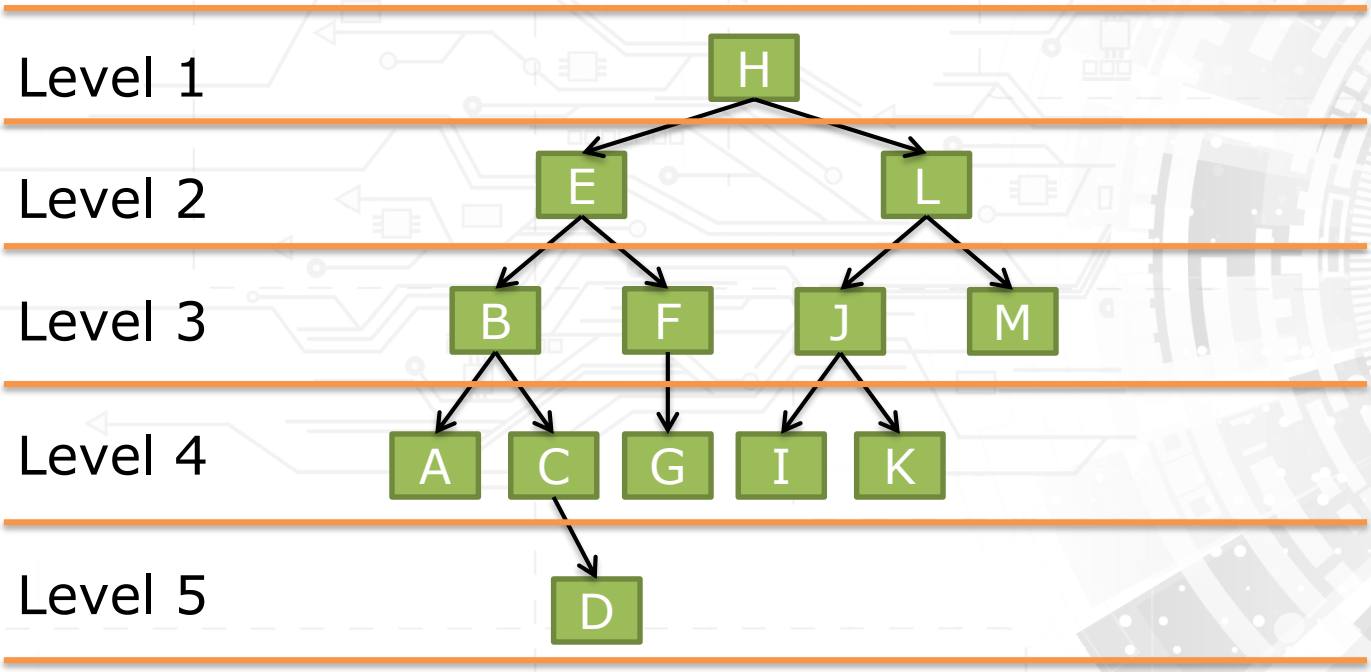


Nodes stored in order accessed in tree…

Content Copyright Nanyang Technological University

LEVEL-BY-LEVEL TREE TRAVERSAL

- Use a queue! Root node should be first

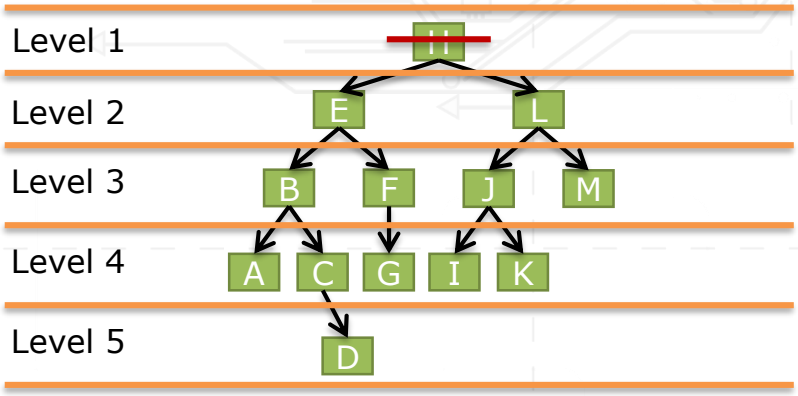


Nodes stored in order accessed in tree

Content Copyright Nanyang Technological University

LEVEL-BY-LEVEL TREE TRAVERSAL

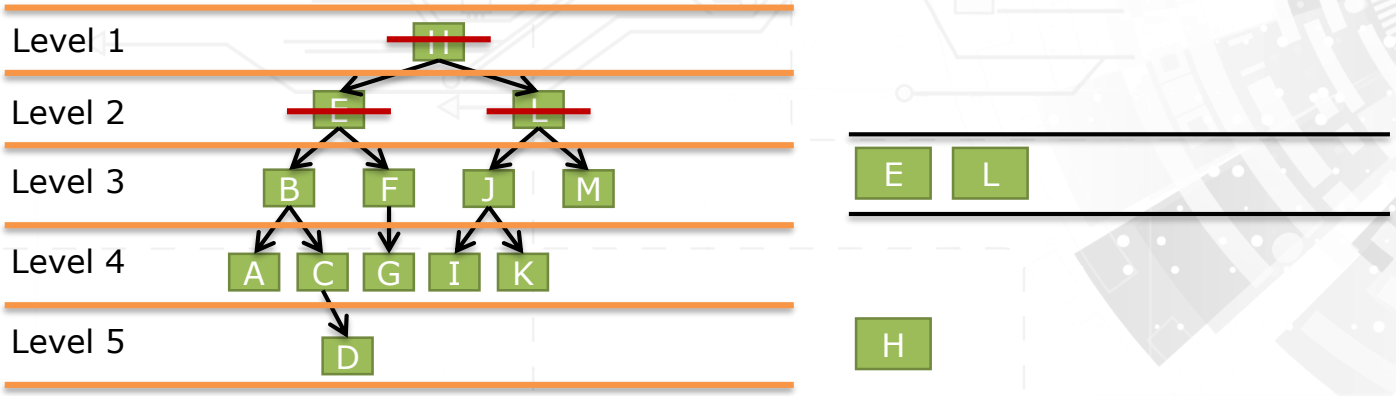
- Enqueue the root, H



Content Copyright Nanyang Technological University

LEVEL-BY-LEVEL TREE TRAVERSAL

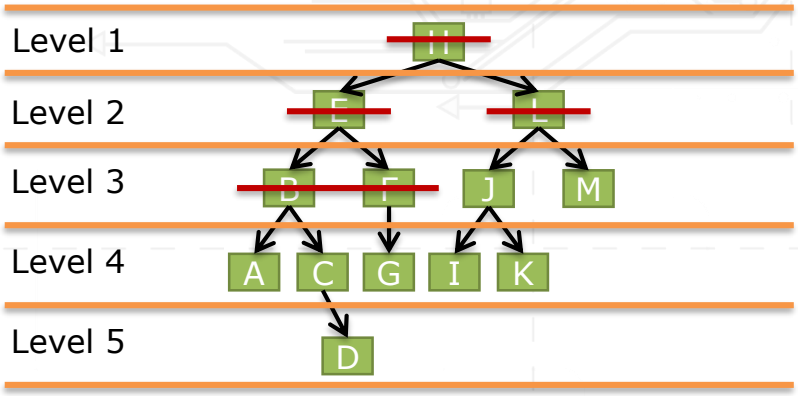
- Enqueue the root, H
- Dequeue H, and enqueue H's children

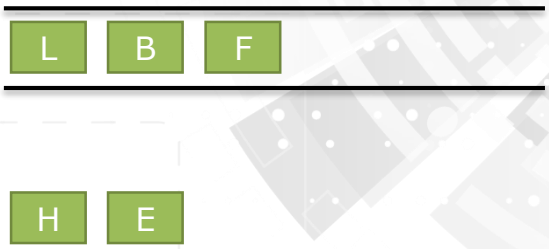


Content Copyright Nanyang Technological University

LEVEL-BY-LEVEL TREE TRAVERSAL

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

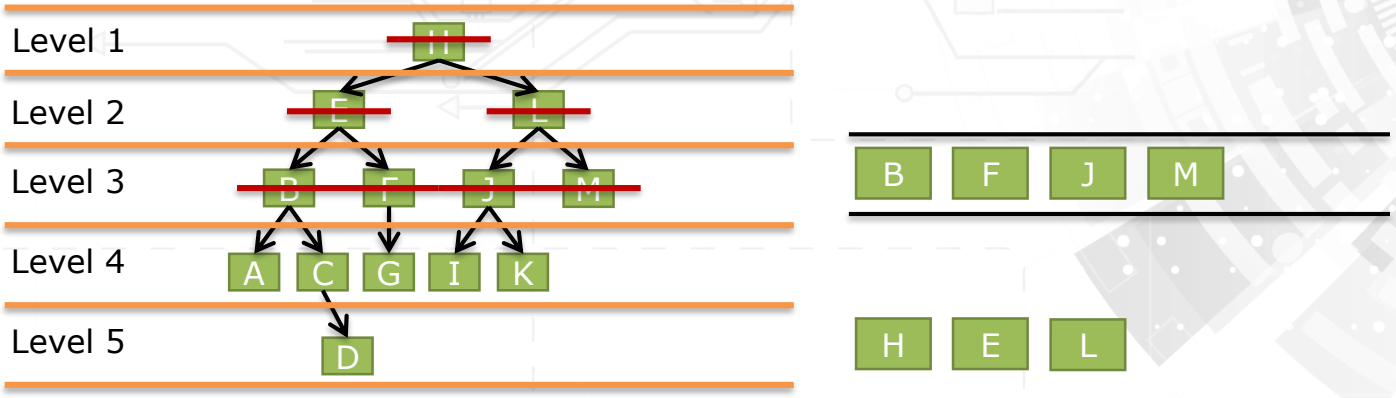




Content Copyright Nanyang Technological University

LEVEL-BY-LEVEL TREE TRAVERSAL

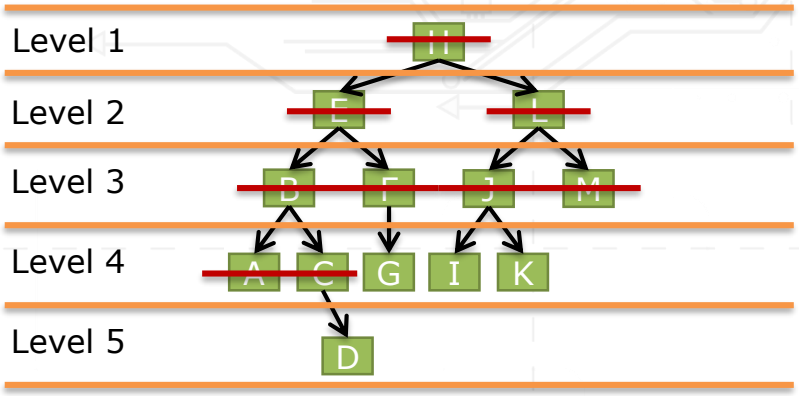
- Enqueue the root, H
- Dequeue H, and enqueue H's children
- Dequeue E, and enqueue E's children
- Dequeue L, and enqueue L's children

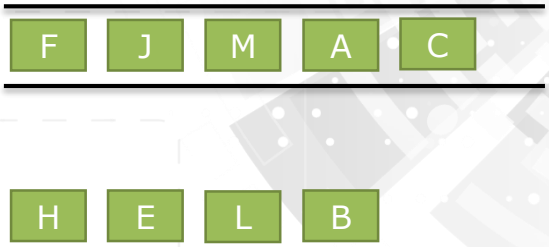


Content Copyright Nanyang Technological University

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
- Dequeue B, and enqueue B's children





Content Copyright Nanyang Technological University

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**

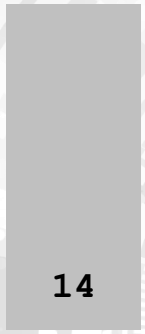
Content Copyright Nanyang Technological University

PREORDER TRAVERSAL WITH A STACK

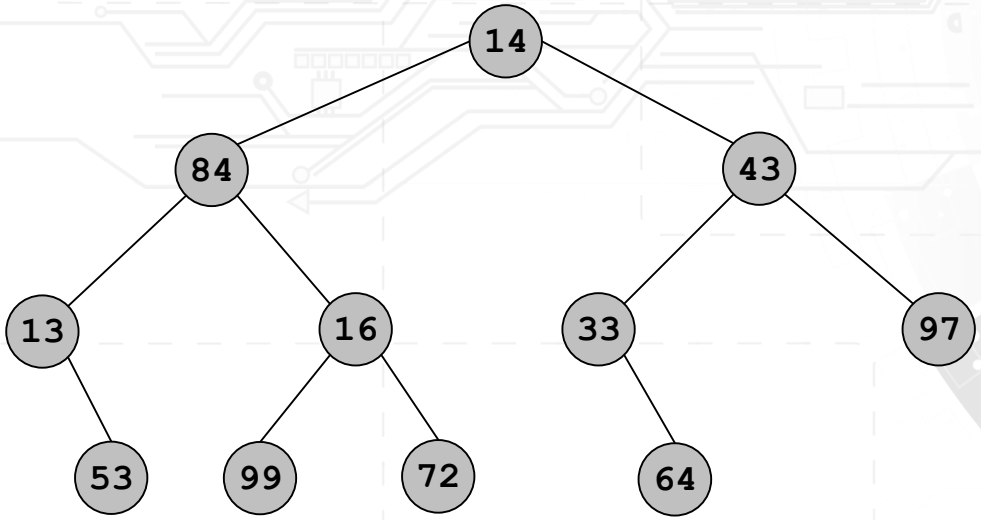
Push the root onto the stack.

While the stack is not empty

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



Stack



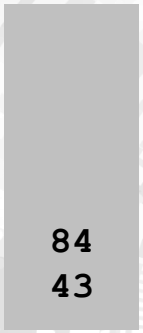
Content Copyright Nanyang Technological University

PREORDER TRAVERSAL WITH A STACK

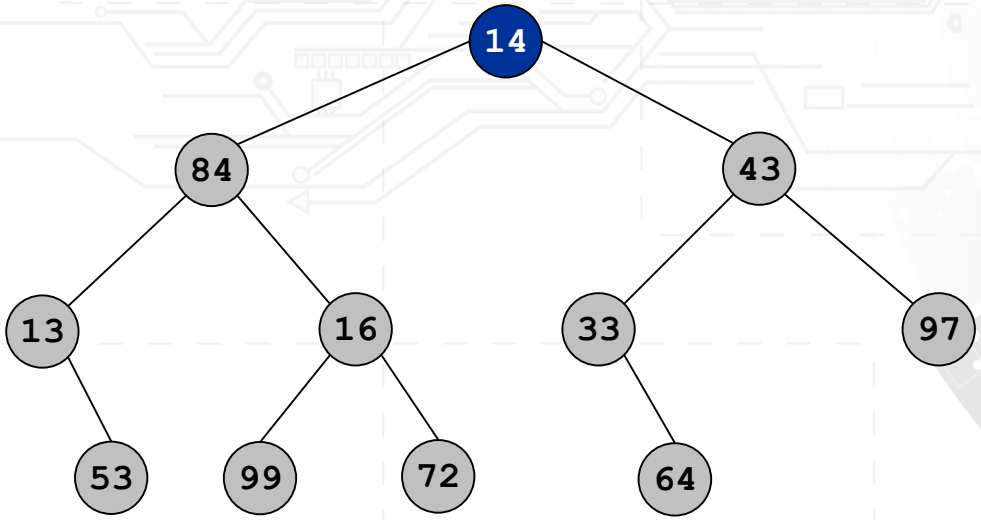
Push the root onto the stack.

While the stack is not empty

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



Stack



Content Copyright Nanyang Technological University

PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

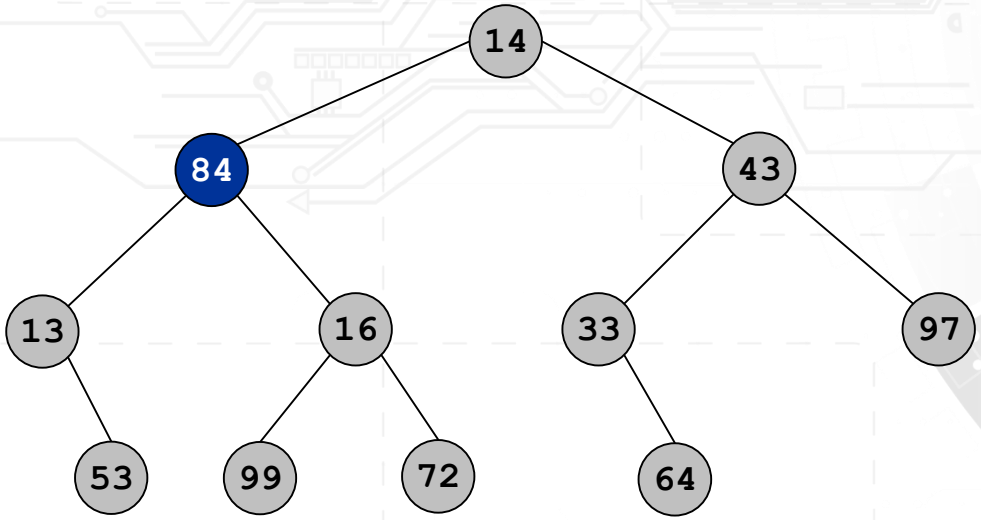
While the stack is not empty

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

```
14 84
```

```
13
16
43
```

Stack



Content Copyright Nanyang Technological University

PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

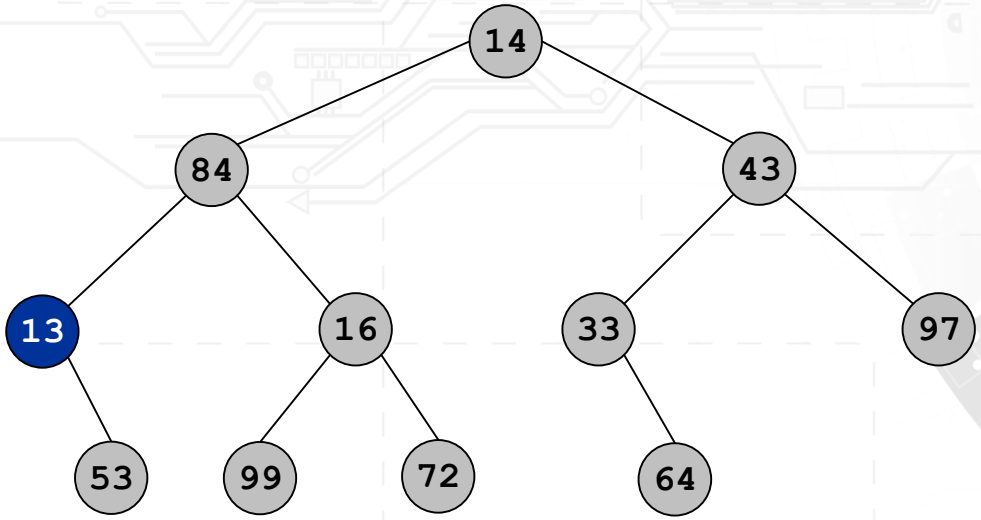
While the stack is not empty

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

```
14 84 13
```

```
53
16
43
```

Stack



Content Copyright Nanyang Technological University

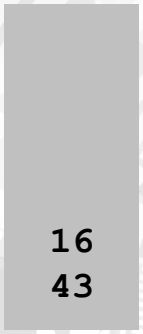
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

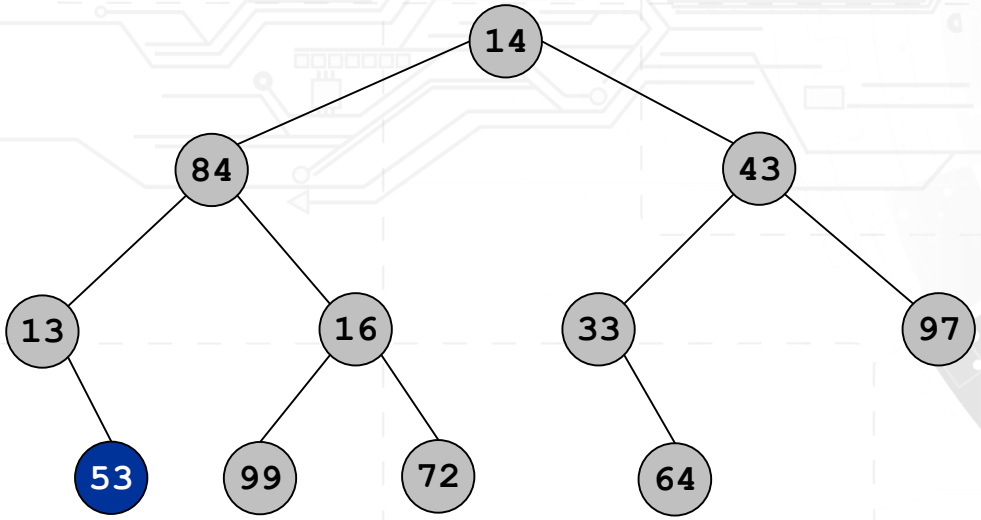
While the stack is not empty

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

```
14 84 13 53
```



Stack



Content Copyright Nanyang Technological University

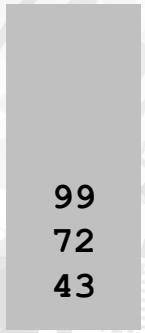
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

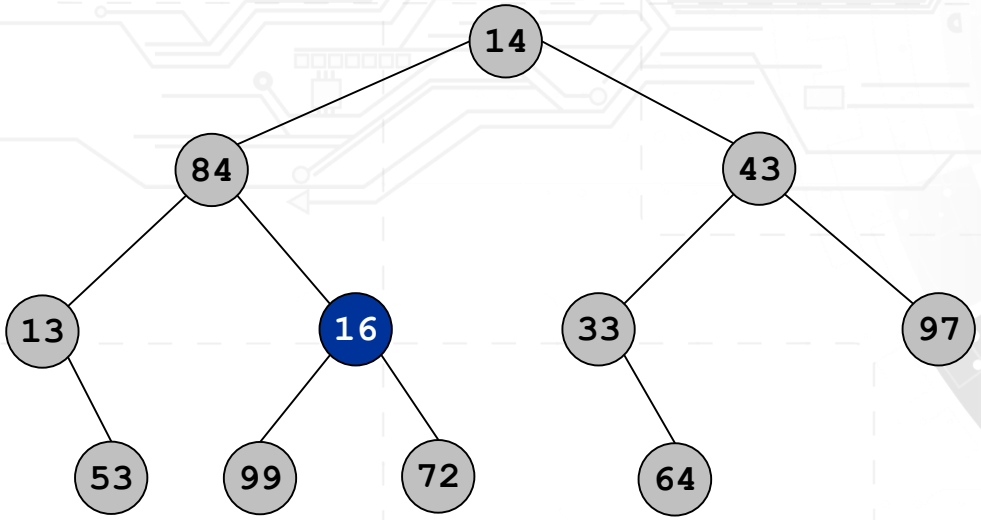
While the stack is not empty

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

```
14 84 13 53 16
```



Stack



Content Copyright Nanyang Technological University

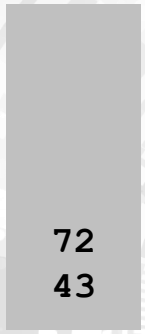
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

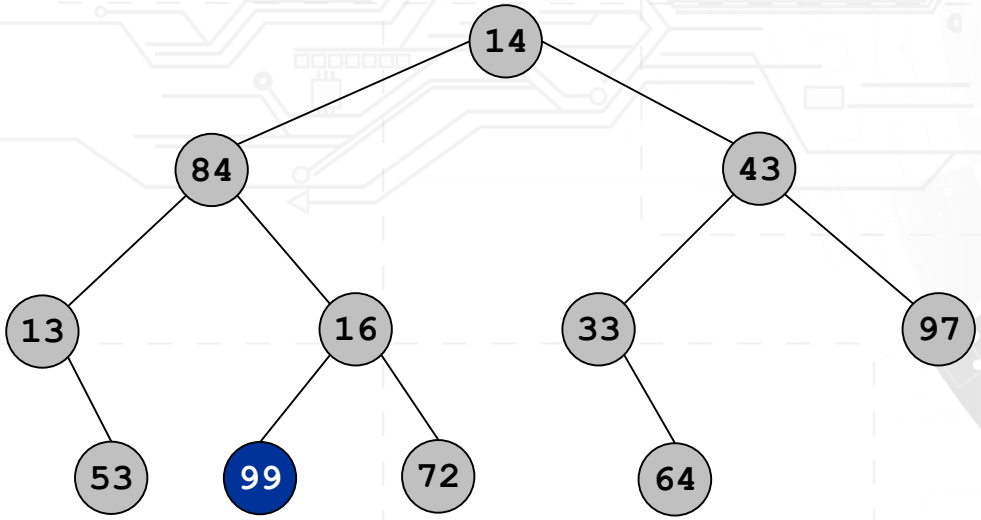
While the stack is not empty

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

```
14 84 13 53 16 99
```



Stack



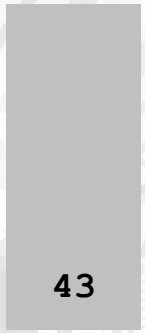
Content Copyright Nanyang Technological University

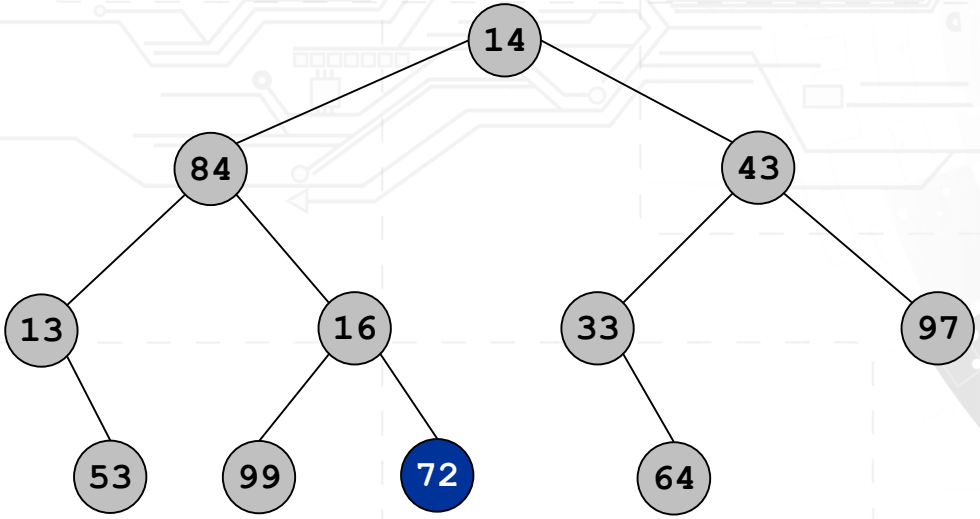
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

While the stack is not empty

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





Content Copyright Nanyang Technological University

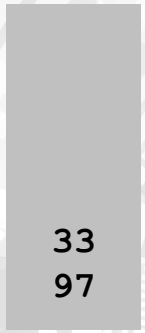
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

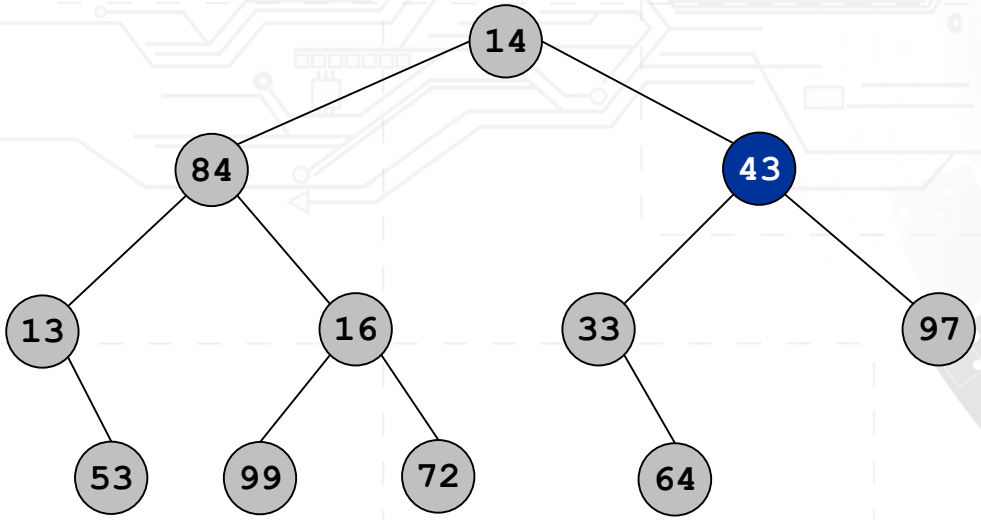
While the stack is not empty

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

```
14 84 13 53 16 99 72 43
```



Stack



Content Copyright Nanyang Technological University

PREORDER TRAVERSAL WITH A STACK

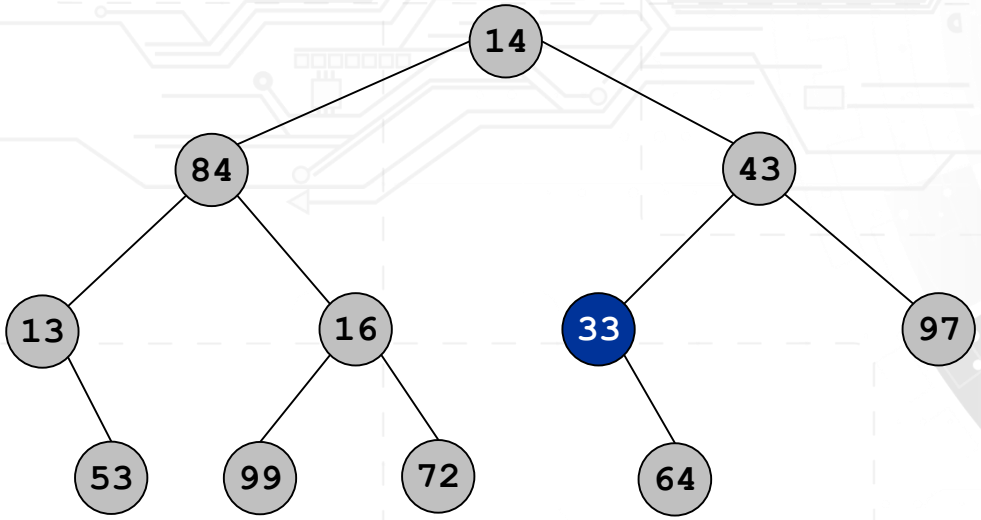
Push the root onto the stack.

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
```

Stack



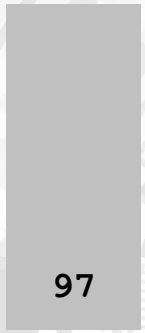
Content Copyright Nanyang Technological University

PREORDER TRAVERSAL WITH A STACK

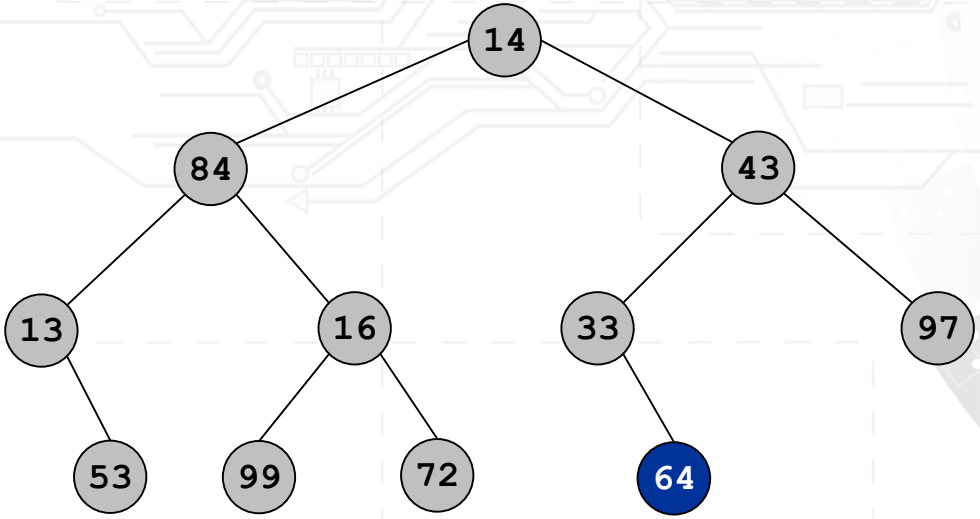
Push the root onto the stack.

While the stack is not empty

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



Stack



Content Copyright Nanyang Technological University

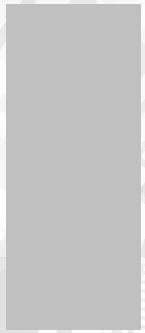
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

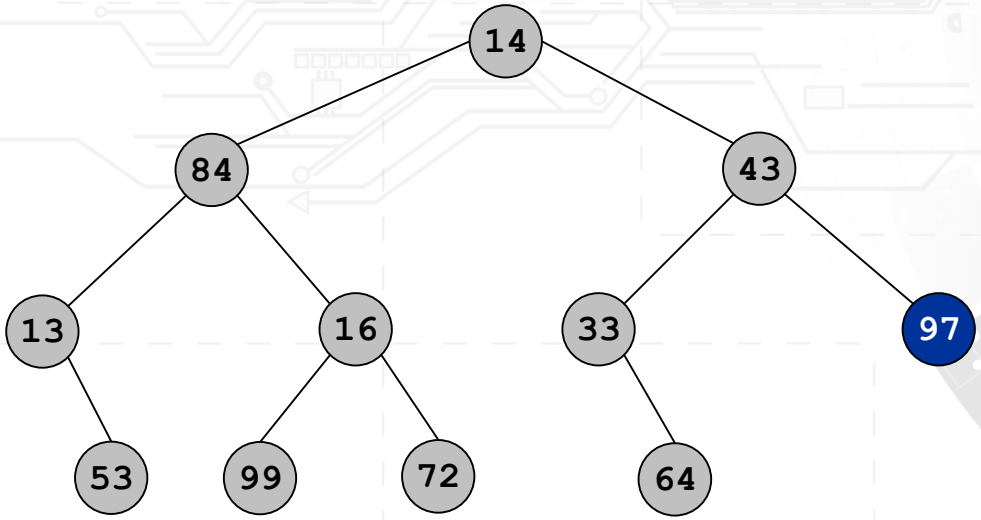
While the stack is not empty

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

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



Stack



Content Copyright Nanyang Technological University

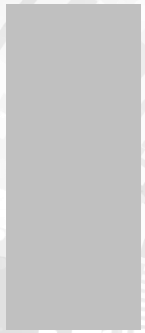
PREORDER TRAVERSAL WITH A STACK

Push the root onto the stack.

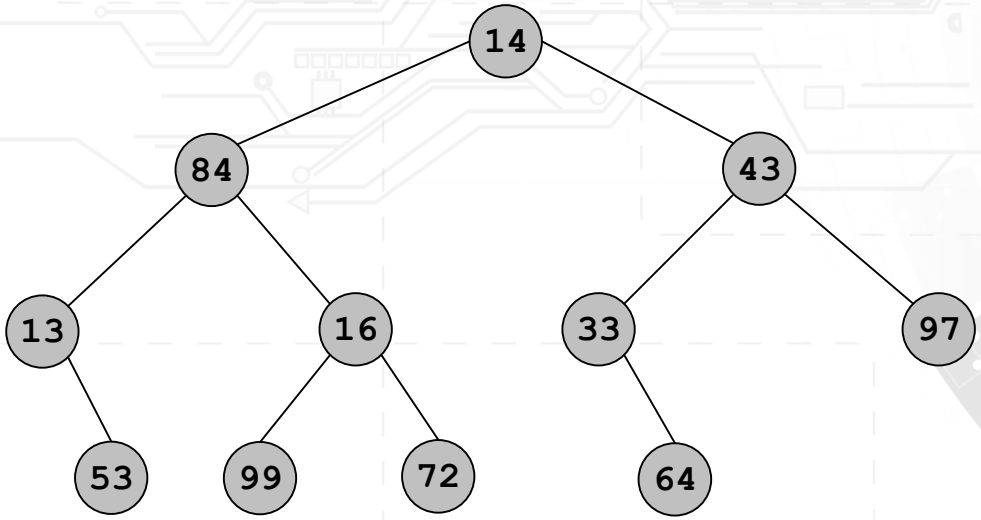
While the stack is not empty

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

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



Stack



Content Copyright Nanyang Technological University

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.

Content Copyright Nanyang Technological University