NANYANG TECHNOLOGICAL UNIVERSITY

CE1007/CZ1007 DATA STRUCTURES

Lecture 07: **Binary Trees**

Dr. Owen Noel Newton Fernando

College of Engineering

School of Computer Science and Engineering

OUTLINE

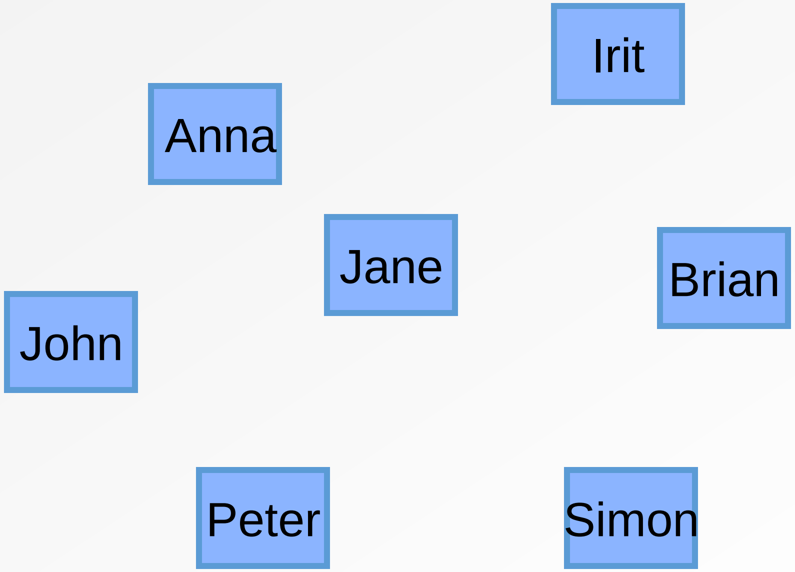
- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary tree traversal
- Example application

OUTLINE

- **Non-linear data structures**
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary tree traversal
- Example application

RECALL

- Suppose you have a set of names



Irit

Anna

Jane

Brian

John

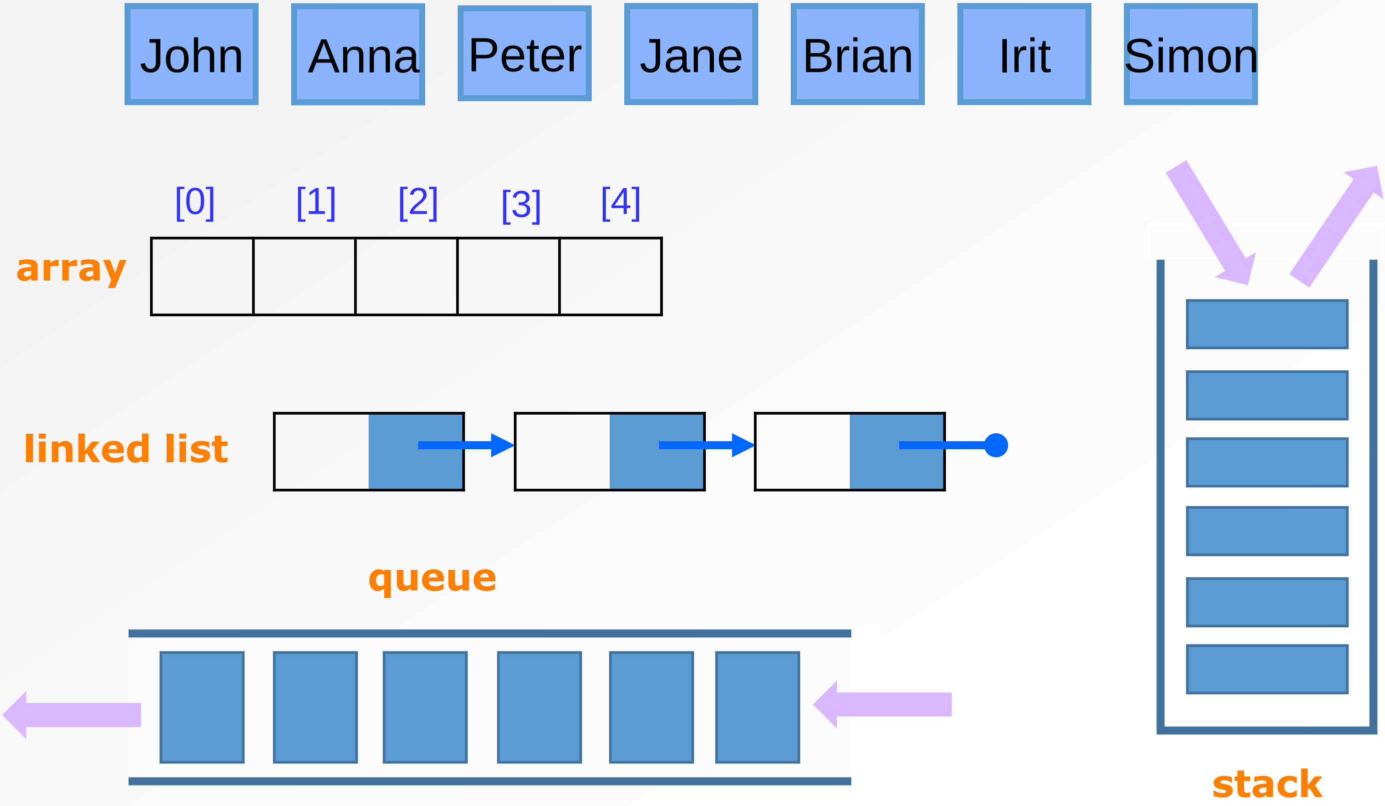
Peter

Simon

- How do you manage them?

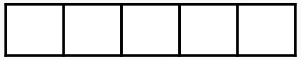
LINEAR DATA STRUCTURE

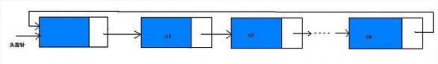
- Array, linked list, queue, stack



DATA STRUCTURES SO FAR…

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

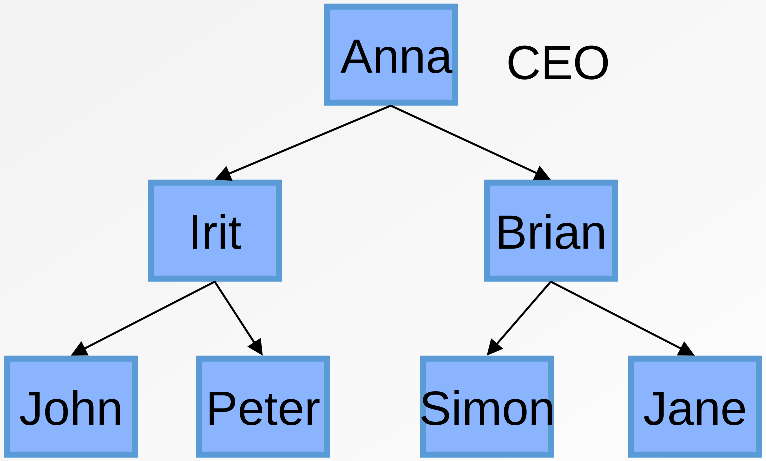




- Used them to store lists of numbers, lists of people, lists of moves, etc
  - Linear data

NON-LINEAR DATA STRUCTURE

- Suppose you have a set of names



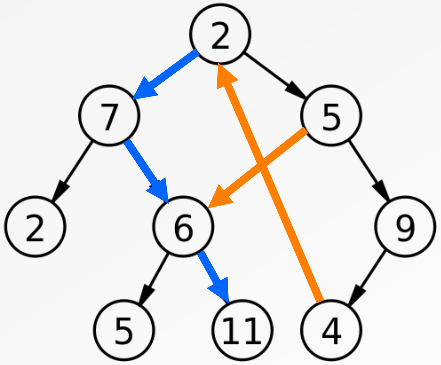
Tree

- Company organization

Not good to use linear data structure to store hierarchical relationships

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:

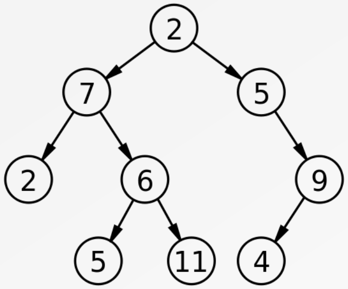
- **If we follow one path of a tree, we get a linked list**

OUTLINE

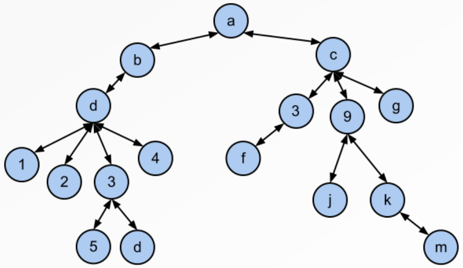
- Non-linear data structures
- **Tree data structure**
  - **Binary trees**
- Implement binary tree nodes in C
- Binary tree traversal
- Example application

TREE DATA STRUCTURE

- Tree data structure looks like…
  - 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

- <u>**Binary**</u> tree (we'll work with this in our course)
  - Each node can have links to **at most two other nodes**

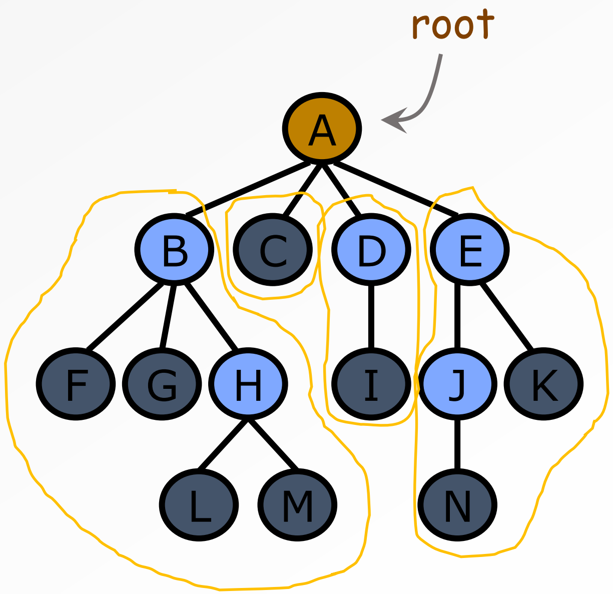
TREE DATA STRUCTURE

- Similar to family tree concept
  - One special node: root
  - Each node can has many children

    A has four children: B, C, D, E
  - Each node (except the root) has a parent node

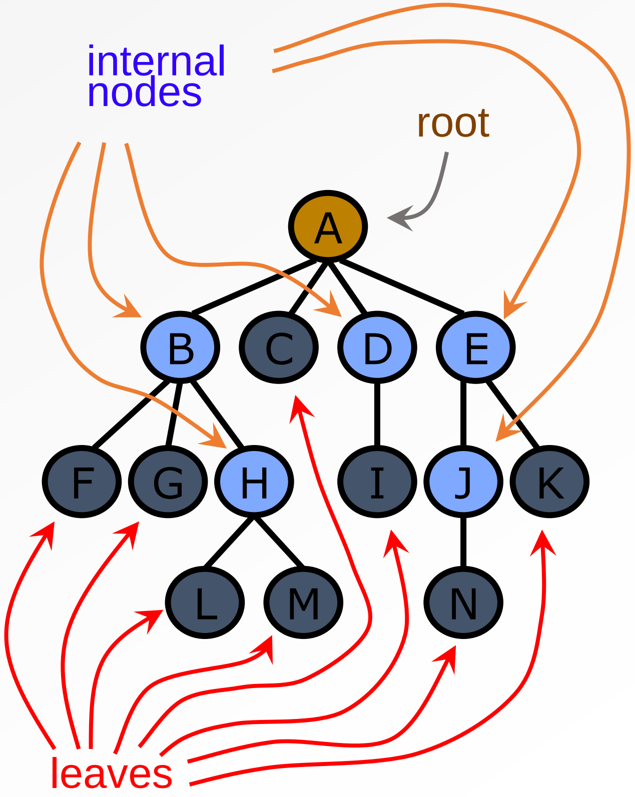
    A is the parent of B, C, D, E
  - Other children of your parent are your siblings

    B, C, D and E are siblings
  - Subtree: Your child and her descendant nodes form a subtree



TREE DATA STRUCTURE

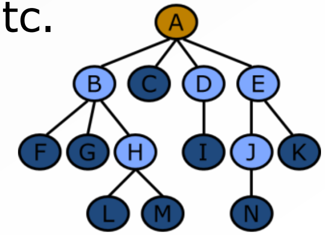
- A tree is composed of nodes
- Each node contains a value
- Types of nodes
  - **Root**: only one in a tree, has no parent.
  - Internal (non-leaf): Nodes with children are called **internal nodes**
  - Leaf: nodes without children are called **leaves**



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**

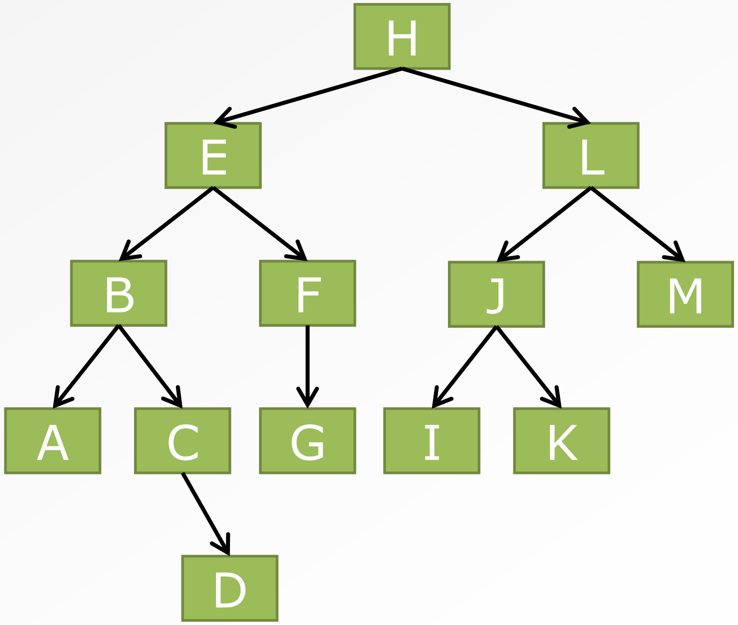


EXAMPLE APPLICATION OF TREE

- You have the following information
  - F has son G
  - J has sons I & K
  - B has sons A & C
  - L has sons J & M
  - H has sons E & L
  - C has son D
  - E has sons B & F
- Now answer these questions
  - Who has no son?
  - Who has no father?
  - Who are the descendants of L?
  - Who are the ancestors of J?
  - Who has exactly 3 descendants?

EXAMPLE APPLICATION OF TREE

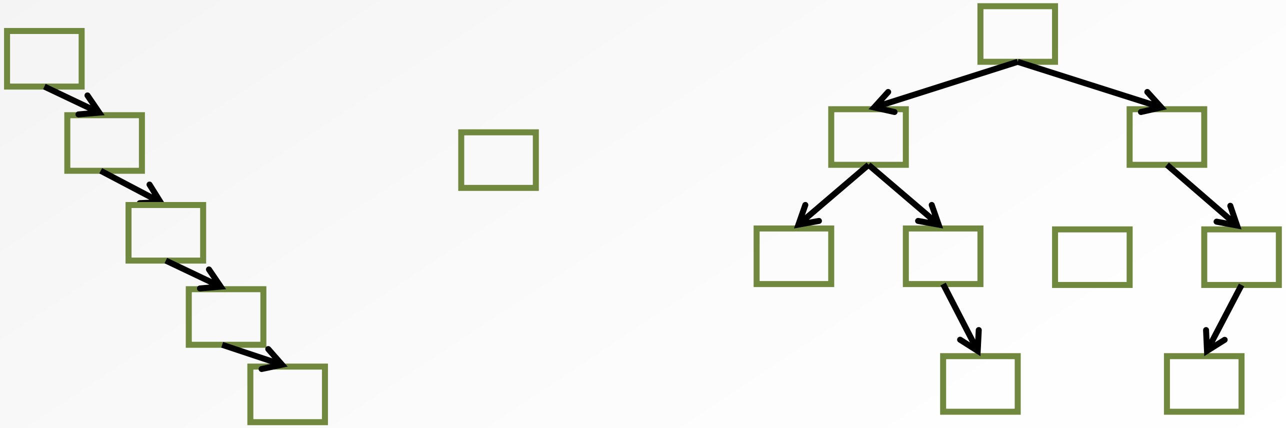
- Build the representative tree
  - F has son G
  - J has sons I & K
  - B has sons A & C
  - L has sons J & M
  - H has sons E & L
  - C has son D
  - E has sons B & F



- Now, questions again
  - Who has no son?
  - Who has no father?
  - Who are the descendants of L?
  - Who are the ancestors of J?
  - Who has exactly 3 descendants?
- Much better!

POSSIBLE TREE CONFIGURATIONS

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



OUTLINE

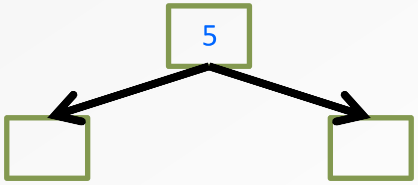
- Non-linear data structures
- Tree data structure
  - Binary trees
- **Implement binary tree nodes in C**
- Binary tree traversal
- Example application

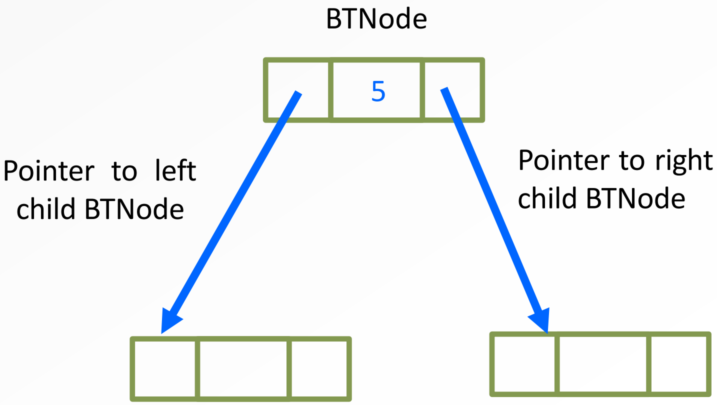
IMPLEMENTATION

- 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





BTNode

Pointer to left child BTNode

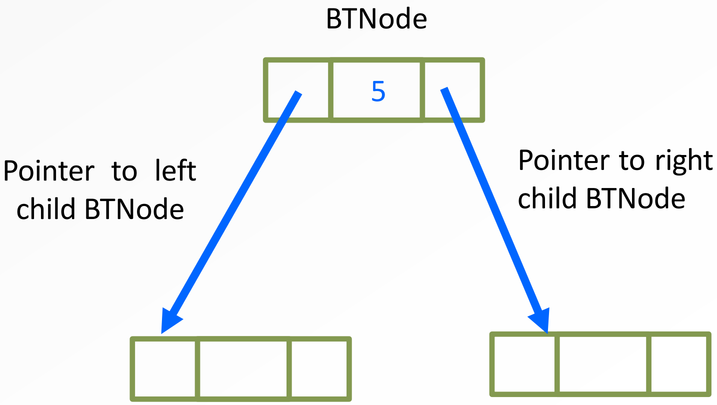
Pointer to right child BTNode

BTNode

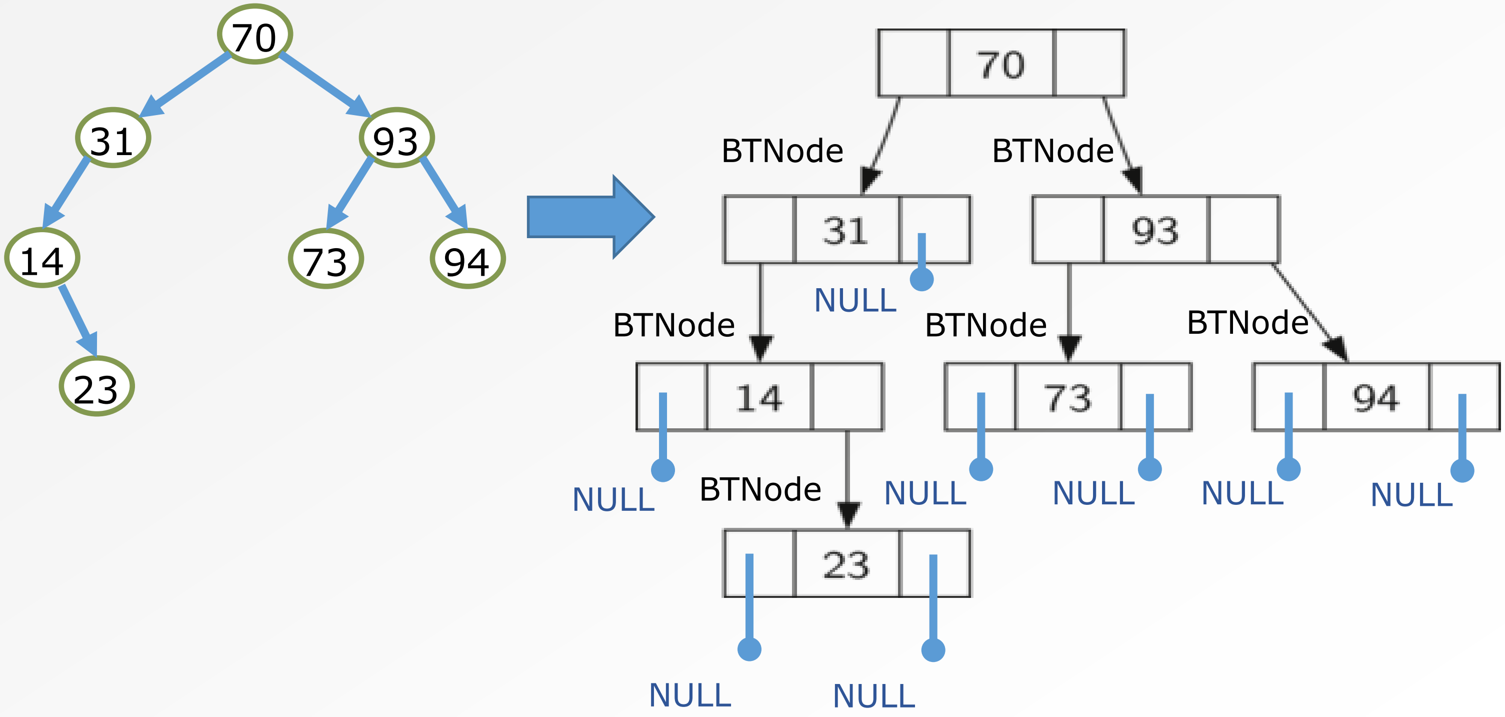
- Start with a simple BTNode that stores an integer
  - The type of item can be character, string, or structure, etc.

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

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



EXAMPLE BINARY TREE

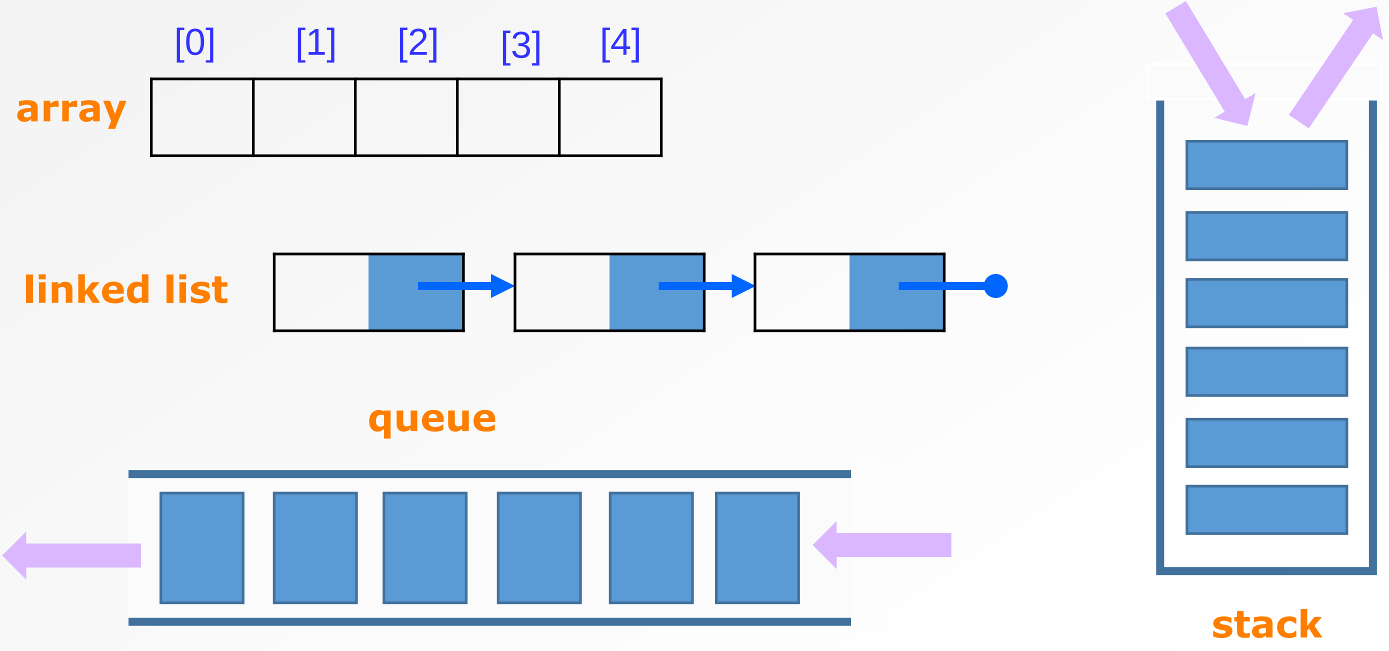


OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- **Binary tree traversal**
- Example application

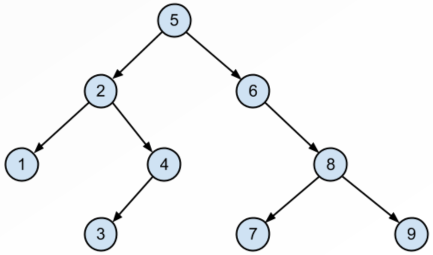
TREE TRAVERSAL

- 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



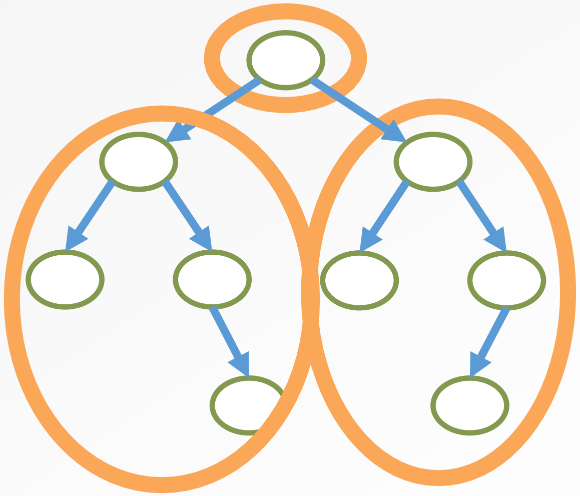
TREE TRAVERSAL

- 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

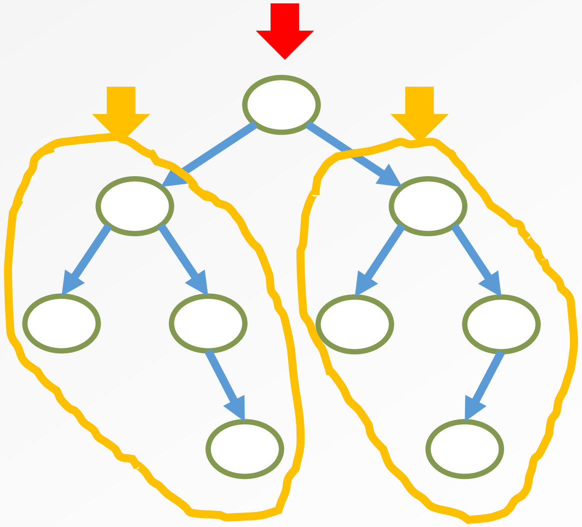
  = size of N's left subtree

  + size of N's right subtree

  + N itself

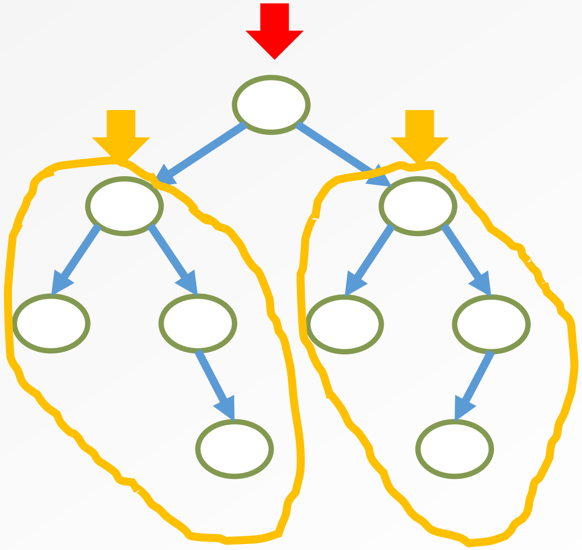
TREE TRAVERSAL

- 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

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



Let’s go through the process!

Deep look through animations

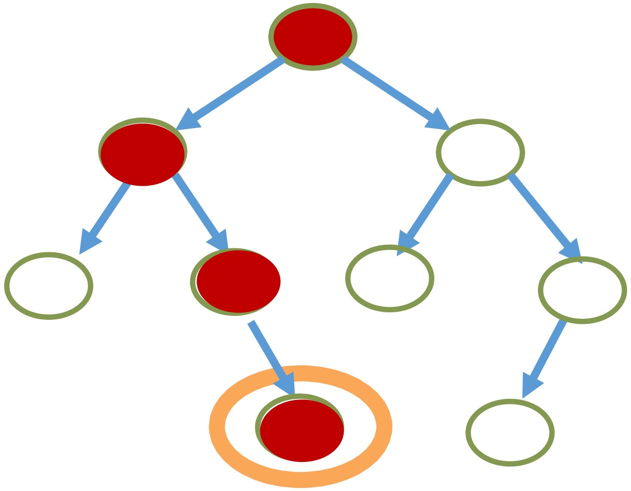
In main(), call
TreeTraversal(root)

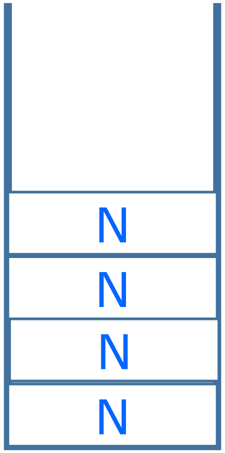
```
...
if (N has left child)
    TreeTraversal(LeftChild);
```

```
TreeTraversal(Node N):
  Visit N;
  if (N has left child)
       TreeTraversal(LeftChild);
  if (N has right child)
      TreeTraversaRightChild);
```

```
TreeTraversal(Node N):
 Visit N;
 if (N has left child)
      TreeTraversal(LeftChild);
 if (N has right child)
     TreeTraversal(RightChild);
```

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

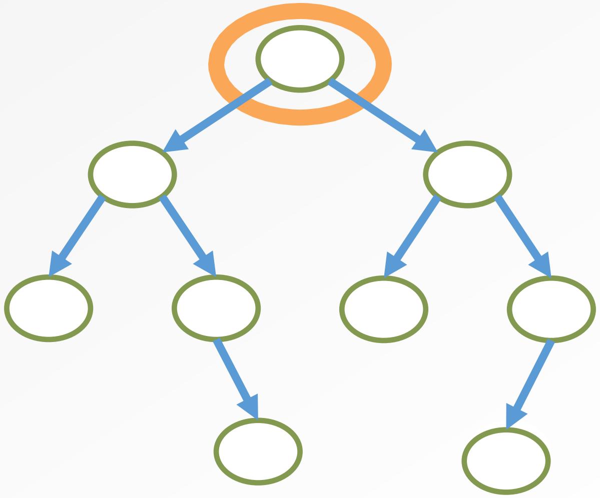




TREE TRAVERSAL TEMPLATE

Pseudocode

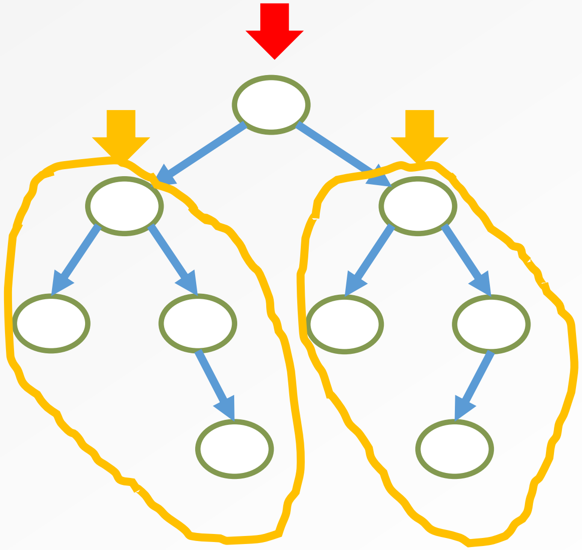
```
TreeTraversal(Node N):
    Visit N;
    If (N has left child)
        TreeTraversal(LeftChild);
    If (N has right child)
        TreeTraversal(RightChild);
    Return; // return to parent
```



In main(), call TreeTraversal(root)

TREE TRAVERSAL TEMPLATE #2

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

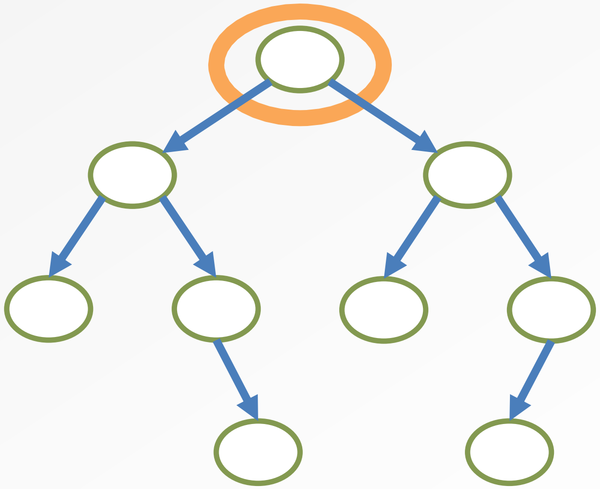


- New version:
  - Always follow links to children
  - Then check if the link is NULL
  - In other words, not actually pointing at a BTNode

TREE TRAVERSAL TEMPLATE #2

Pseudocode

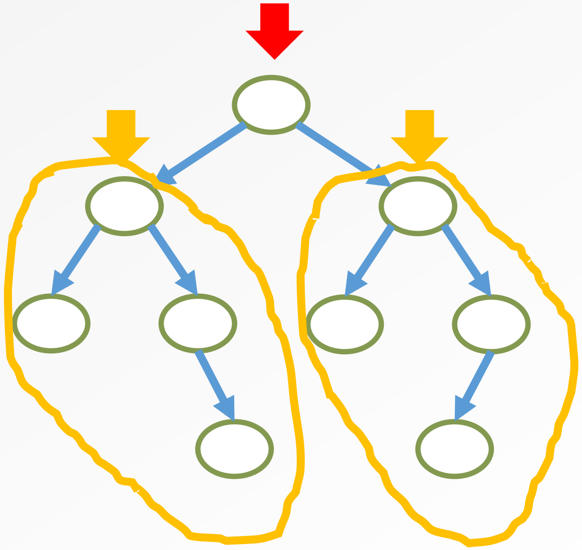
```
TreeTraversal2(Node N):
    If N==NULL return;
    Visit N;
    TreeTraversal2(LeftChild);
    TreeTraversal2(RightChild);
    Return; // return to parent
```



In main(), call TreeTraversal2(root)

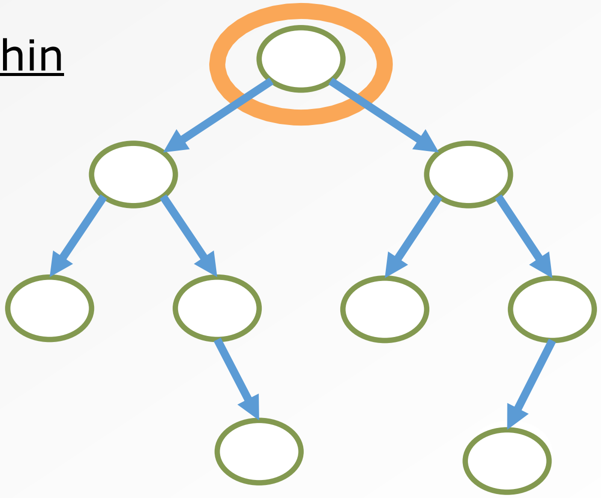
TreeTraversal2() IMPLEMENTATION

```
Void TreeTraversal2(BTNode *cur){

     If (cur == NULL) return;

     PrintNode(cur); // visit cur

     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)

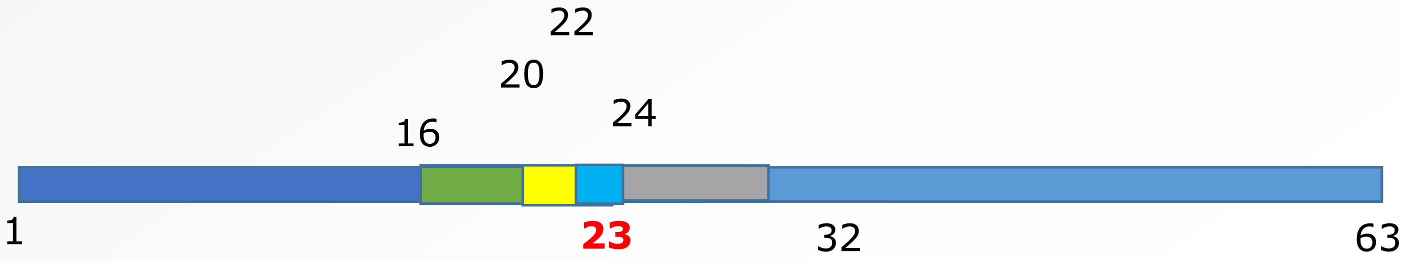
OUTLINE

- Non-linear data structures
- Tree data structure
  - Binary trees
- Implement binary tree nodes in C
- Binary tree traversal
- **Example application**

EXAMPLE APPLICATION: NUMBER GAME

- Player has to think of a number between **1 and 63**.
- Computer asks questions, each with a "**Yes/too big/too small**" answer. Stop if gets a 'Yes' answer
- For example: 6 questions to get the answer (**23**)

- Is it **32**?	**Answer: too big**
- Is it **16?**	**Answer: too small**
- Is it **24?**	**Answer: too big**
- Is it **20?**	**Answer: too small**
- Is it **22?**	**Answer: too small**
- Is it **23?**	**Answer: Yes!**



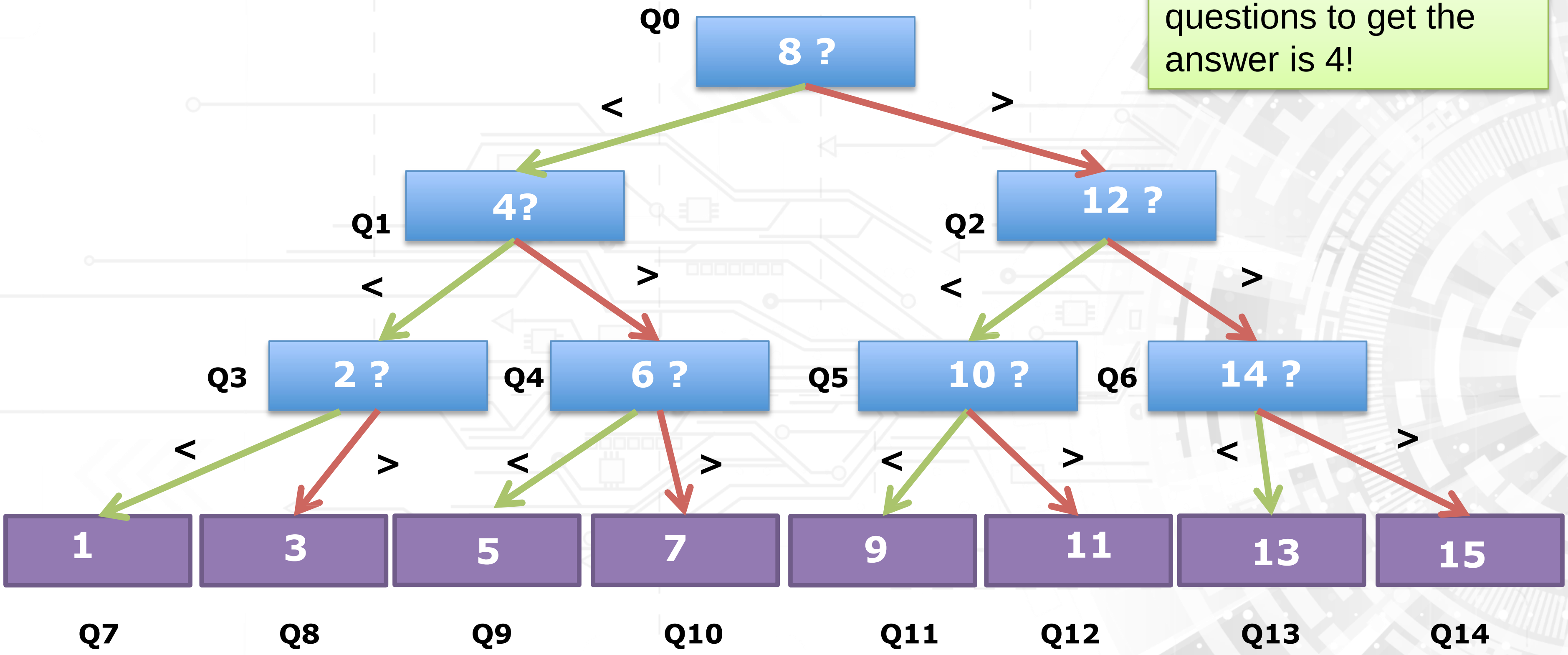
EXAMPLE APPLICATION: NUMBER GAME

- Player has to think of a number between **1 and 63**.
- Computer asks questions, each with a "**Yes/too big/too small**" answer. Stop if gets a 'Yes' answer
- For example: 6 questions to get the answer (**23**)

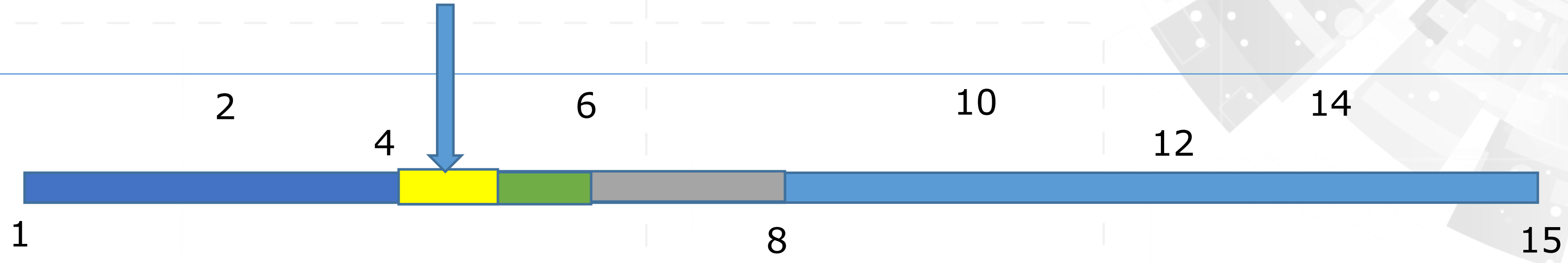
- Is it **32**?	**Answer: too big**
- Is it **16?**	**Answer: too small**
- Is it **24?**	**Answer: too big**
- Is it **20?**	**Answer: too small**
- Is it **22?**	**Answer: too small**
- Is it **23?**	**Answer: Yes!**

- Paths through the set of possible questions form a tree
  - Eventually leads to a guess at what the number is
- We'll play a simplified version – number game of [1,15]

NUMBER GAME [1, 15]



Maximum number of questions to get the answer is 4!



YOU SHOULD BE ABLE TO...

- Implement a node-based binary tree
- Choose a binary tree data structure to solve a problem when appropriate
- Explain the sequence of node visitation for the tree traversal template