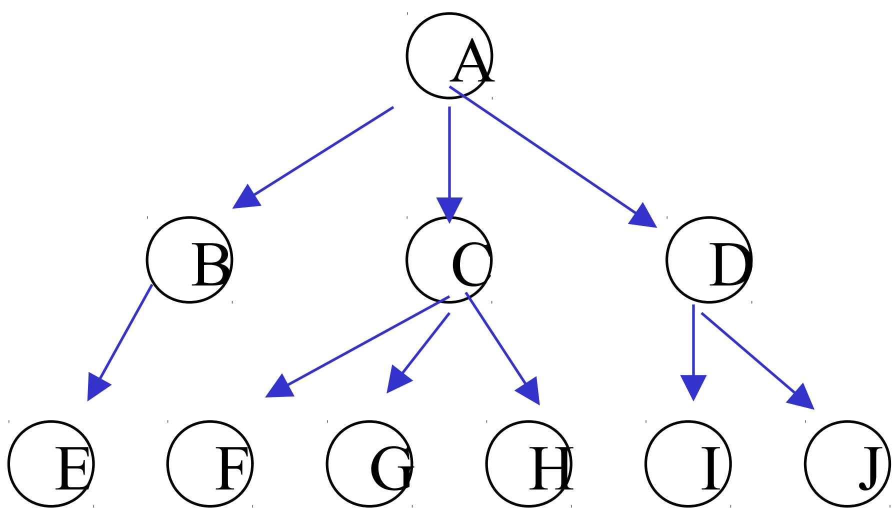
Computer Science 112
Data Structures

Lecture 14:
Trees

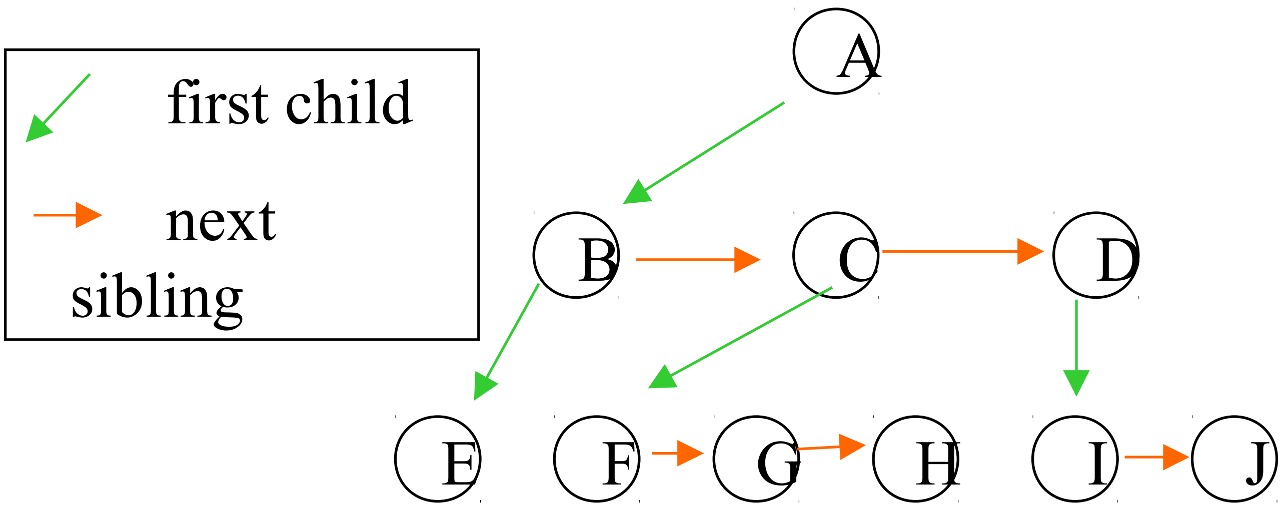
General Trees

- **Each node has an arbitrary number of children**
- **Problem: representation of a node**



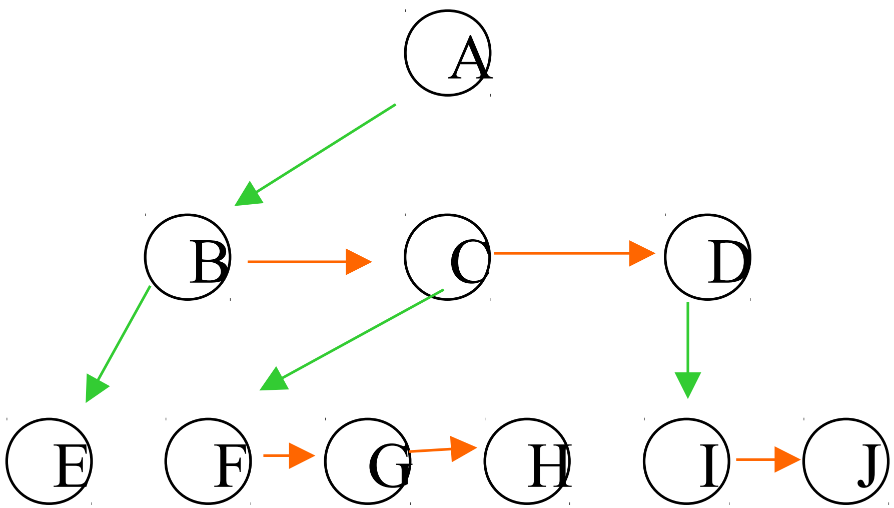
General Trees

- Each node has an arbitrary number of children
- Problem: representation
- Solution: linked list of children



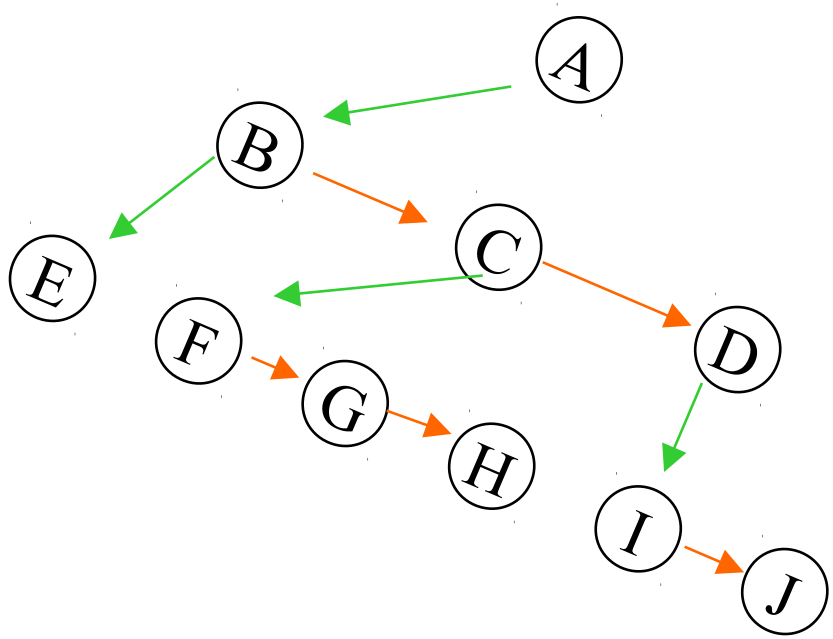
General Tree as Binary

- **First child <=> Left child**
- **Next sib <=> Right child**



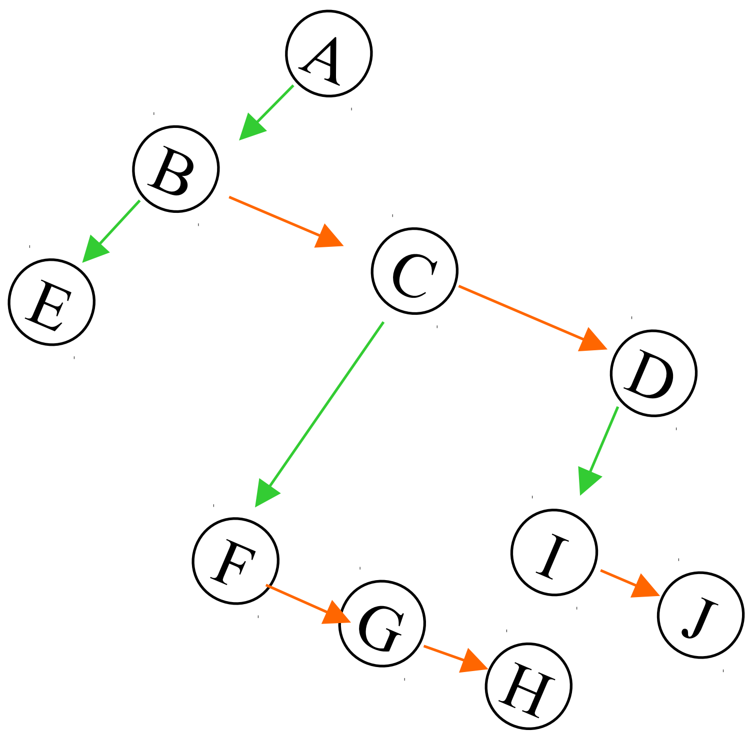
General Tree as Binary

- **First child <=> Left child**
- **Next sib <=> Right child**



General Tree as Binary

- **First child <=> Left child**
- **Next sib <=> Right child**



DOM tree in assignment

<html>

<body>

<p>

this is a line

</p>

<p>

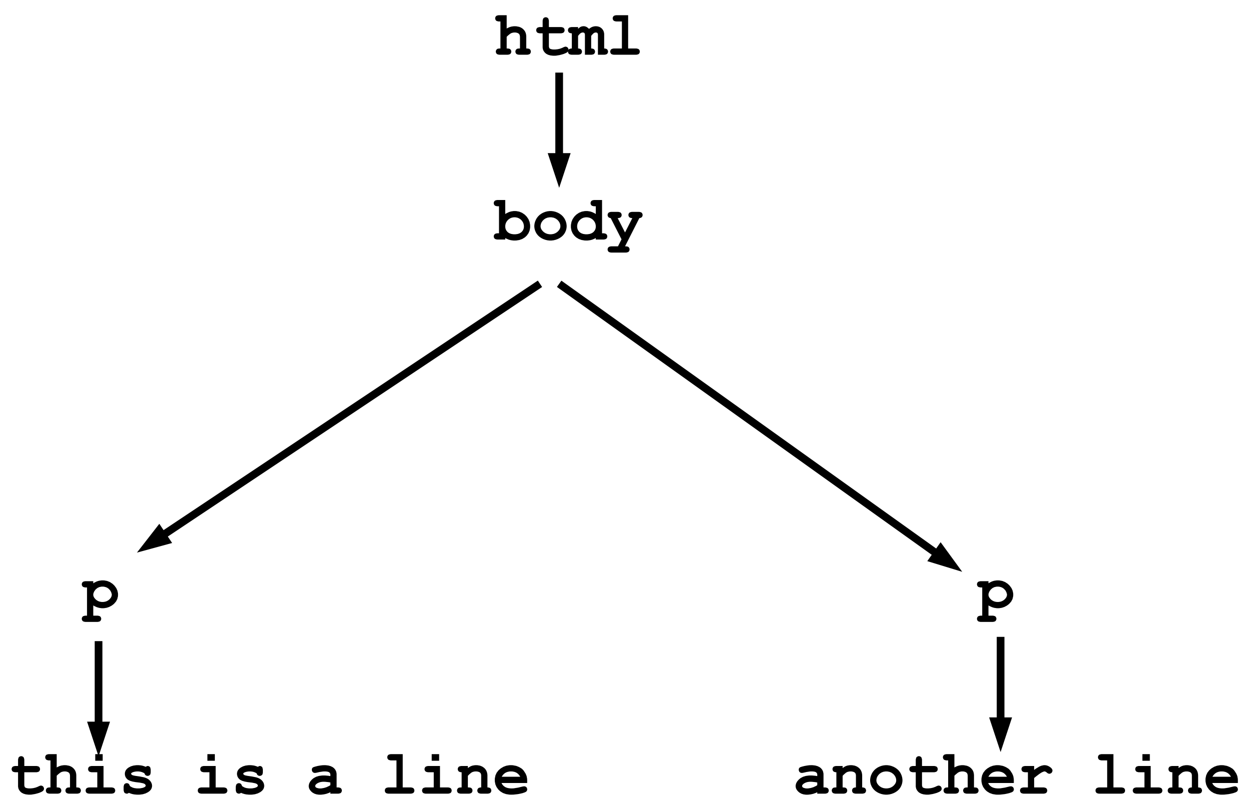
another line

</p>

</body>

</html>

The tree as we think about it

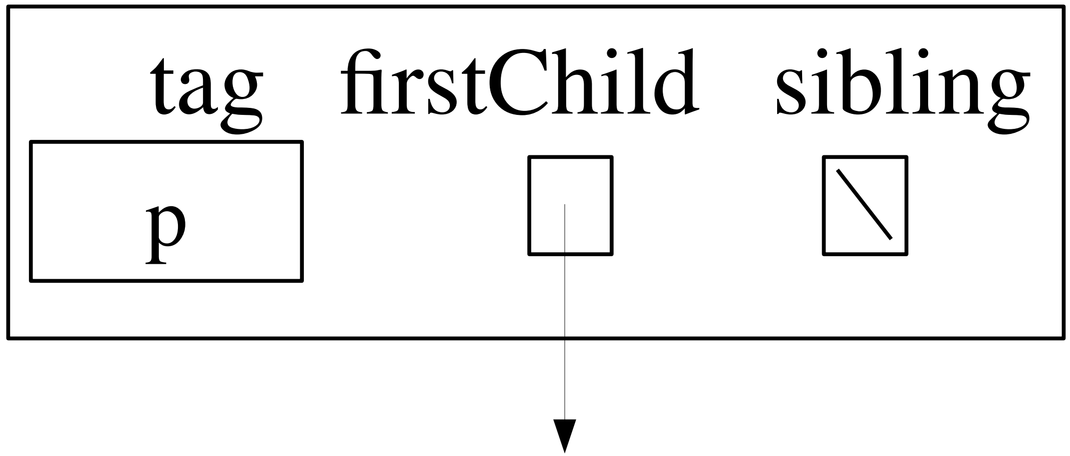


The tree as we drew it

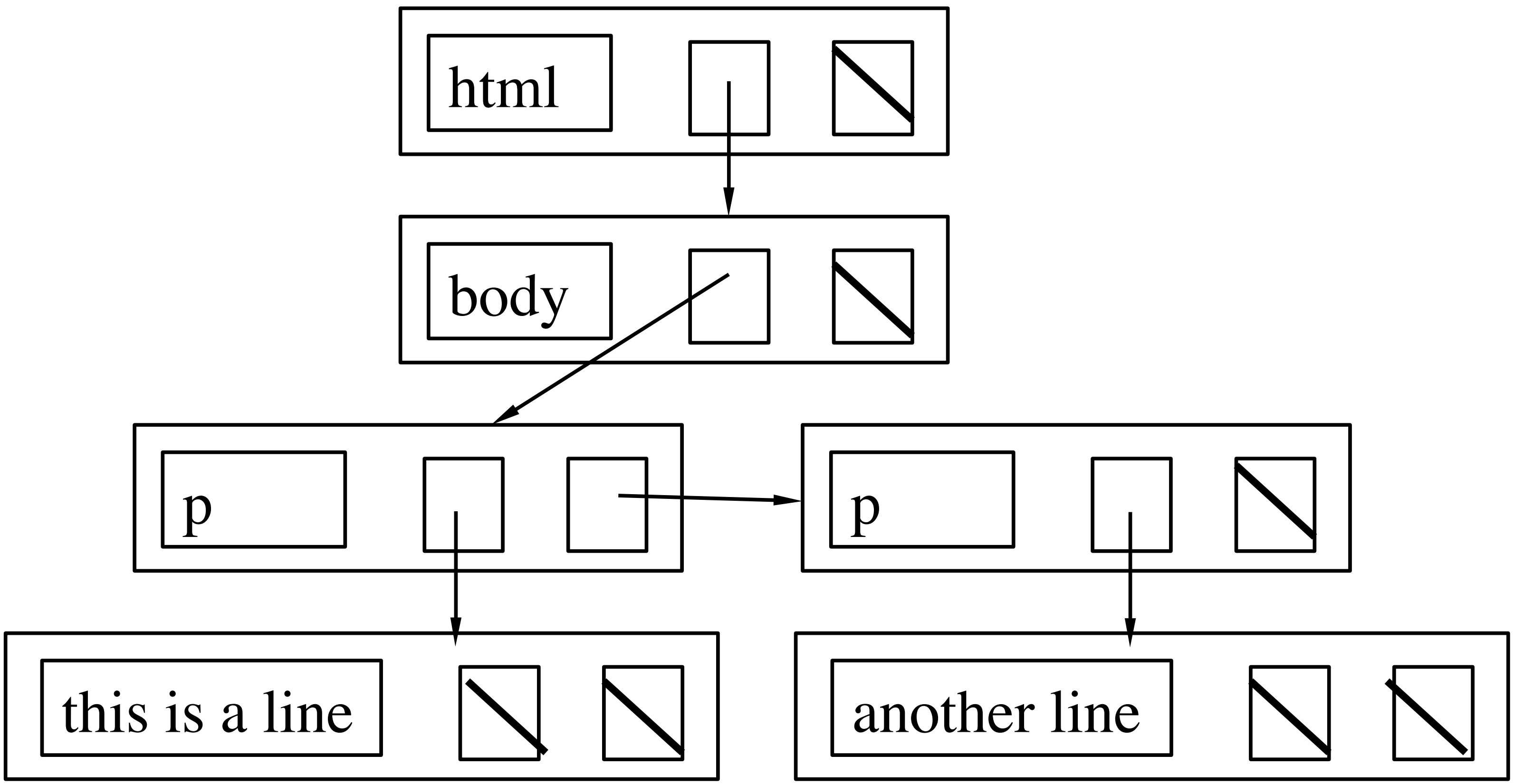
```
               html
                |
               body
                |
   ---------------------------
   |                         |
   p                         p
   |                         |
this is a line          another line
```

TagNodes

```
--------
|p|  |\|
---|---
   |
   v
```

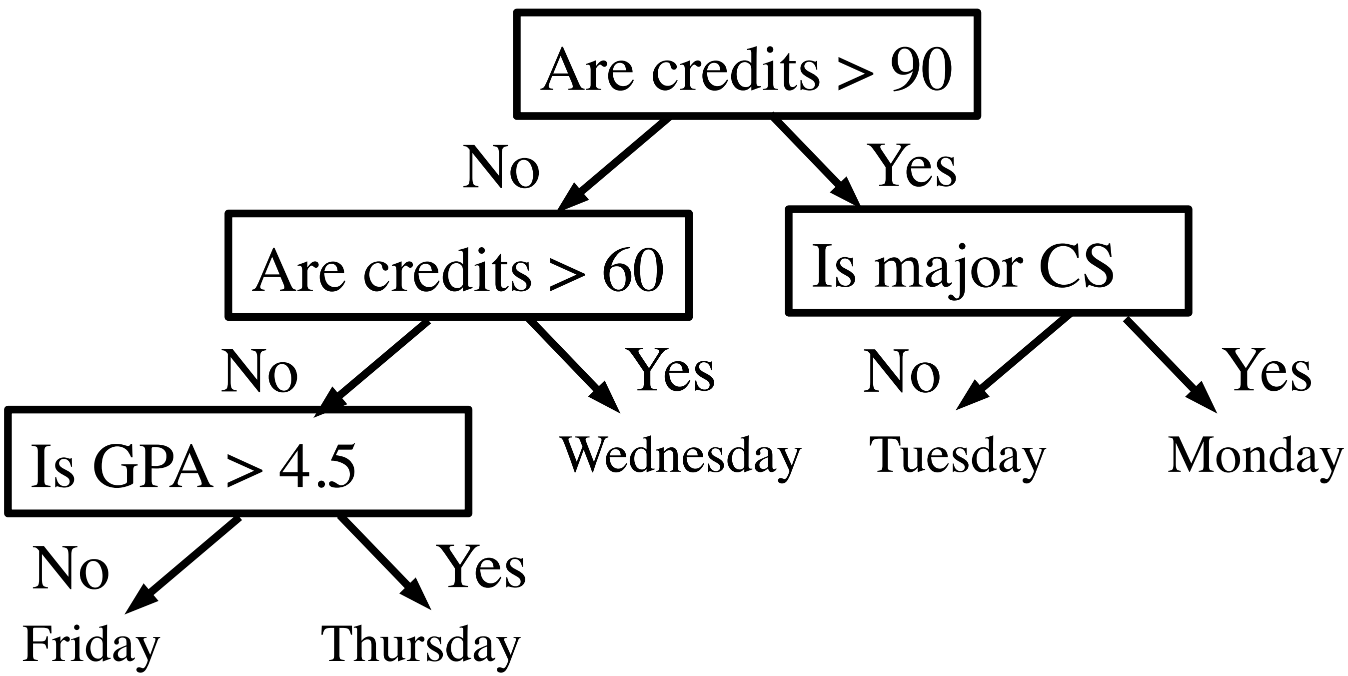


Tree as TagNodes



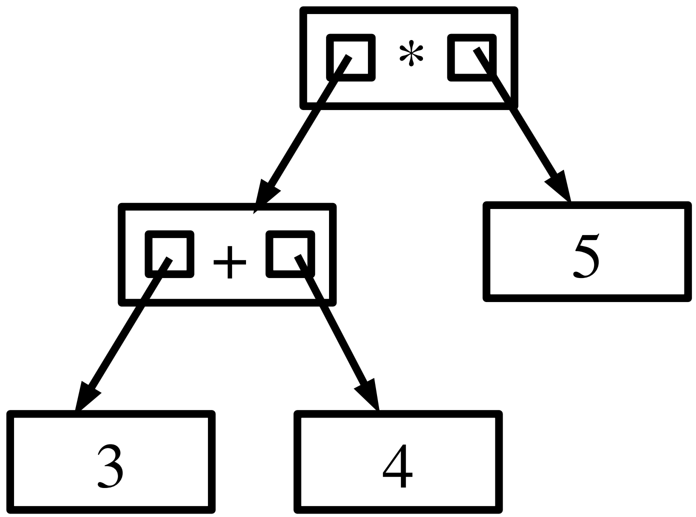
Some uses of Binary Trees

- **Decision trees: e.g., when can you register**



Some uses of Binary Trees

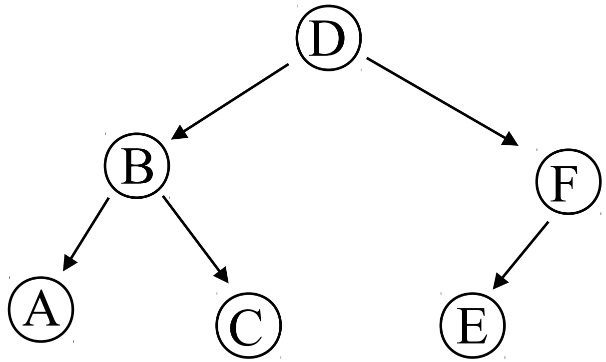
- **Expression Trees: (3+4)*5**

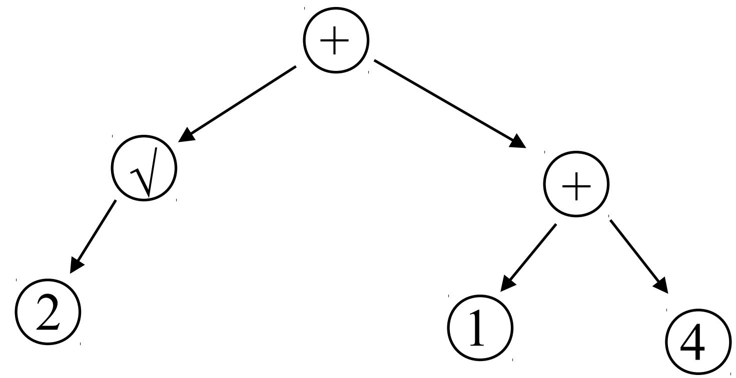


Tree Traversals

- **“Visit” each node in the tree**
  - **“visit” = do something, e.g. print**
- **In some systematic order**
  - **eg “inorder”:**
    visit all nodes in left subtree
    then visit node
    then visit all nodes in right subtree

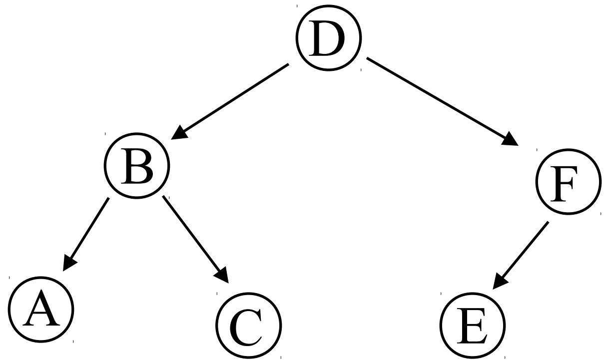
Traversals

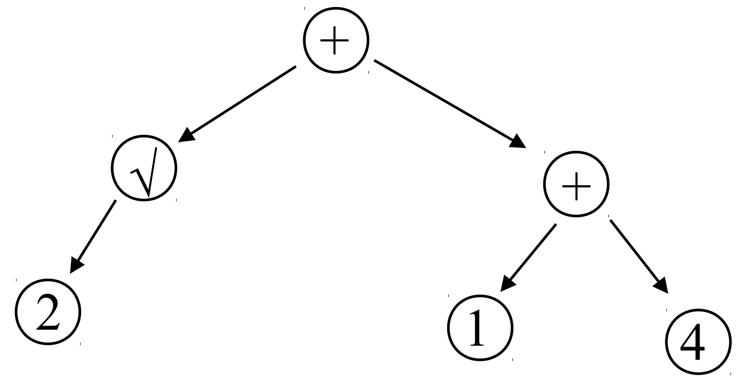




InOrder ____

Traversals





InOrder A B C D E F 2 √ + 1 + 4

See TreeNode.java

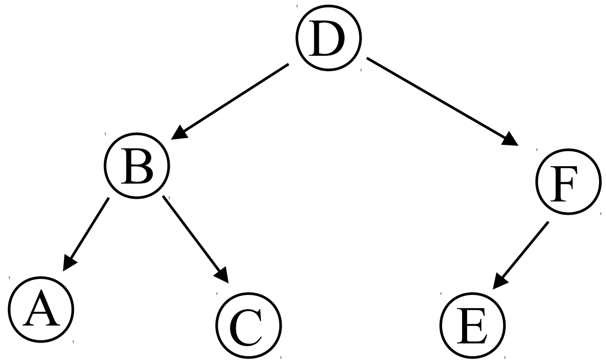
Recursive Traversals

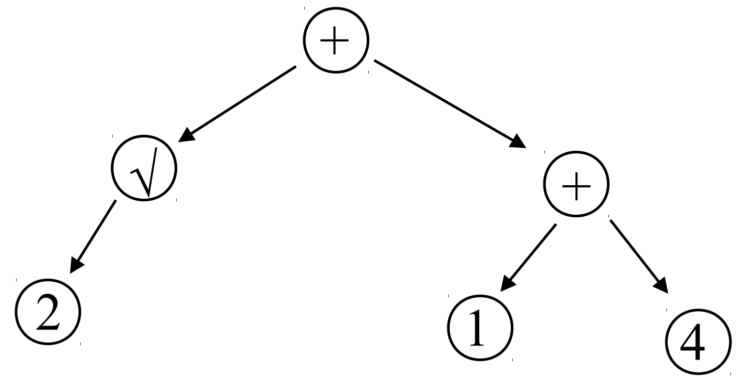
```
inOrderPrint(tree):
    if (tree == null)
        return
    inOrderPrint(tree.lst)
    print(tree.node)
    inOrderPrint(tree.rst)
```

```
preOrderPrint(tree):
    if (tree == null)
        return
    print(tree.node)
    preOrderPrint(tree.left)
    preOrderPrint(tree.right)
```

```
postOrderPrint(tree):
    if (tree == null)
        return
    postOrderPrint(tree.lst)
    postOrderPrint(tree.rst)
    print(tree.node)
```

Traversals



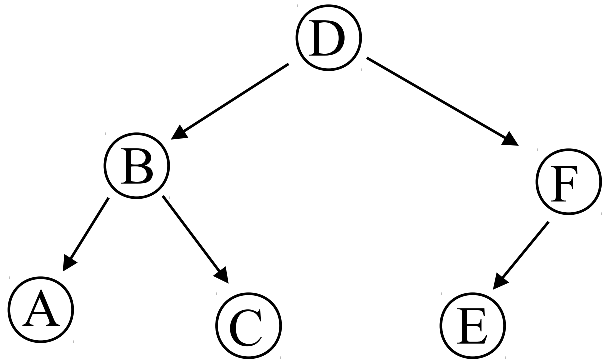


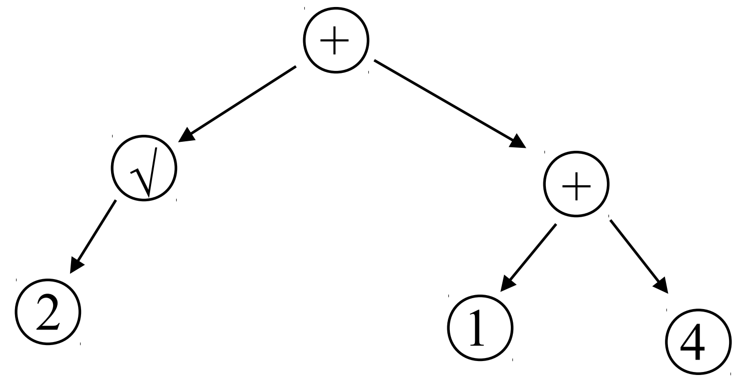
InOrder A B C D E F 2 √ + 1 + 4

PreOrder ______________________________

PostOrder ______________________________

Traversals





InOrder	A B C D E F	2 √ + 1 + 4
PreOrder	D B A C F E	+ √ 2 + 1 4
PostOrder	A C B E F D	2 √ 1 4 + +

From here to end of lecture is optional

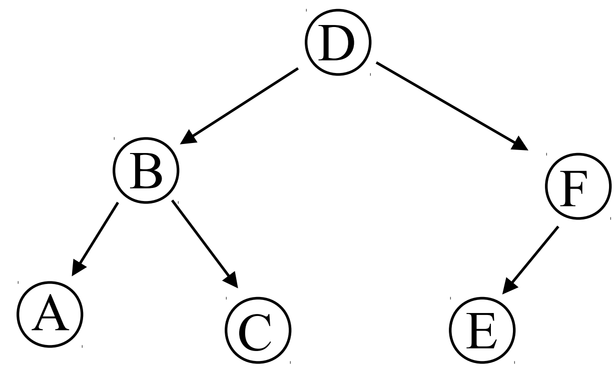
- **Will not be on any exam**

Non-recursive Traversals: Not the books approach

- **Problem: If a node has more than one child**
  - **can't work on all children, grandchildren, … at once**
  - **have to store children that have been found but not processed**
- **Solution: store in a stack or queue**

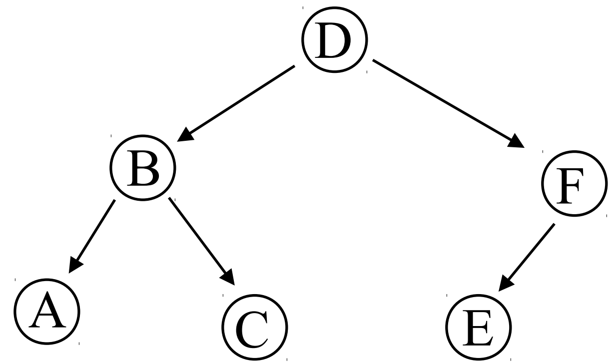
Stack-based Traversal

```
s.push(root);
while(! s.isEmpty( )){
   next = s.pop( );
   if (next != null){
     print next.data;
     s.push next.rightSubTree;
     s.push next.leftSubTree;
}}
```



Queue-based Traversal

```
q.enqueue(root);
while(! q.isEmpty( )){
  next = q.dequeue( );
  if (next != null){
    print next.data;
    q.enqueue next.leftSubTree;
    q.enqueue next.rightSubTree;
}}
```



Breadth vs Depth first

- **Stack: depth first**
  - **do all children before anything else**
- **Queue: breadth first**
  - **do all at same level before anything else**

Size of Stack / Queue

- **Stack: path from root to leaf: O(height)**
- **Queue: entire level: O(2^{height})**
  - **That's a lot!**
  - **Solution: Iterative Deepening**

Iterative Deepening

- **print all nodes at depth d:**

```
idfs(tree, d)
      if d == 0
          print tree.data
      else idfs(tree.lst, d-1)
             idfs(tree.rst, d-1)
```

- **Try all depths**

```
for(j=0; j<maxDepth; j++)
      idfs(tree, j)
```

Iterative Deepening

- **How much extra work?**
  - **How many leaves in complete binary tree of depth d? 2^d**
  - **How many non-leaves: 2^d-1**
- **Time overhead: roughly a factor of 2**

Stack based traversal: The book's approach

- **What to store?**
- **Mimic stack of invocation records from recursive version**
  - **TreeNode node;**
  - **int milestone;**

- **See TreeNode.java**