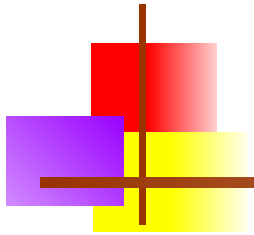
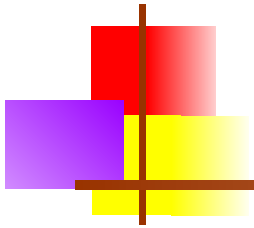


Trees

LO:
build a tree of a data structure



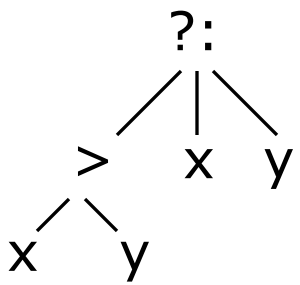
File systems

- File systems are almost always implemented as a tree structure
  - The nodes in the tree are of (at least) two types: *folders* (or *directories*), and *plain files*
  - A folder typically has children—subfolders and plain files
    - A folder also contains a link to its parent—in both Windows and UNIX, this link is denoted by **..**
    - In UNIX, the root of the tree is denoted by **/**
  - A plain file is typically a leaf

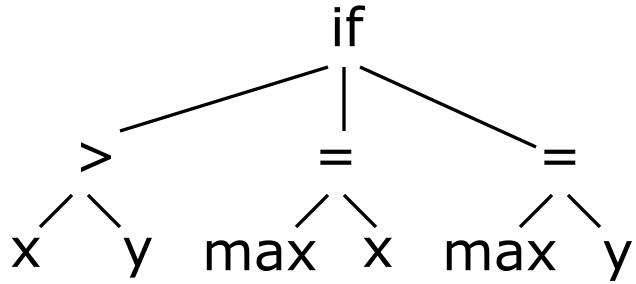
Family trees

Trees for expressions and statements

- Examples:



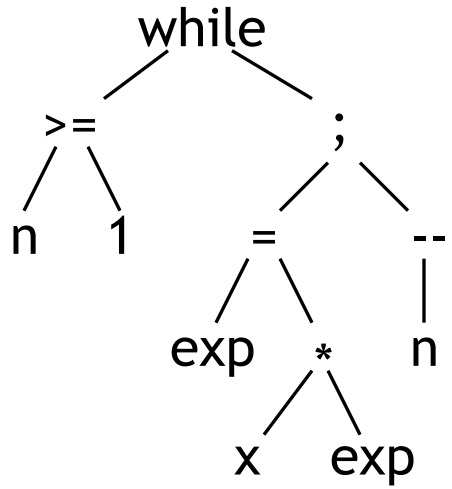
The expression x > y ? x : y



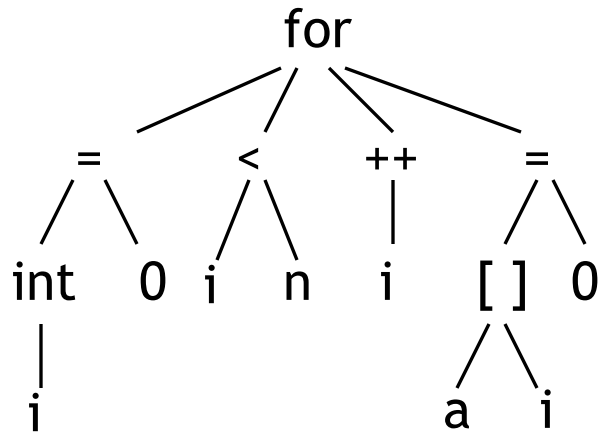
The statement if (x > y) max = x;
else max = y;

More trees for statements

```
while (n >= 1) {
    exp = x * exp;
    n--;
}
```

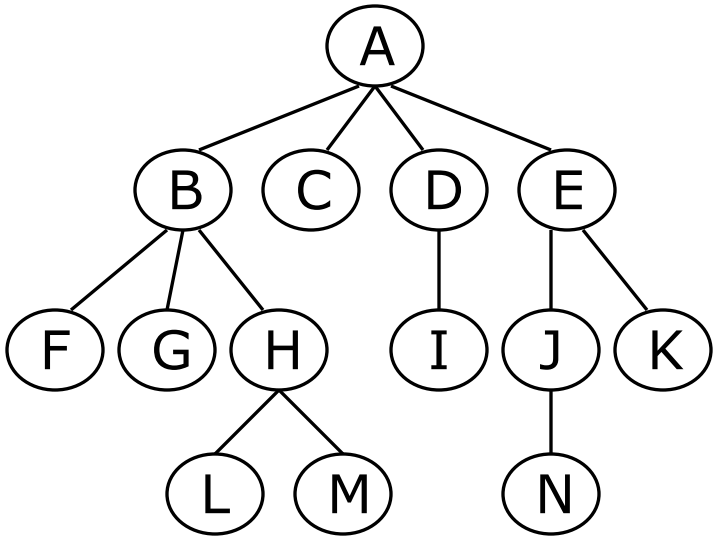



```
for (int i = 0; i < n; i++)
    a[i] = 0;
```

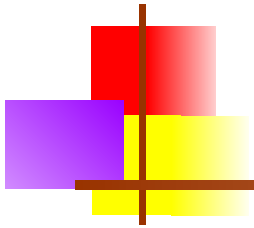


Definition of a tree

- A tree is a node with a value and zero or more children
  - Depending on the needs of the program, the children may or may not be ordered



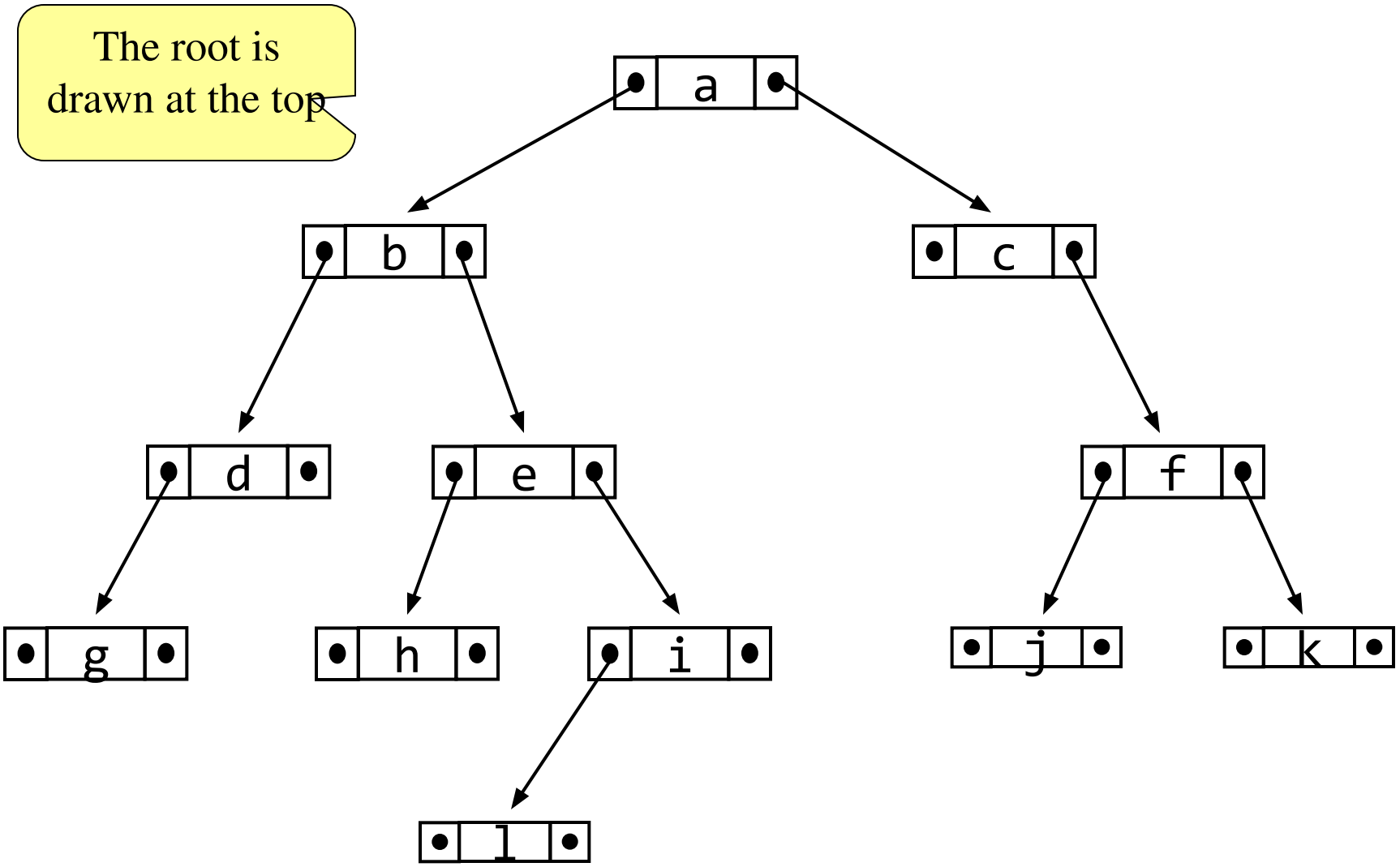
- A tree has a root, internal nodes, and leaves
- Each node contains an element and has branches leading to other nodes (its children)
- Each node (other than the root) has a parent
- Each node has a depth (distance from the root)



Parts of a binary tree

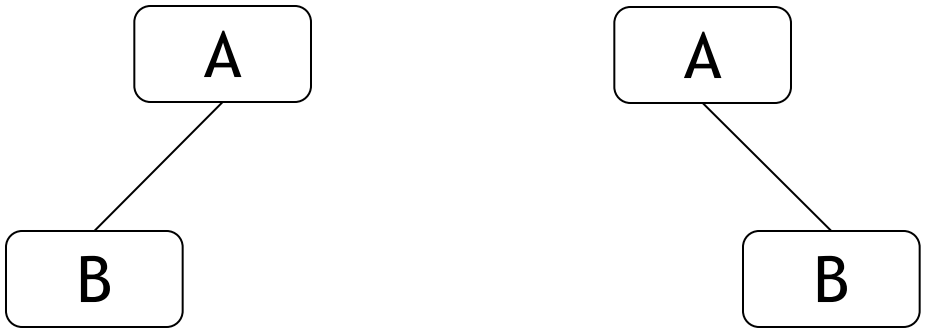
- A binary tree is composed of zero or more nodes
- Each node contains:
  - A value (some sort of data item)
  - A reference or pointer to a left child (may be `null`), and
  - A reference or pointer to a right child (may be `null`)
- A binary tree may be *empty* (contain no nodes)
- If not empty, a binary tree has a root node
  - Every node in the binary tree is reachable from the root node by a *unique* path
- A node with no left child and no right child is called a leaf
  - In some binary trees, only the leaves contain a value

Picture of a binary tree



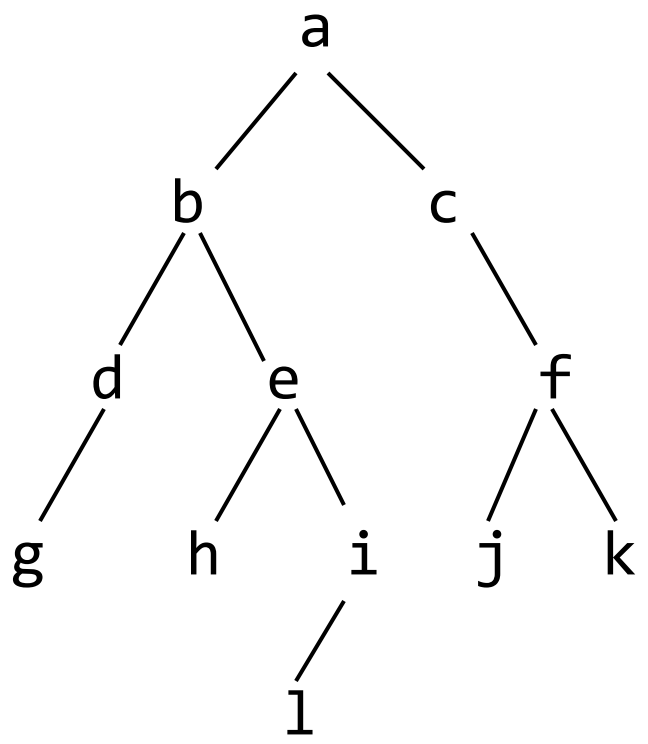
Left ≠ Right

- The following two binary trees are *different:*



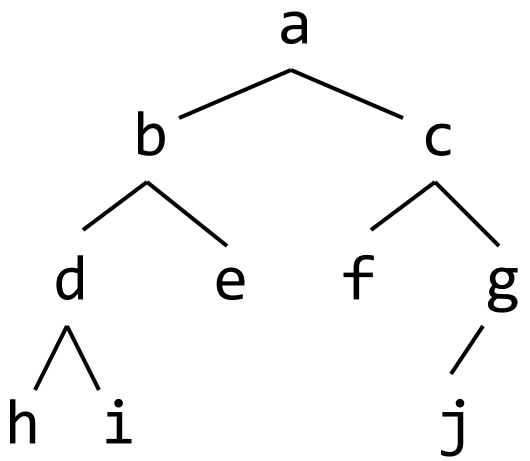
- In the first binary tree, node A has a left child but no right child; in the second, node A has a right child but no left child
- Put another way: Left and right are *not* relative terms

Size and depth

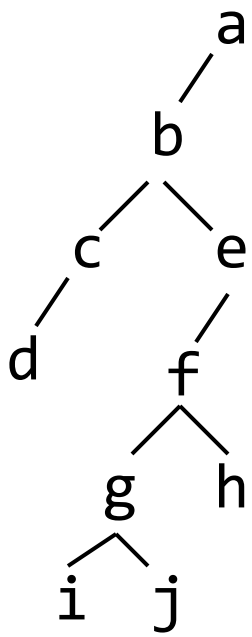


- The size of a binary tree is the number of nodes in it
  - This tree has size 12
- The depth of a node is its distance from the root
  - a is at depth zero
  - e is at depth 2
- The depth of a binary tree is the depth of its deepest node
  - This tree has depth 4

Balance



A balanced binary tree



An unbalanced binary tree

- A binary tree is balanced if every level above the lowest is "full" (contains 2^n nodes)
- In most applications, a reasonably balanced binary tree is desirable

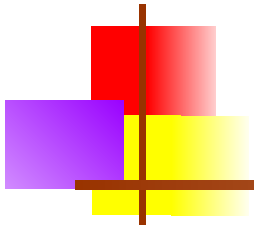
- **Breadth-first**

Traversing a tree in breadth-first order means that after visiting a node X, all of X's children are visited, then all of X's 'grand-children' (i.e. the children's children), then all of X's 'great-grand-children', etc. In other words, the tree is traversed by sweeping through the breadth of a level before visiting the next level down.

- **Depth-first**

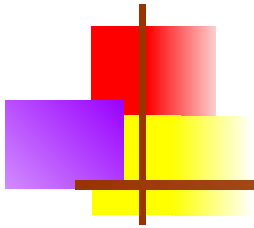
As the name implies, a depth-first traversal will go down one branch of the tree as far as possible, i.e. until it stops at a leaf, before trying any other branch. The various branches starting from the same parent may be explored in any order. For the example tree, two possible depth-first traversals are F B A D C E G I H and F G I H B D E C A.

- **Depth First traversal generally uses a Stack**
- **Breadth First generally uses a Queue**



Tree traversals

- A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree
- To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once
- Tree traversals are naturally recursive
- Since a binary tree has three "parts," there are six possible ways to traverse the binary tree:
  - root, left, right
  - left, root, right
  - left, right, root
  - root, right, left
  - right, root, left
  - right, left, root

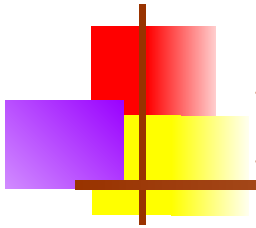


Preorder traversal

In preorder, the root is visited *first*

If each node is visited before both of its subtrees, then it's called a pre-order traversal. The algorithm for left-to-right pre-order traversal is:

- Visit the root node (generally output it)
- Do a pre-order traversal of the left subtree
- Do a pre-order traversal of the right subtree

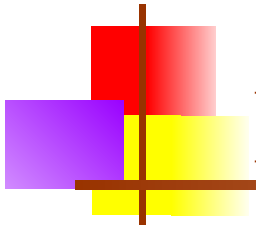


Inorder traversal

- In inorder, the root is visited *in the middle*

If each node is visited *between* visiting its left and right subtrees, then it's an *in*-order traversal. The algorithm for left-to-right in-order traversal is:

- Do an in-order traversal of the left subtree
- Visit root node (generally output this)
- Do an in-order traversal of the right subtree



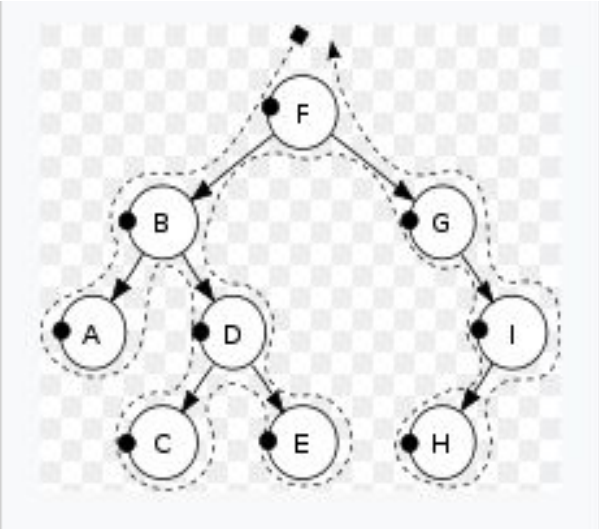
Postorder traversal

- In postorder, the root is visited *last*

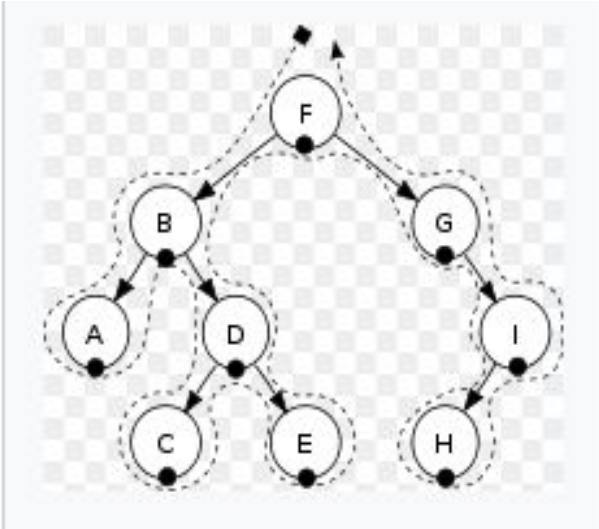
If each node is visited *after* its subtrees, then it's a *post*-order traversal. The algorithm for left-to-right post-order traversal is:

- Do a post-order traversal of the left subtree
- Do a post-order traversal of the right subtree
- Visit the root node (generally output this)

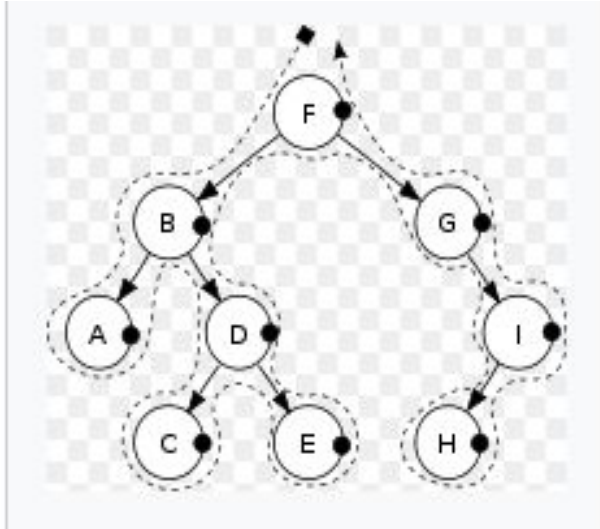
The 3 different types of left-to-right traversal



Pre-order
FBADCEGIH



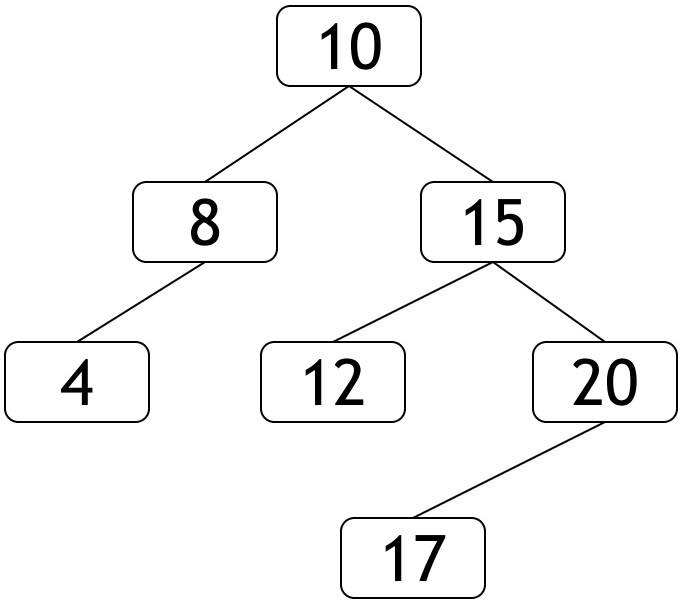
In-order
ABCDEFGHI

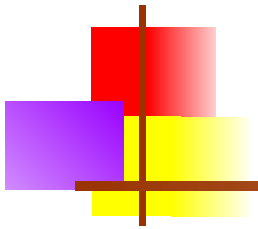


Post-order
ACEDBHIGF

Sorted binary trees

- A binary tree is sorted if every node in the tree is larger than (or equal to) its left descendants, and smaller than (or equal to) its right descendants
- Equal nodes can go either on the left or the right (but it has to be consistent)





- https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_data_structures/Trees
- https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_algorithms/Tree_traversal
- https://en.wikibooks.org/wiki/Data_Structures/Trees