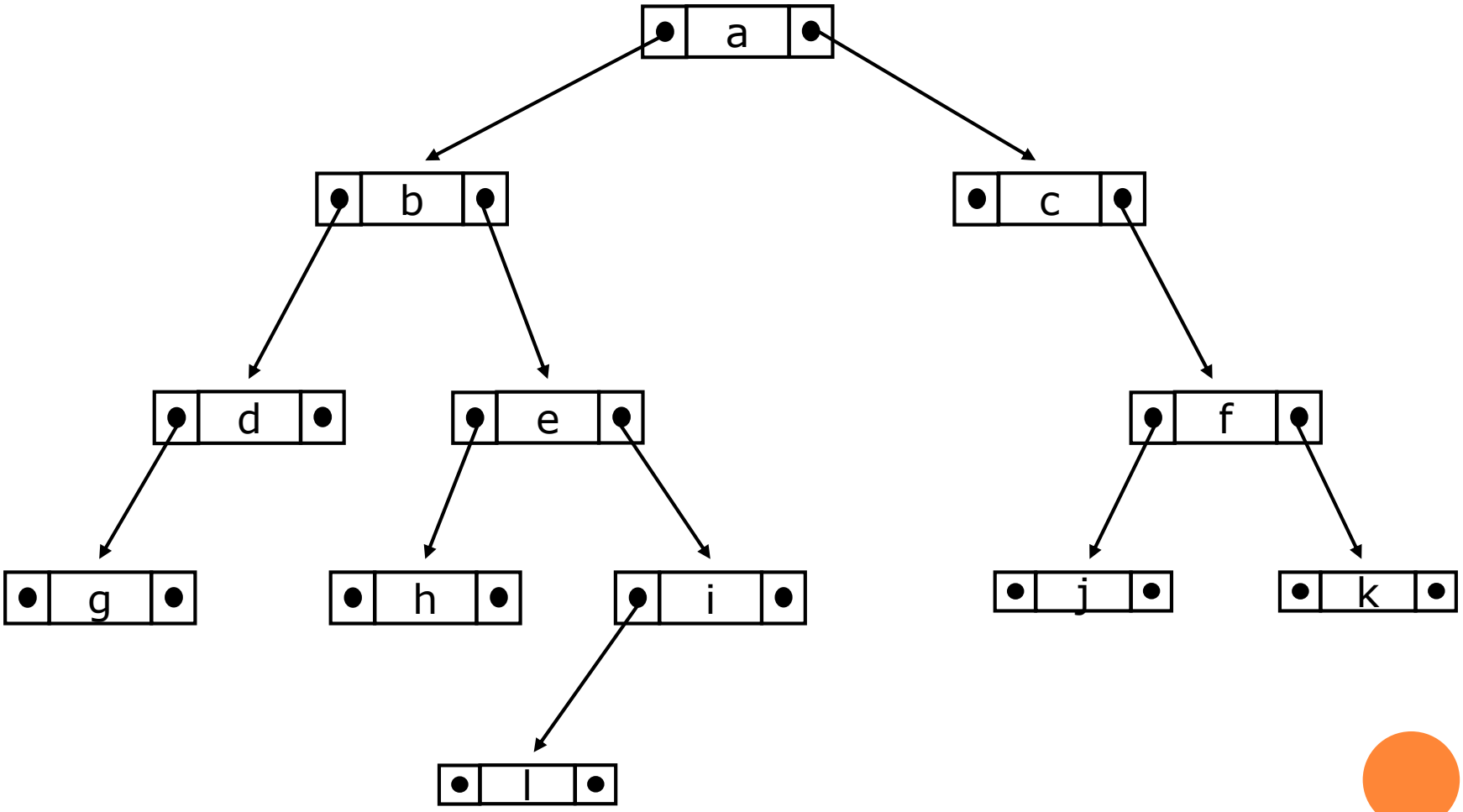
Binary Trees

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 neither a left child nor a right child is called a leaf
  - In some binary trees, only the leaves contain a value

Picture of a binary tree



Example of representation

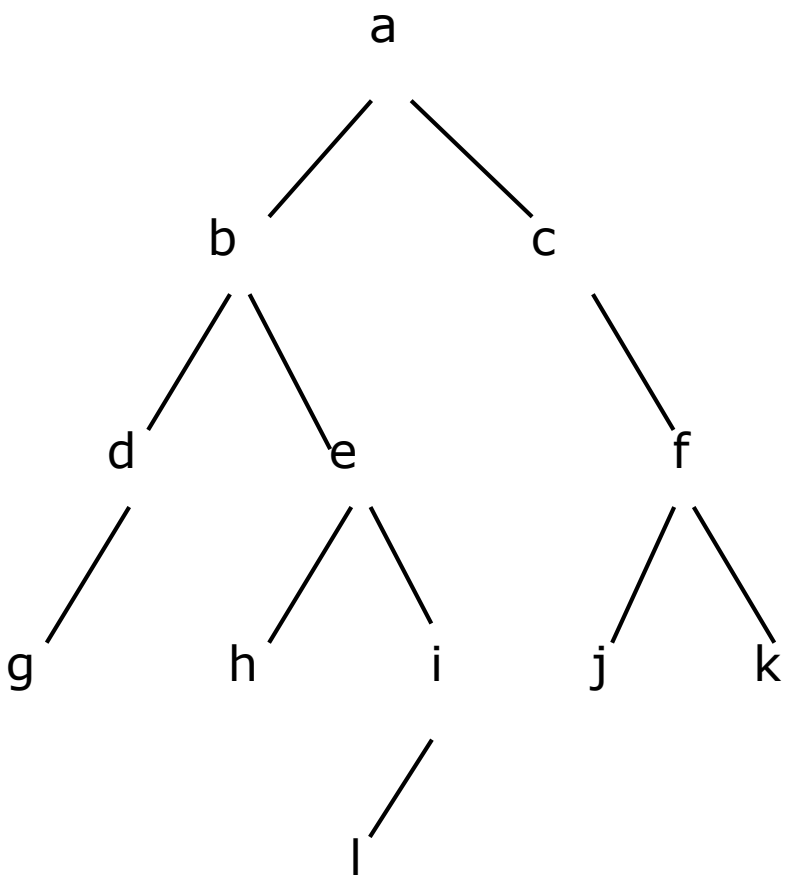
```
template <typename T> class BinaryTree
 {
    public:
        BinaryTree();
        ~BinaryTree();
    private:
        BinaryTree<T> *leftNode;
        BinaryTree<T> *rightNode;
        T *pData;
};
```

For all members of a node, you must assign memory dynamically, but not in the constructor! Allocate memory only when you need it.

BinaryTree<T> *node = new BinaryTree<T>(); delete node;

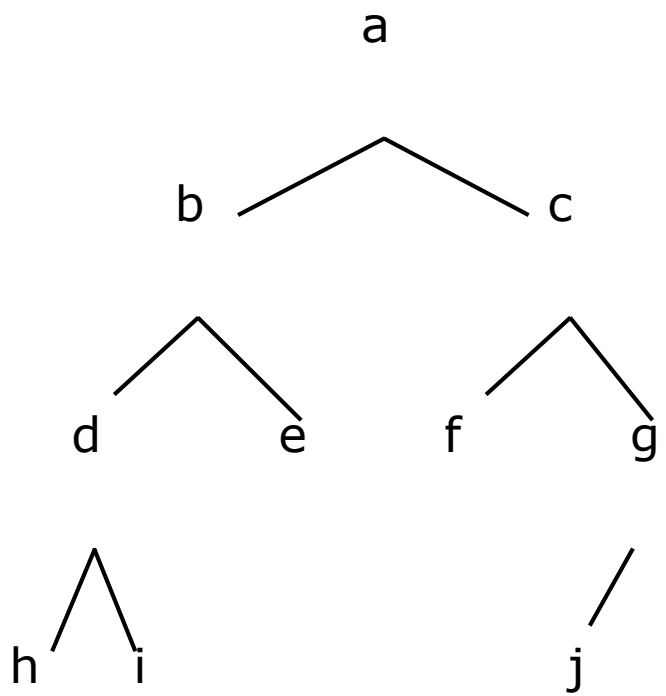
T *pData = new T; delete pData;

Size and depth

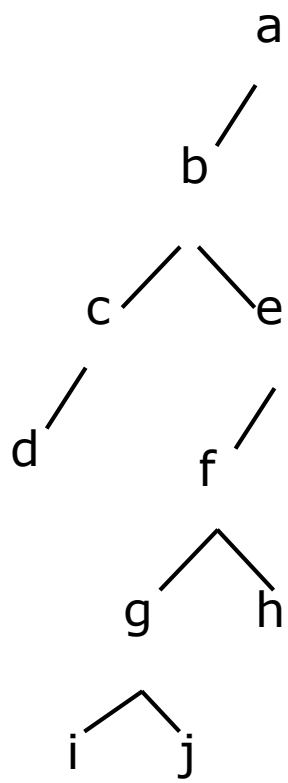


- The size of a binary tree is the number of nodes in it
  - This tree has size 12
- The depth of a node is its distance from the root
  - a is at depth zero
  - e is at depth 2
- The depth of a binary tree is the depth of its deepest node
  - This tree has depth 4

Balance



A balanced binary tree



An unbalanced binary tree

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

Tree traversals (1)

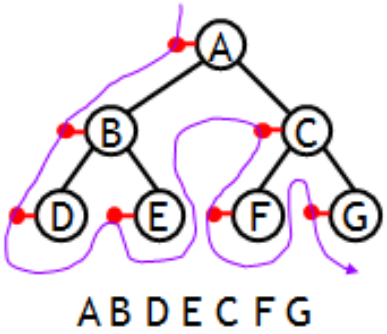
- A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree
- To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once
- Tree traversals are naturally recursive
- Since a binary tree has three "parts," there are six possible ways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

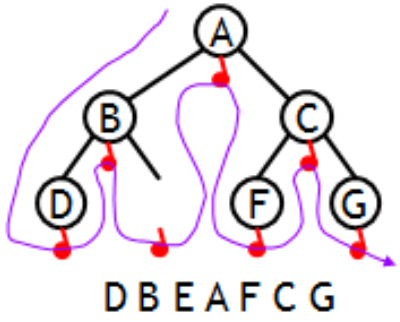
Tree traversals (2)

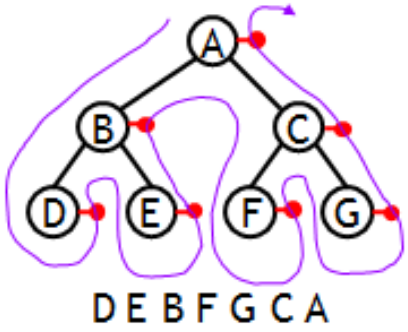
- In preorder, the root is visited *first:* root, left, right

```
PreorderTraverse(BinaryTree<T> *node) {
    Process(node->pData);
    PreorderTraverse(node->leftNode);
    PreorderTraverse(node->rightNode);
}
```

- In inorder, the root is visited *in the middle:* left, root, right
- In postorder, the root is visited *last:* left, right, root

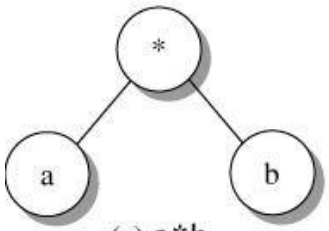




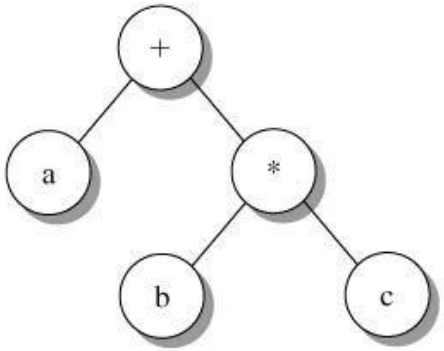


Application: binary expression tree

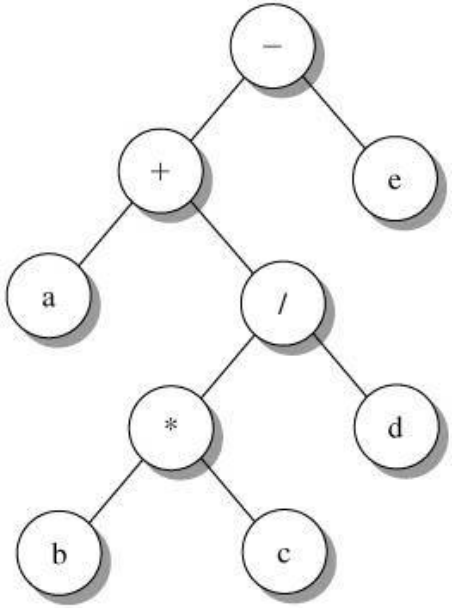
- A binary expression tree represents an arithmetic expression.
- In an expression tree each operator is an interior node whose children are operands or subexpressions. Operands are in leaf nodes.



(a) a*b

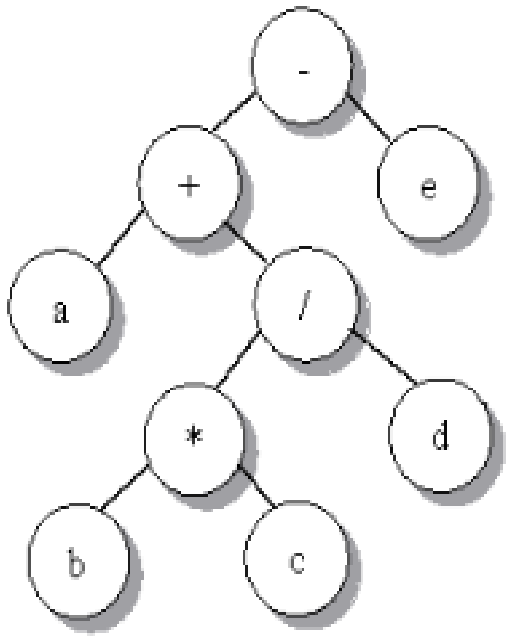


(b) a + b* c



(c) a + b*c/d − e

Application: binary expression tree



a + b*c/d - e

- **Preorder(Prefix):** - + a / * b c d e
- **Inorder(Infix):** a + b * c / d – e
- **Postorder(Postfix):** a b c * d / + e -

Evaluating Arithmetic Expressions

```
Evaluate(Node nod) {
  // If it's not a leaf...
  if (nod->left || nod->right) {
    // Evaluate expressions from sub-trees
    res1 = Evaluate(nod->left);
    res2 = Evaluate(nod->right);
    // ... and combine the results applying the operator
    return ApplyOperator(nod->op, nod->left, nod->right);
  } else {
    // If the leaf contains a variable, then we return its value
    if (nod->var)
      return Valoare(nod->var);
    else // We have a constant
    return nod->val;} }
```

Homework

- **Implement an expression tree and find the value of the expression it represents. Write on a paper an example of the algorithm (for evaluating an expression).**
- **References:**
- **http://math.hws.edu/eck/cs225/s03/binary_trees/**
- **http://www.dreamincode.net/forums/topic/37428-converting-and-evaluating-infix-postfix-and-prefix-expressions-in-c/**

In-class Assignment

- **Ex.1: Implement a binary tree in which all the nodes have integer values and all the left nodes have lower values than the parent node and the right nodes have greater values than the parent node. (Tip: you should write a criterion function for inserting new nodes)**
- **Ex2: Write a program that reads an array of integers and displays the number of occurrences of each number.**