

For Monday

- Read chapter 11, sections 1-2
- No homework

Program 8

- Any questions?

Exam 3

- Next Wednesday
- Covers linked lists through recursion

Binary trees

- Definition:
 - A binary tree t is a finite (possibly empty) collection of elements. When the binary tree is not empty, it has a root element and the remaining elements (if any) are partitioned into two binary trees, which are called the left and right subtrees of t .
- Differences from trees:
 - Each non-empty node has exactly two subtrees
 - Binary tree may be empty
 - The subtrees are ordered

Binary Expression Trees

- We can use binary trees to represent arithmetic expressions.
- Tree determines the order operations are executed in
- No need for parentheses

Binary Tree Properties

- Shares the tree properties.
- A binary tree of height h , $h \geq 0$, has at least h and at most $2^h - 1$ elements in it.
- The height of a binary tree that contains n , $n \geq 0$, elements is at most n and at least the ceiling of $\log_2(n+1)$.

Special Cases

- Full binary tree
 - A binary tree of height h that contains exactly $2^h - 1$ elements
 - In other words, if one element is added to the tree, the height must increase
- Complete binary tree

Representation of Binary Trees

- We can represent binary trees in arrays
- We don't use index 0
- The root is at index 1
- Node i 's children are at indexes $2i$ and $2i+1$
- Advantages?
- Disadvantages?

Linked Binary Trees

- A node is represented as a class
- Each node has two pointers to other nodes
- One pointer is to the Left child; the other is to the Right child
- An empty subtree is represented by a null pointer
- Sometimes it is convenient to include a pointer to the node's parent

Binary Tree Traversal

- In a binary tree **traversal**, we **visit** each node in the tree exactly once
- There are four different orders in which we often choose to visit the nodes
 - Preorder
 - Inorder
 - Postorder
 - Level order

Preorder Traversal

- ```
public static void preOrder(BinTreeNode tree)
{ // preOrder traversal of tree
 if (tree!=null) {
 Visit(tree); // visit the root
 preOrder(tree.left) // do left subtree
 preOrder(tree.right) // do right subtree
 }
}
```

# Inorder Traversal

- ```
public static void inOrder(BinTreeNode tree)
{    // inOrder traversal of tree
    if (tree != null) {
        inOrder(tree.left)        // do left subtree
        Visit(tree);              // visit the root
        inOrder(tree.right)       // do right subtree
    }
}
```

Postorder Traversal

- ```
public static void postOrder(BinTreeNode* tree)
{ // postOrder traversal of tree
 if (tree != null) {
 postOrder(tree.left) // do left subtree
 postOrder(tree.right) // do right subtree
 Visit(tree); // visit the root
 }
}
```

# Level Order Traversal

```
public static void levelOrder(BinTreeNode tree)
{ // level order traversal of tree
 Queue q;
 while(tree != null) {
 Visit(tree); // visit tree
 // put children on the queue
 if (tree.left != null) q.add(tree.left);
 if (tree.right != null) q.add(tree.right);
 // get next node to visit
 if (q.isEmpty())
 tree = null;
 else
 tree = q.delete();
 }
}
```

# Uses for Traversals