

For Friday

- Read Weiss, chapter 6, sections 1-3
- Homework:
 - Weiss, chapter 4, exercises 1-2. Make sure you do ALL of 2. Make a table with nodes down the side and the parts of the question across the top.

Programming Assignment 1

- Any questions?

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

Linked Binary Trees

- A node is represented as a struct or 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

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?

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

- ```
void PreOrder(BinTreeNode* tree)
{ // PreOrder traversal of tree
 if (tree) {
 Visit(tree); // visit the root
 PreOrder(tree->left) // do left subtree
 PreOrder(tree->right) // do right subtree
 }
}
```

# Inorder Traversal

- ```
void InOrder(BinTreeNode* tree)
{ // InOrder traversal of tree
  if (tree) {
    InOrder(tree->left) // do left subtree
    Visit(tree);       // visit the root
    InOrder(tree->right) // do right subtree
  }
}
```

Postorder Traversal

- ```
void PostOrder(BinTreeNode* tree)
{ // PostOrder traversal of tree
 if (tree) {
 PostOrder(tree->left) // do left subtree
 PostOrder(tree->right) // do right subtree
 Visit(tree); // visit the root
 }
}
```

# Level Order Traversal

```
void LevelOrder(BinTreeNode* tree)
{ // PreOrder traversal of tree
 Queue q;
 while(tree) {
 Visit(tree); // visit tree
 // put children on the queue
 if (tree->left) q.Add(tree->left);
 if (tree->right) q.Add(tree->right);
 // get next node to visit
 if (q.IsEmpty())
 tree = NULL;
 else
 tree = q.Delete();
 }
}
```