1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
Outline

1 Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2 Expression Trees

3 Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4 Assignment
Definition (Binary Tree Traversal)

To traverse a binary tree is to visit systematically every vertex in the tree exactly once.

- There are only two natural ways to traverse a list: head to tail and tail to head.
- Because a tree is nonlinear, there are many ways to traverse it.
Outline

1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
Pre-Order Traversals

Definition (Pre-Order Traversal)

A pre-order traversal of a binary tree visits the root node first, then performs a pre-order traversal of its left subtree first, and then a pre-order traversal of its right subtree. A pre-order traversal of an empty tree does nothing.
Pre-Order Traversal

```cpp
void preorderTraversal(BinaryTreeNode<T>* node,
        void (* visit)(BinaryTreeNode<T>*)) const
{
    if (node != NULL)
    {
        visit(node);
        preorderTraversal(node->left, visit);
        preorderTraversal(node->right, visit);
    }
}
```
Outline

1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
Definition (Post-Order Traversal)

A post-order traversal of a binary tree performs a post-order traversal of its left subtree first, then a post-order traversal of its right subtree, and then it visits its root node. A post-order traversal of an empty tree does nothing.
void postorderTraversal(BinaryTreeNode<T> * node, 
    void (* visit)(BinaryTreeNode<T> *)) const 
{
    if (node != NULL) 
    {
        postorderTraversal(node->left, visit);
        postorderTraversal(node->right, visit);
        visit(node);
    }
}
Outline

1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
In-Order Traversals

Definition (In-Order Traversal)

A in-order traversal of a binary tree performs a in-order traversal of its left subtree first, then visits the root node, and then performs a in-order traversal of its right subtree. A in-order traversal of an empty tree does nothing.
In-Order Traversals

**In-Order Traversal**

```c
void inorderTraversal(BinaryTreeNode<T>* node, 
    void (* visit)(BinaryTreeNode<T>*)) const 
{
    if (node != NULL)
    {
        inorderTraversal(node->left, visit);
        visit(node);
        inorderTraversal(node->right, visit);
    }
}
```
Outline

1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
Definition (Expression Tree)

A binary expression tree is a binary tree that is used to represent an expressions with binary operators. Each interior node represents an operator. At each interior node, the left subtree represents the left operand and the right subtree represents the right operand.

- As a consequence, each terminal node represents an operand.
- The order of operation is indicated by the structure of the tree.
For example, \((3 + 4) \times 5\) may be represented as:

```
3 + 4
```

\[ * \\
\]

3 4

5
If there is more than one operator, the order of operation is indicated by the structure of the tree.
Perform a pre-order traversal of the expression tree and print the nodes.
Perform an in-order traversal of the expression tree and print the nodes.
Perform a post-order traversal of the expression tree and print the nodes.
A post-order traversal is used to evaluate the expression.
1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
Definition (Binary Search Tree)

A binary search tree is a binary tree with the following properties.

- There is a total order relation on the members in the tree.
- At every node, every member of the left subtree is less than or equal to the node value.
- At every node, every member of the right subtree is greater than or equal to the node value.
A Binary Search Tree
The BinarySearchTree class is implemented as a subclass of the BinaryTree class.
Mutators

```cpp
void insert(const T& value);
void remove(const T& value);
```

- **insert()** – Insert a new node containing the value into the binary search tree.
- **remove()** – Remove the node containing the value from the binary search tree.
Binary Search Tree Interface

Other Member Functions

```c
T* search(const T& value) const;
void countBalance();
```

- `search()` – Search the binary search tree for the value. Return a pointer to the node where the value is found. Return `NULL` if the value is not found.
- `countBalance()` – Count-balance the binary search tree.
1 Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2 Expression Trees

3 Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4 Assignment
Searching a Binary Search Tree

- Beginning at the root node, apply the following steps recursively.
  - Compare the value to the node data.
  - If it is equal, you are done.
  - If it is less, search the left subtree.
  - If it is greater, search the right subtree.
  - If the subtree is empty, the value is not in the tree.
Outline

1 Binary Tree Traversals
   • Pre-Order Traversals
   • Post-Order Traversals
   • In-Order Traversals

2 Expression Trees

3 Binary Search Trees
   • Searching a BST
   • Inserting into a BST
   • Deleting from a BST
   • Count-Balancing a BST

4 Assignment
Inserting into a Binary Search Tree

- Beginning at the root node, apply the following steps recursively.
  - Compare the value to the node data.
  - If it is less (or equal), continue recursively with the left subtree.
  - If it is greater, continue recursively with the right subtree.
  - When the subtree is empty, attach the node as a subtree.
Outline

1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
Deleting a Value from a BinarySearchTree

- Perform a search to locate the value.
- This node will have
  - Two children, or
  - One child, or
  - No child.
A Binary Search Tree

100

50

30
40

80
90

20
40
60
90

10
70

110

130

120
140
Deleting a Value from a Binary Search Tree

Case 1: No Child
- Delete the node.
Delete a Node with No Child
Delete a Node with No Child

Binary Tree Applications
Case 2: One Child

- Replace the node with the subtree of which the child is the root.
Delete a Node with Two Children

Robb T. Koether  (Hampden-Sydney College)  Binary Tree Applications  Wed, Apr 15, 2015  37 / 56
Delete a Node with Two Children
Delete a Node with Two Children
Case 3: Two Children

- Locate the next smaller value in the tree. This value is the rightmost value of the left subtree.
  - Move left one step.
  - Move right as far as possible.
- Swap this value with the value to be deleted.
- The node to be deleted now has at most one child.
Delete a Node with Three Children
Delete a Node with Three Children
Delete a Node with Three Children

![Binary Tree Diagram]

- Node 50 is deleted.
- The right child of node 30 (node 50) is replaced by the left child of node 60 (node 30).

Robb T. Koether (Hampden-Sydney College)  Binary Tree Applications  Wed, Apr 15, 2015  43 / 56
Delete a Node with Three Children
Delete a Node with Three Children

```
    100
   /   \
  60    110
 / \
30  80  130
/ \
20 40 90 120
/  \
10 70
```

Robb T. Koether (Hampden-Sydney College)
1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
Write a function `moveNodeRight()` that will move the largest value of the left subtree to the right subtree.

The `moveNodeRight()` Function

- Locate the largest value in the left subtree.
- Delete it (but save the value).
- Place it at the root.
- Insert the old root value into the right subtree.
A Binary Search Tree
A Binary Search Tree

Robb T. Koether (Hampden-Sydney College)
A Binary Search Tree
A Binary Search Tree

- Example binary search tree diagram
  - Root: 90
  - Left subtree:
    - Root: 50
    - Left subtree:
      - Root: 30
      - Left subtree:
        - Root: 20
      - Right subtree:
        - Root: 40
    - Right subtree:
      - Root: 80
      - Left subtree:
        - Root: 60
      - Right subtree:
        - Root: 70
  - Right subtree:
    - Root: 110
    - Left subtree:
      - Root: 100
    - Right subtree:
      - Root: 130
      - Left subtree:
        - Root: 120
      - Right subtree:
        - Root: 140

- Key points:
  - Binary search trees are self-balancing trees that maintain a search order.
  - Each node has at most two children, with the left child being less than the parent and the right child being greater.
  - They are used to efficiently search, insert, and delete elements.
Count-Balancing a BinarySearchTree

Count-Balancing a Tree

- Write a similar function `moveNodeLeft()`.  
- Apply either `moveNodeRight()` or `moveNodeLeft()` repeatedly at the root node until the tree is balanced at the root.  
- Then apply these functions recursively, down to the leaves.
Suppose we wish to transmit the nodes of a balanced binary search tree to another computer and reconstruct the tree there. In what order should the values be transmitted?
We could use an in-order traversal to transmit them.

At the receiving end, simply call `insert()` to insert each value into the tree.

The constructed tree will be identical to the original.

What do we get if we transmit the values using a pre-order traversal?

Using a post-order traversal?
Outline

1. Binary Tree Traversals
   - Pre-Order Traversals
   - Post-Order Traversals
   - In-Order Traversals

2. Expression Trees

3. Binary Search Trees
   - Searching a BST
   - Inserting into a BST
   - Deleting from a BST
   - Count-Balancing a BST

4. Assignment
Assignment

- Read Section 19.2.