What is a binary tree. A **binary tree** is a tree data structure in which each node has at most two children. Typically the child nodes are called *left* and *right*. Binary trees are commonly used to implement binary search trees and binary heaps. Each child node to the left has a value less than its parent node. And each child node to the right has a value equal to or greater than its parent node. Binary trees can store duplicate values, but it is not recommended because one would need a second "tie breaker" value to determine whether the node references the correct value or not.

A **balanced binary tree** is where the depth of all the children differs by at most one level. Balanced trees have a predictable depth (how many nodes are traversed from the root to a child). Searching a balanced binary tree is very fast, O(log n). In VB, use this formula: Log(TreeHeight)/Log(2). So, how fast is fast? A tree containing one million nodes can be completely searched in less than 20 comparisons!

There are several varieties of binary trees. A sorted list of values added to a binary tree is also known as a linked list. Every node has just one child. When a binary tree becomes a linked list, it is called a degenerate tree. A degenerate tree search time is exponentially greater than a balanced tree. Searching for the 256th node (final node) in a linked list requires comparing all 256 nodes whereas searching in a balanced tree is no more than 8 comparisons (2^8=256).

So how does one prevent a binary tree from becoming a degenerate tree? There are many options really. Some of them include, adding nodes in random order which produces a good tree, but unbalanced. However, getting nodes added in random order is rarely possible since one does not generally have control over the data/keys they will have. Other options include Red-Black trees, AVL trees, threaded binary trees, and many more. The method used in this project uses the simplest form of a binary tree, adding only a "balance factor" flag to each node to indicate when the tree falls out of balance so that rebalancing can occur immediately.

The data structure consists of: Key, Value, BalanceFactor, LeftChld, RightChild. The LeftChild & RightChild are references to other nodes. BalanceFactor is a value that ranges from -2 to +2. Key is a string value that uniquely identifies a node. The Value member can be string, numeric, variant, or pointer to some other structure(s). When the nodes will only contain strings or numerical data; the Key or Value can be excluded from the structure.

Balance factors. Whenever a node's lowest subtree/children are at equal levels, the balance factor will be zero.
When a node has one more level on the left side, its balance factor is 1 and -1 if one more on the right side.
By adding one for a new left level and subtracting one for a new right level, the tree only becomes unbalanced when
one of the node's balance factor becomes 2 or -2, which indicates one side of the node has 2 more levels than the other. Balanced binary trees can never have a difference greater than one level. During updating the balance factors, the propogation up the tree stops when one of two conditions are met: 1) The node propogated to is already balanced, or 2) the tree is unbalanced, then rebalanced; therefore, making that node balanced.

Some Real World Uses of Binary Trees, just a few examples
1. Fast searching for databases, dictionaries, and collections for example. A binary tree is searched based on a value which could be a word, a numerical value, or maybe a class property.
2. Genetic and ancestory data describing lineage, organizational charts.
3. Cryptography, Huffman encoding

4. File systems
5. Algebraic expressions (binary expression tree) that use words like "Add", "Subtract", "Divide", etc.
Ref: http://msdn.microsoft.com/en-us/library/aa227586.aspx

References: Here are some links I used when researching this project:
http://msdn.microsoft.com/en-us/library/ms379572.aspx
http://www.mactech.com/articles/mactech/Vol.06/06.08/BinaryTrees/index.html
http://en.wikipedia.org/wiki/Binary_tree
http://upload.wikimedia.org/wikipedia/en/c/c4/Tree_Rebalancing.gif

INSERTION REBALANCING ALGORITHM
There are 4 specific cases to be handled when insertion occurs. Two are simply a reverse case of the others, so we
will look at insertion on the left side of the tree only. Insertion on the right side is same algorithm, but reversed.

Case 1: Left Left Insertion
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000