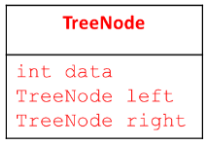
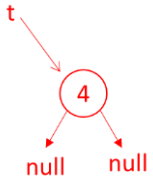
Reference-based Representation

What does a Binary Tree node look like?

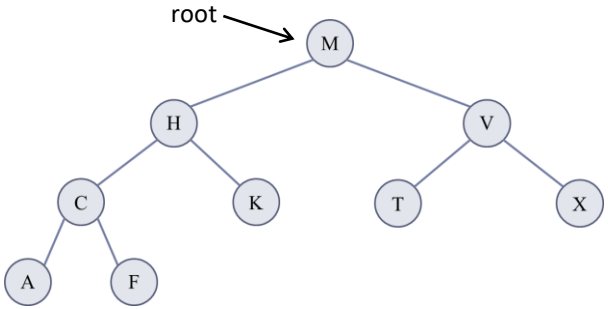


TreeNode t = new TreeNode(4);
can be visualized as:



Similar to a linked list, tree node's should have a data field and then references to other nodes. Instead of next/prev, in a tree there are references to the left and right child. Some implementation may also have a parent reference too.

Using what we know from linked-lists, how can we traverse a tree?



Observation:

With our definition of a TreeNode above, every node in the tree is reachable from the root

(In this particular tree, node's are alphabetically ordered too – letter's that come before a node are in its left subtree, letters than come after are in its right subtree.)

Starting from root, how do we access T:

```
Node cur = root;
cur = cur.right;
cur = cur.left;
```

Starting from root, how do we access F:

```
Node cur2 = root.left.left.right;
```

How can we add a node to an existing tree at a certain location?

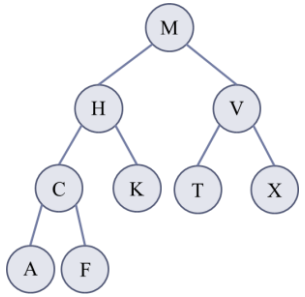
```
TreeNode n = new TreeNode("L");
TreeNode cur = root;
cur = cur.left;
cur = cur.right;
cur.right = n;
```

Array-based representation

What would an array look like representing the same Binary Tree?

For a tree of ints
int[] data;
int numElements;

It is very likely there will be gaps in the array!

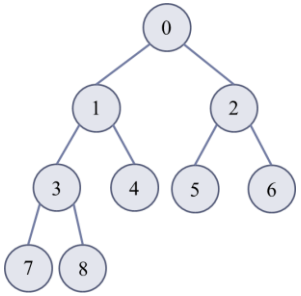


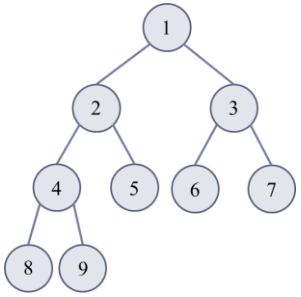
Two methods:

M	H	V	C	K	T	X	A	F						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

-	M	H	V	C	K	T	X	A	F					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Viewing the tree by the index number of each item:





Left child: $2(i) + 1$
Right child: $2(i) + 2$
Parent: $\lfloor (i-1)/2 \rfloor$

Left child: $2i$
Right child: $2(i) + 1$
Parent: $\lfloor i/2 \rfloor$

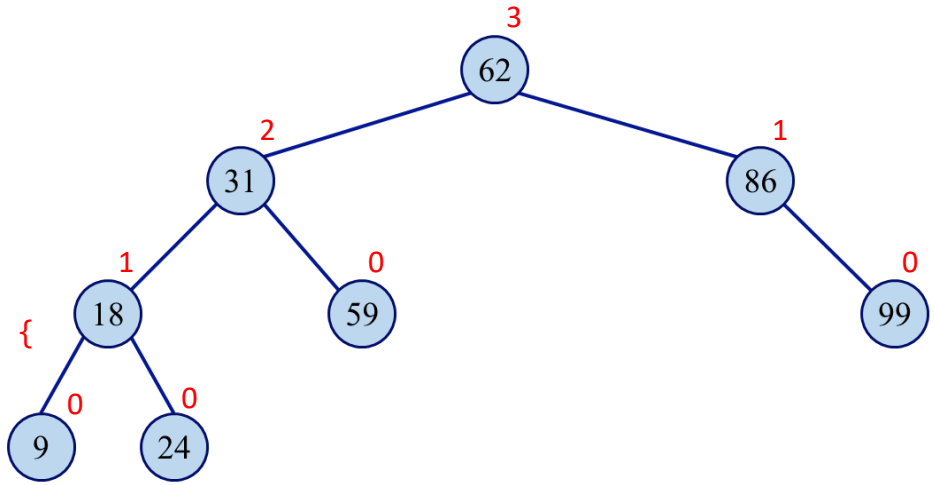
What index would we insert the value **U** so that it was **T**'s right child?

0-based index: $2(5) + 2 = 12$

1-based index: $2(6) + 1 = 13$

Programming Exercise:

Write a recursive method that, given a node, returns the height of that node. Assume that the height of a tree with just a single root node is 0.




```
public int height(TreeNode cur) {
      if (cur == null) {
            return -1;
      } else if (height(cur.left) > height(cur.right) {
            return 1 + height(cur.left);
      } else {
            return 1 + height(cur.right);
      }
}

// shorter implementation:
public int height(TreeNode cur) {
      if (cur == null) {
            return -1;
      }

      return 1 + max(height(cur.left), height(cur.right));
}

// a max function is trivial:
public static int max(int a, int b) {
      if (a > b) {
            return a;
      } else {
            return b;
      }
}
```

Inside the BinaryTree class, we could call our recursive height method from the height method, and give it the root as a parameter to calculate the height of the whole tree

```
public int height() {
      return height(root);
}
```