CSC 2111 **Spring 2013**

Name(s): ________________________________

Lab 11: Binary Search Trees

Reference: *C++ Plus Data Structures*
Chapter 8 - "Binary Search Trees"

Objective: To gain a better understanding of how binary search trees work and continue to work with the concept of recursion.

Files: *treetester_win*

Introduction

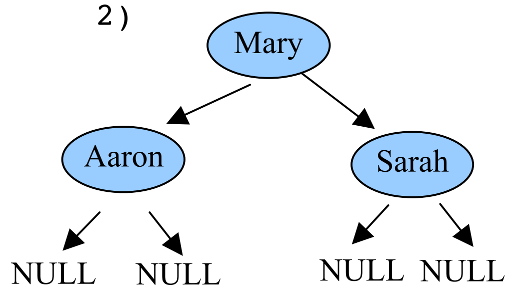
Tree is a recursive data structure with a simple definition. The definition is as follows:

A tree is either empty or consists of a root element having n children which are trees.

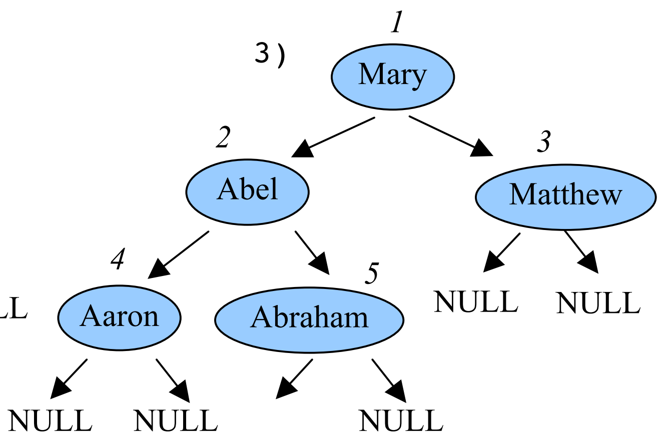
Given the above definition, here are three examples of trees:

1) NULL

2)



3)



In the above examples, the first is a tree that is empty. The second is a tree with a root element containing "Mary". That root has two children that are trees. The left child is rooted at the element containing "Aaron", and the right child is rooted at the element containing "Sarah". In the third example, root of the left subtree ("Abel") has two children, while the root of the right subtree ("Matthew") does not have any.

Note that for the above trees, each node has zero, one or two children. These trees are a special case of trees in general. They are called **binary trees** because each node is binary (can have, at most, two children). Also, you can see that some nodes in the above examples have no children (i.e. NULL children). These nodes are referred to as *leaf* nodes.

Properties of binary trees

Binary trees have some good properties. For example, consider the array of 5 names:

Mary	Aaron	Matthew	Abraham	Abel

Note that the array has the same names as the binary tree #3 on the previous page. How many comparisons must we do to find the name Abel? With the array-based container, we have to do 5 comparisons to find “Abel”. However, notice that the following holds true for the binary tree in example #3:

> *For every node* n *in the tree, every node in the left subtree of* n *has a key that is less than* n*'s key and every node in the right subtree of* n *has a key value greater than or equal to* n*'s key.*

Such a tree is termed a **binary search tree**. Searching a binary search tree can be more efficient than linearly searching an array, if we apply the following algorithm:

```
search(root, mykey)

if root is NULL return false        // we did not find mykey
if root's key is equal to mykey
     return true
if mykey is less than root’s key
     return search(root.left, mykey)
else
     return search(root.right, mykey)
```

The definition of the search algorithm is *recursive*. In English terms, we check the root node. If the key is found in the root node, then we are done. If not, then the key we are searching for is either less than the key in the root node, or it is greater than or equal to the key in the root node. If it is less than the key in the root node, it must be in the left subtree of the root (if it exists at all). However, if it is greater than or equal to the key in the root node, then it must be in the right subtree (or not in the tree at all). Notice that at each node, we can disregard a whole subtree when doing comparisons!

Pretend that you are searching for the key “Abraham” in binary tree #3 (previous page). Apply the `search` algorithm above. Using the number depicted above the nodes(see the tree on the previous page) to represent `root` in the algorithm, **show the recursion as you did in Lab #9:**

search(1, “Abraham”)   return
                                        search(______, “Abraham”)

How many comparisons did it take before you found Abraham?

Apply the above recursive algorithm to find the key "Abram":

What does the search algorithm return?

Why?

Implementation

How do we implement a Binary Search Tree, or BST, in C++ code?

First we have to define a C++ BST node. A BST node must have a data element, and two pointers that represent the left and right subtree. These two pointers must be self-referencing pointers similar to the "next" pointer that you defined for your `LinkedList` class in a previous assignment, but instead of being pointers to a linked list node, they must be pointers to a BST node.

In the space provided below, **write C++ code that defines a BST node**:

```
template<class T>
classBSTNode
{
     private:
          // put the data element:


          // Now for the left subtree:


          // Now for the right subtree:


     friend class BST;
};
```