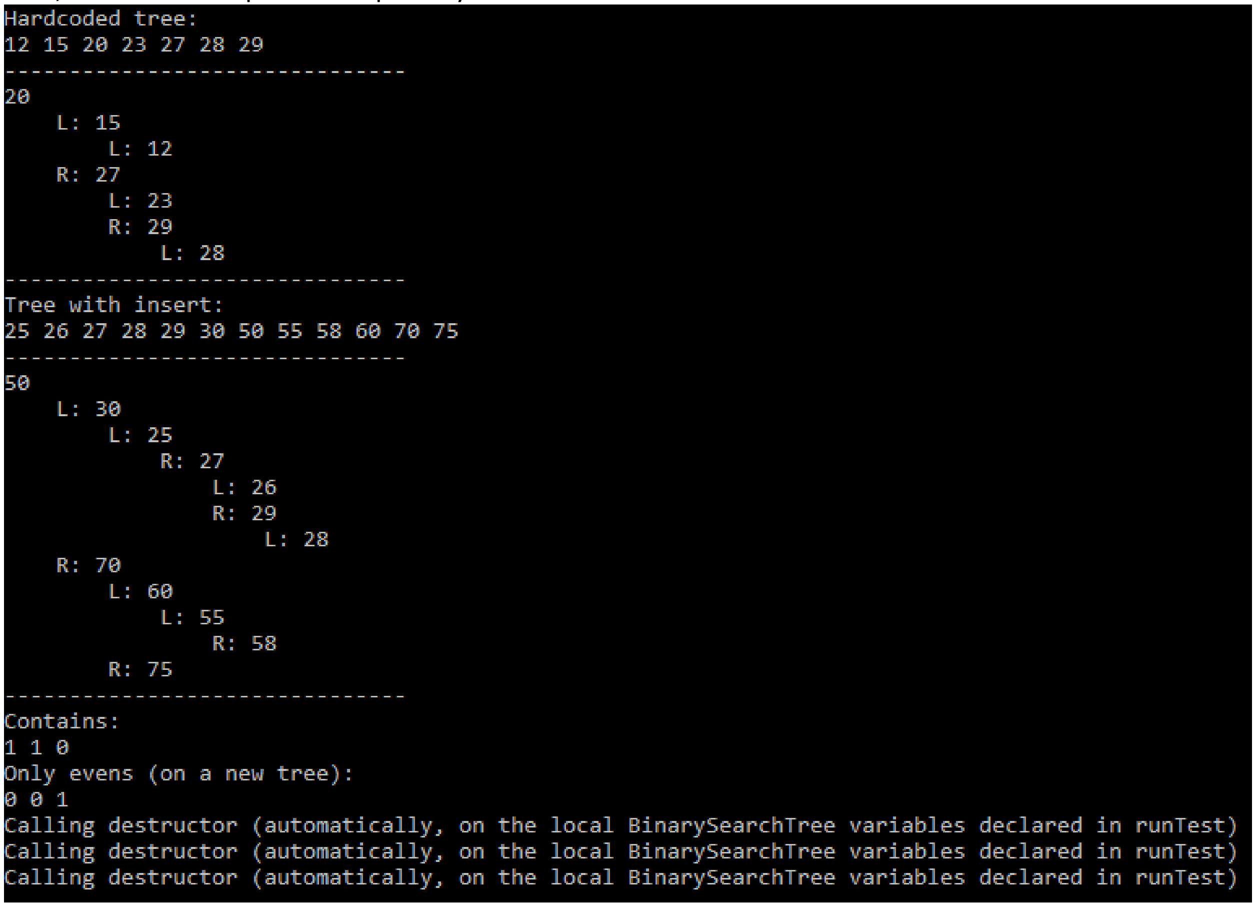
CSC 122
hw6 – Binary Search Trees
Due: Monday 4/27 at 8:00 AM EDT

In this assignment, you'll implement a BinarySearchTree class to work on ints.

First, note the code provided in the "hw6 Given Code.txt" file. Make a new Visual Studio project. Put the runTest and main functions in a file called main.cpp. Put the remaining functions in a file called BinarySearchTree.cpp.

You'll need to complete BinarySearchTree.cpp, BinarySearchTree.h, and IntBTNode.h for this assignment.

First, observe the expected output as you look at the runTest function:

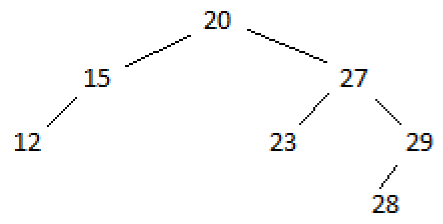


```
Hardcoded tree:
12 15 20 23 27 28 29
-------------------------------
20
    L: 15
        L: 12
    R: 27
        L: 23
        R: 29
            L: 28
-------------------------------
Tree with insert:
25 26 27 28 29 30 50 55 58 60 70 75
-------------------------------
50
    L: 30
        L: 25
            R: 27
                L: 26
                R: 29
                    L: 28
    R: 70
        L: 60
            L: 55
                R: 58
        R: 75
-------------------------------
Contains:
1 1 0
Only evens (on a new tree):
0 0 1
Calling destructor (automatically, on the local BinarySearchTree variables declared in runTest)
Calling destructor (automatically, on the local BinarySearchTree variables declared in runTest)
Calling destructor (automatically, on the local BinarySearchTree variables declared in runTest)
```

Provided methods:

- **makeHardcodedTree** – A "hardcoded tree" is generated by this method. I provided this method so that you can test your work as you go, even if you don't yet have a working insert method.
- **print** – I provided this method so that you can see what a tree looks like in a somewhat graphical way while you're debugging.
    - printHelper and printSpaces should be private methods, since they're only used by print.

    - Look at the output above, and see how the first tree corresponds to this visual representation:



- **~BinarySearchTree** – This is the destructor, a kind of method you may recall seeing for the first time in the linear structure homework assignment. It recursively moves through the tree and deletes the dynamically-allocated memory.
    - destructorHelper should be a private method since it's used only by the destructor.

If you get an error in any of the above methods, it means there is an error in your code – most likely an error in how your tree is built.

Methods you'll need to write:

- **printInOrder** – does a recursive in-order traversal of the tree. Since we're working with binary *search* trees, this should print the numbers in increasing order.
- **insert** – puts the provided value at the correct place in the tree.
    - To keep things simple, let's just assume that there will be no duplicate values (you don't have to check for this) so that there's no ambiguity.
    - This will be similar to the function version we saw in class. For practice, please do this recursively. Look at how I handled root as the first value of curr in ~BinarySearchTree, and design this code similarly.
- **contains** – returns true if the tree contains the given argument, false otherwise.
    - Assume you have a properly formed binary *search* tree when you write this.
    - This is only a little different from another function we saw in class. Again, please do this recursively, and look at the destructor for a hint about how to initialize curr.
- **onlyEvens** – returns true if there are only even numbers in the tree, false otherwise.
    - This is a new problem. Again, please do this recursively, and look at the destructor for a hint about how to initialize curr.

Other notes:

- For the methods you need to write, you'll also need to write some helper methods. Make sure these are all private, since they shouldn't be called outside of the class.
- When you're testing, put whatever code you want in your main function, to test whatever you'd like. When you turn this in, though, please delete the extra code in the main function, and only call runTest. Please do not change runTest at all.
- Recall that in C++, when you print a bool value, it prints 1 for true, and 0 for false.
- Every node you create for this assignment should be dynamically allocated.