COMP241
Fall 2015
Programming Assignment 4
Assigned: 12 November, 2015
Deadline: 19 November, 2015 (early deadline: 17 November, 2015) at the beginning of class

Submission guidelines:

- Please post your solution on Moodle. It is due ***at the beginning of class.***
- +5 extra credit if you turn it in on November 17th (also at the beginning of class)
- Use MS Visual Studio 2012 for the first two problems. Submit your source code files (.cpp and .h files) separately for each problem.
- Include a comment with the course number, term, assignment, your name, and the date, at the top of each code file.
- For problem 3, you'll submit a text file rather than source code -- the result of running some experiments. This should go on moodle also.
- Although I encourage collaboration if you want to work with a classmate, you must turn in your own work. Duplicate code and plagiarism will be considered a violation of the honor code and handled accordingly.

Problem 1
35 points
Write a class to represent a binary search tree of integers. Assume an underlying linked structure. You don't need to capture every single operation that binary trees can do, but you'll need to write at least two functions: (1) insert an element into the tree, (2) do a pre-order traversal of the tree.

Once your class is ready, use your driver to create a tree object and populate with integers that you repeatedly prompt the user for. This prompt should be done in a loop, stopping only when the user indicates that they don't want to add any more.

Insert each of the values the user gives you into the tree object, using your insert function.

Finally, print the tree to the terminal window by doing a pre-order traversal of the tree.

Problem 2
25 points
On gdrive, you'll find two text files representing trees: Tree 1 (http://bit.ly/1Q6ZVGu) and Tree 2 (http://bit.ly/1kcSkt3).

Write a program to read in each of these files, assuming the file format we discussed in class. Print to the terminal: (1) whether the file in question contains a valid binary search tree; (2) whether the file in question contains a valid binary tree, and (3) the height of the tree.

Problem 3
40 points
This problem will be experimental. You will implement both recursive versions of a factorial function ("regular" recursion and tail recursion), and you will determine whether your compiler optimizes tail recursion in any discernible way.

Implement both factorial versions (below for reference) and call them both with increasing values of *n*. Your input *n* should start at 1 or 0 for both functions, and increase in increments.

You'll need to calculate the wall-clock time (execution time) of running each factorial version. This can be tricky, because an ideal time-performance comparison would usually involve a dedicated machine or 5 that runs absolutely nothing but your program. For the purposes of this experiment, simply close down all non-C++ programs to get the best estimation you can.

Never calculated the wall-clock time of a C++ program execution before? Start here to get that code going.

Make sure you turn on optimization in your compiler, so that it *can* take advantage of tail recursion if it wants to. You can use any C++ compiler, it doesn't have to be Visual Studio. The way to turn on optimization varies; look here and here .

For this problem, you don't need to submit your code; instead, submit a file on moodle that makes an argument about whether tail recursion makes a difference. Your submission file should include:

- The takeaway: Does your compiler optimize tail recursion? Did it make a significant difference in execution time?
- The way that you turned on optimization and which compiler you tested on
- The values of n you tested for each version of the function
- The wall-clock time you saw for each value of n (or a graph showing how the wall-clock time grows; you don't need to list every single value of n if you have a ton of them)

Version #1: Regular recursion

```
int factorial(int n) {
      if(n == 1)
            return 1;
      else
            return n * factorial(n-1)
}
```

Version #2: Tail recursion

```
int factorial(int n, int a) {
      if(nr == 1) {
            return a;
       else
      return factorial(n - 1, A * n);}
```