

# Homework 3

Submit Assignment

**Due** Friday by 11:55pm      **Points** 100      **Submitting** a file upload      **File Types** zip  
**Available** Feb 18 at 12am - Mar 8 at 11:55pm 20 days

## COMS W3134 Spring 2016 (Sections 1 and 2)

### Homework 3

**Due: 4:00pm on Friday, March 4th**

#### Written (35 pts)

For the written section of this assignment, type up your answers and submit a computer based document to us. You can submit MS Word doc files, pdf files, or txt files.

1. Problem 1 (6 pts): Weiss 4.6 - We're looking for a full induction proof
2. Problem 2 (6 pts): Weiss 4.9 - In part a show the tree after each insert - a total of 8 trees. In part b, we're doing full deletion, not lazy deletion.
3. Problem 3 (6 pts): Show the same sequence of inserts as listed in the previous problem, except this time show the insertions into an AVL tree. Make sure to show each rotation.
4. Problem 4 (6 pts): Prove that a preorder and postorder traversal of a binary tree is not necessarily sufficient to reconstruct the tree uniquely.
5. Problem 5 (6 pts): Based on Weiss 4.16 - Describe any modifications that you would need to make to the BinaryNode itself, and then show the implementation for findMin. You don't actually have to give us a full working class.

#### Programming (65 pts)

For the programming portion of the assignment, please submit only your .java files and your README.txt. Your code should be well commented. In addition please include a detailed README.txt file which indicates exactly what files you are submitting, what problem they solve, and how to compile and run them.

1. (35 pts) Implementing Expression Trees:

Implement a class called `ExpressionTree`. The constructor to `ExpressionTree` will take in a `String` that contains a postfix expression. The operands will be integers and the binary operators will be restricted to `+`, `-`, `*`, and divide. Individual tokens, that is the operands and operators, will be delimited by spaces. So for example:

`34 2 - 5 *`




would mean  $(34-2)*5$ .

Your constructor will run the stack based algorithm we discussed in class to build an expression tree. `ExpressionNodes` will be a nested class within `ExpressionTree`. You may use any code posted on canvas or from the Weiss textbook as a starting point for this class.

Once you have the `ExpressionTree` constructed you should provide the following four methods:

1. `public int eval()` - this method, when invoked on an expression tree object will return the integer value associated with evaluating the expression tree. It will need to call a private recursive method that takes in the root.
2. `public String postfix()` - this method, when invoked on an expression tree object will return a `String` that contains the corresponding postfix expression. It will need to call a private recursive method that takes in the root.
3. `public String prefix()` - this method, when invoked on an expression tree object will return a `String` that contains the corresponding prefix expression. It will need to call a private recursive method that takes in the root.
4. `public String infix()` - this method, when invoked on an expression tree object will return a `String` that contains the corresponding **correct** infix expression. Keep in mind that parenthesis will be needed (excessive parenthesis will be tolerated as long as it is correct). It will need to call a private recursive method that takes in the root.

Write a class called `Problem1.java` that instantiates an expression, and demonstrate your methods.

2. (30 pts) Write a command line application, in the file `Problem2.java` that indexes the words contained in a text file (provided to the program as a command line argument). Your program should go through the input file line by line. For each line, extract each word, and insert that word, along with it's location into an AVL tree. Each element of the AVL tree should contain a unique word and a linked list of line numbers where that word occurs. If word already exists in the AVL Tree, simply add the new line number to the existing node. If a word appears on the same line twice, it should only have one entry in the list for that line. When you have finished, print out each unique word that appeared in the input file along with a list of line numbers on which that word appears. You may use Weiss' AVL tree code (available from the textbook website and attached [here](#)  [↗](#) - also note you must include the Weiss' `UnderFlowException` class, which is attached [here](#)  [↗](#)) as a starting point for your program, you will need to modify it. Ignore case (insert everything as lower case), and strip out all punctuation. You may test your file on the [attached text file 2016su.txt](#)  [↗](#).

#### What you need to submit:

On your home system make a submit directory for your submission. The directory's name should include your name or UNI and homework 3. For example, I might name my submit directory: `psb15homework3`. Copy **only** the files that you wish to submit into that directory. **DO NOT SUBMIT CLASS FILES**, only submit the source java files, the README.txt and any written documentation we required. We will compile the programs for you.

1. written.txt

2. ExpressionTree.java
3. MyStack.java
4. Problem1.java
5. AvlTree.java
6. UnderFlowException.java
7. Problem2.java
8. README.txt

Once all the files are in place, make a zip file out of that directory. Call it something like psb15@homework3.zip. Make a copy of the zip file and then unpack it to confirm that the files are actually there.

Login to courseworks and go to the "Assignments" section. Click on Homework 3 and then follow the instructions to upload the zip file that you created above.

Resubmissions are allowed but you must resubmit the entire assignment each time. Only the last submission will be graded.