SE 461 – Spring 2021
Assignment #4- Tree Calculator

Due: 4/16/2021

In Assignment 3, you used a Stack in order to build a simple expression evaluator, a calculator if you will. For this assignment, you will complete the second implementation of the basic expression evaluator using a Tree structure instead of the Stack to accomplish your goal. You will once again use the Infix to Postfix algorithm, that was discussed in lecture, to achieve this evaluation. Moreover, to improve the design of your expression evaluator, you will need to use the **Composite Pattern** to capture the structure of the expression, the **Builder Pattern** to create the expression tree, and the **Visitor Pattern** to evaluate the expression tree. We will discuss each of these patterns and their use in lecture – references can also be found in your textbook (GoF book).

Development Process:

For this assignment, all development must take place on the **master branch**. It is strongly recommended that you commit and push often!

You are to create a complete program. The expression evaluator must be able to handle the following operators/tokens: +, -, /, *, %, (,), integers (positive and negative). All expression will have a space between each token to simplify parsing. All input should come from the `STDIN`. The program must loop until the user types **QUIT**—notice the all caps. All output should go to `STDOUT`.

The assignment will be graded (compiled/run) on Thomas – please ensure that your code executes properly in this environment prior to submission.

All of the same requirements from Assignment #3 still hold.

Submission:

All assignments must be submitted on Butler GitHub (github.butler.edu). The name of your Butler GitHub repository must be as follows: **se461_spring2021_tree**. Your repo **must** be private with myself (rrybarcz) added as a collaborator.

Each source file (.cpp/.h/.inl) **must** include the Honor Pledge and Digital Signature – failure to do so will result in a loss of points.

For this assignment I am not providing you a list of files that are required – you will need to decide this on your own. Obviously, you could accomplish this in a single class file – but as we know that would not be a good design choice – please don't try it! You should make use of the "good practices" and concepts that we have covered in lecture and apply the patterns in an appropriate fashion. You are expected to submit any and all source/header/template files that are required to run your program – you are also required to submit an MPC file and valid Valgrind.txt file with your submission on GitHub. Please make sure that your repository is clean and does not contain any unneeded files. Failure to do so will result in a loss of points