Week 5: Objects and Operator Overloading

This week, we will be building a simple calculator program and briefly explore a software development cycle. Your calculator will take math expressions like "1+5" and output the correct answer, "6". But order of operations makes things complicated – we have to make sure "1+2*3" outputs "7" and not "9"! Luckily, we'll ignore parentheses and negative signs. If you're interested, know that there is an algorithm for evaluating complex expressions such as "((((1+3)sin(45*tan(-3)))/23)+234)" and you can read about it here: https://en.wikipedia.org/wiki/Shunting-yard_algorithm.

Waterfall Method

The software development cycle we will explore is called the Waterfall method. It's relatively straightforward and you can read more about it here if you are interested: https://en.wikipedia.org/wiki/Waterfall_model.

Requirements

The first task in any software project is to figure out what your program needs to do. Typically, you'd ask your customer, they'd tell you something vague, you'd ask for more clarification, and the cycle repeats. I'll be your "customer" and here's what I'd like:

The input should just be one line: a string that is a mathematical expression. There won't be any inputs that are invalid mathematical expressions.

The output should be the number that the inputted mathematical expression evaluates to.

You may ask for any clarifications you'd like, individually. Clarifications will not be made public.

Sample input:
1+2

Sample output:
3

Architecture

After you've figured out what you need to build, you should plan out how your program will fit together. Poor design is one of the biggest reasons why software projects fail (in addition to vague or unstable requirements, bad estimates, unpleasant team members,…).

I would suggest making appropriate abstractions with classes and functions to make it easier to write your program. Draw things out on the whiteboard too! What things have the potential to change? Make sure you abstract out the "volatile" things so that changes are easy to make.

Make sure you spend enough time on this step. It can really save you time later in the development cycle. While you might get away with no architecture in ENGR 151, you certainly won't get away easily in EECS 281 and upper level CS technical electives.

Construction (Coding)

Should be self-explanatory (make sure you utilize Github as your source control).

Unit Testing

One important aspect of software engineering is unit testing. How do you know your program works? You can wait until you've written all your code and test it, but that's like looking for a spider in a really messy dorm room; besides, do you even know if there is a spider?

Instead, you want to divide your code into individual modules and test them independently. Modules, which might be an individual function or an individual class, perform one specific task (such as adding two numbers together, or parsing an integer from a string).

System Testing

After you've made unit tests, you can test your entire calculator system as a whole. This is when you'd give it all kinds of interesting mathematical expressions and see if the output makes sense. What are some good things to check?

Release

If you've finished system testing, you can let me know and I'll run some test cases on it. On some projects, you only get paid when the customer is satisfied. Also, if you slacked off in any of the previous steps, you will pay dearly in this step.