First things I did:

* write a value class
* write a node class
* write a stack and a queue class
* download the scanner module

Version 1: read an expression of numbers and operators, each number and operator stored in a value object

Version 2: stored each of the value objects into a queue (my node class has a next pointer and an item pointer; the item pointer points to the value object) -- afterwards, I walked through the queue, printing what I found there

Version 3: took the queue of value objects and printed out the postfix expression using an algorithm for converting from infix to postfix I found online

Version 4: instead of printing, I stored the objects to be printed back in a queue and then printed the queue out (yielding the postfix expression again

Version 5: took the postfix queue and processed the postfix expression, yielding a single result

TODO: add the ability to deal with multiple expressions, each expression ending with a semicolon

TODO: add the storage and updating of variables

TODO: add the -d switch

Version 6: added declarations of variables -- when reading in the initial expression, I set a flag if I see the variable "var" and then throw away the "var". When I read in a variable, I check the flag. If the flag is set, I add the variable to my binary search tree with a null value. In any case, I clear the flag. Without the "var", the declaration looks like a normal expression, so I handle the rest of the declaration like it was a normal expression, which then updates the variable in the tree to the initializer value.
Version 7: added handling of the semicolon -- the program can read in one expression, need to add a loop to read and process expressions next

Subject: Re: The evolution of my solution
Posted by lusth on Wed, 24 Aug 2016 14:14:06 GMT

Version 8: added the expression loop, also added defining, finding, and updating variables. I think all that's left is the -d option, which will be trivial since I place the postfix expression on a queue that I send to the postfix expression evaluator. I just need to print out that queue instead of sending it to the evaluator.

Then, comes some heavy testing.

Subject: Re: The evolution of my solution
Posted by sestephens on Mon, 05 Sep 2016 20:44:16 GMT

Are we to read all the input at once into a large queue and then process it logically later, dealing with semicolons on the fly; or is each expression to be read into its own queue at the start, delimited by semicolons, and then processed individually later?

Subject: Re: The evolution of my solution
Posted by lusth on Mon, 05 Sep 2016 21:18:47 GMT

Your choice. I processed each semicolon terminated expression before reading the next one.