Cpsc 3740 Project

Lukas Grasse

001172543

April 06, 2017

For this project, I have implemented a calculator in Haskell. The calculator accepts statements in Reverse Polish Notation form, and prints the result of evaluating the statement.

Calculator Design

The calculator accepts statements in the form of strings that are in Reverse Polish Notation.

To evaluate a string, simply type: calculator “statement”

for example:

Main> calculator "1 2 / 4 +"

will return 4.5

The calculator assumes operands are doubles. I have designed the calculator to be robust in its error handling; if the calculator encounters something it cannot parse as an operator or operand, it will return ‘Nothing’. It is able to do this using a haskell feature known as monads. In Haskell, a monad is similar to the concept of a class from the OOP paradigm, in that it encapsulates certain functionality and provides a simple interface to use the monad. In the case of my calculator, I used a monad called Maybe, that defines an element that could have a value, or could be null. Since all my calculator code handles a Maybe of type double, the calculator will simply print Nothing, instead of throwing an error if it encounters malformed input.

The calculator supports the following operators:

Operator
+
-
*

/
^
cos
sin
tan
acos
asin
atan
log
log10
log2
sqrt
cbrt
abs

Test Cases Used

I tested the calculator with lots of different test cases; both valid inputs and invalid inputs. I tested all the operators in the table above, and expressions from learnyouahaskell.com such as:

*Main> calculator "2.7 ln"

0.9932517730102834

*Main> calculator "10 2 ^"

100.0

*Main> calculator "43.2425 0.5 ^"

6.575902979819578

*Main> calculator "1 2 * 4 +"

6.0

*Main> calculator "1 2 * 4 + 5 *"

30.0

*Main> calculator "1 2 * 4"

Nothing

*Main> calculator "1 8 wharglbllargh"

Nothing

The Maybe monad is awesome for error handling, since all invalid input can be handled in one line of the readMaybe, and a couple lines in the printFunction. One shortcoming of my calculator design, that I should fix in a future version, is the handling of non-integer cube roots. For example, the cube root of 16777216:

*Main> calculator "16777216 cbrt"

255.99999999999991

Where "16777216 cbrt" should actually evaluate to 256.

Experience of Haskell

I chose to implement this project using Haskell for various reasons. The first is that the functional programming paradigm is on the rise, and I felt it would be beneficial to gain more exposure to it. It is possible to program functionally in Javascript, and recent frameworks, such as React by Facebook have started adopting this mindset. This is why I was excited to start learning Haskell.

After seeing the basics, I was surprised at how small implementations of certain programs were. The first wow moment I experienced was seeing how small the implementation of quicksort was. This calculator also had a simple implementation, since haskell is well suited to parsing RPN statements recursively. Using monads was also interesting, and unlike anything I had really done before in programming.

In the end, the thing I like about haskell is that it forces me to break down a problem into small components. It also forces me to think more about the edge cases of problems,

whereas in imperative languages it is easy to just start programming and not stop to think about what is being implemented. Even if I don't use haskell for a large project in the future, I hope to apply the principles behind it when programming in Javascript and other languages.

References

https://en.wikipedia.org/wiki/Monad_(functional_programming)

http://learnyouahaskell.com/functionally-solving-problems#reverse-polish-notation-calculator

http://learnyouahaskell.com/a-fistful-of-monads#getting-our-feet-wet-with-maybe