# Haskell Calculator Project Paper

Main gets the input from the user, and unless the input is q, it will call the function rpn on the input and after print the result. Import Control.Monad allows us to use the command unless.

```
import Control.Monad

main = do
    putStr "> "
    input <- getLine
    unless(input=="q") $ do
        let n = rpn input
        print n
        main
```

rpn is a function that will convert the string we pass in, into a double. If given no string it will output the number 0, but if it is given the string it will return the head of the entire rest of the function. The foldl applies the function rpn' to the string it's given and words separate the numbers and operators in the string as long as there is a space in between. Once the foldl function is don't, and because it is an rpn calculator there should only be one number left on the stack and thus the string will equal the head of that stack which is the answer.

```
rpn :: String -> Double
rpn [] = 0
rpn x = head $ foldl rpn' [] $ words x
```

rpn' takes in a double and converts it to a string then back to a double for the result of the operation that is applied to each set of numbers. The four basic functions, add, subtract, multiply, and divide are in this calculator, as well as power, natural log, factorial, and the three trigonometric functions sin, cos, and tan. Sqrt takes the square root of the number before it. The second logarithm is the regular log where you get to choose the base of the log.

```
rpn' :: [Double] -> String -> [Double]
```

```
rpn' (x:y:ys) "*" = (x*y):ys
rpn' (x:y:ys) "-" = (x-y):ys
rpn' (x:y:ys) "+" = (x+y):ys
rpn' (x:y:ys) "/" = (y/x):ys
rpn' (x:y:ys) "^" = (y**x):ys
rpn' (x:xs) "ln" = log x:xs
rpn' (x:xs) "!" = fac x:xs
rpn' (x:xs) "cos" = cos x:xs
rpn' (x:xs) "sin" = sin x:xs
rpn' (x:xs) "tan" = tan x:xs
rpn' (x:xs) "log" = log x:xs
rpn' (x:xs) "sqrt" = sqrt x:xs
rpn' ys n = (read n):ys
```

Factorial takes a double, if the input is 0 then it will return 1, otherwise, it does a recursive function to calculate the factorial.

```
fac :: Double -> Double
fac 0 = 1
fac n = n * fac (n-1)
```

Test Cases used:

```
5 4 /  = 1.25
2 3 + 3 * 5 ln * = 24.14..
123 cos = -0.08..
55 45 * 35 + sqrt = 50.099..
5 4 ^ = 625
3 4 + ! = 5040
22 23 + ln = 3.8..
4 5 - = 1
12 12 * log = 4.9698..
```

Our experiences with Haskell:
Haskell was hard to get a grasp of using all the functions in tandem with one another,

mainly due to us having knowledge of c++ which is object-oriented, whereas Haskell is a functional programming language. Haskell seems like an easier language to get into, with prior knowledge of some functional programming. It was just our previous programming knowledge that gave us troubles. In Haskell, it is easier to apply functions to lists and the code needed for functions is smaller than some c++ programs.

By. Wesley Waldern and Riley Weasel Fat