Functional Programming

Will Johansson and Joseph Pecoraro

Literate Haskell - Testing

- “>” Indicates Code, “@” Indicates a Test
- Expected to Evaluate to True, will print PASS or FAIL

```
> module Tester where
>   tst True  = "PASS"
>   tst False = "FAIL"

@ [] == []
@ tst True  == "PASS"
@ tst False == "FAIL"
```

Subset

- Subset and Select are “opposites” of each other.
- Will defined sub* it terms of select:

```
> sub :: Int -> [a] -> [[a]]
> sub n xs = select (length xs - n) xs

@ sub1 [1,2,3,4] =~= [[1,2,3],[1,2,4],[1,3,4],[2,3,4]]
@ sub2 [1,2,3,4] =~= [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
@ sub3 [1,2,3,4] =~= [[1],[2],[3],[4]]
@ sub1 "PROGRAM" =~= ["PROGRA","PROGRM","PROGAM","PRORAM",
                      "PRGRAM","POGRAM","ROGRAM"]
```

Select

- Classic Combination: C(n,r) = C(n-1,r-1) + C(n-1,r)

```
> select :: Int -> [a] -> [[a]]
> select 0 _ = [[]]
> select _ [] = []
> select n (x:xs) = map (x:) (select (n-1) xs) ++ select n xs

@ select 1 [1,2,3]   =~= [[1],[2],[3]]
@ select 2 [1,2,3]   =~= [[1,2],[1,3],[2,3]]
@ select 4 [1,2,3]   =~= []
@ select 0 [1,2,3,4] =~= [[]]
@ select 4 [1,2,3,4] =~= [[1,2,3,4]]
```

Matrix Format

- Matrix is a List of Lists. Each inner list is a Row:
  - [[row], [row], [row]]
- The Columns are in the Rows:
  - [[col1, col2], [col1, col2], [col1, col2]]

Matrix Helpers

```
> split :: (a -> a -> b) -> [a] -> b
> split f [x,y] = f x y

> row :: [a] -> Int -> a
> row = (!!)

> col :: [[a]] -> Int -> [a]
> col matrix i = [ row !! i | row <- matrix ]

@ split (-) [2,1] == 1
@ row [[1,2],[3,4]] 0 == [1,2]
@ row [[1,2],[3,4]] 1 == [3,4]
@ col [[1,2],[3,4]] 0 == [1,3]
@ col [[1,2],[3,4]] 1 == [2,4]
```

Matrix Addition

```
> -- Recursive
> add (x:xs) (y:ys) = zipWith (+) x y : add xs ys

> -- Pipeline
> add' left right = zipWith (zipWith (+)) left right

@ add [[1,2],[3,4]] [[5,6],[7,8]] == [[6,8],[10,12]]
```

Inner and Friends

```
> pair :: a -> a -> [a]
> pair a b = [a,b]

> inner = zipWith
> mplus = zipWith (+)

@ pair 1 2 == [1,2]
@ pair [1] [2] == [[1],[2]]
@ inner pair [1,2,3] [4,5,6] == [[1,4],[2,5],[3,6]]
@ inner (+) [1,2,3] [4,5,6] == [5,7,9]
@ inner mplus [[1,2],[3,4]] [[5,6],[7,8]]
                            == [[6,8],[10,12]]
```

Transpose

- Swap Rows and Columns. Order of the results matter:

```
> transpose :: [[a]] -> [[a]]
> transpose [] = []
> transpose m = [col m i | i <- [0.. numColumns-1]]
>     where numColumns = length (head m)

@ transpose [[1,2,3,4]] == [[1],[2],[3],[4]]
@ transpose [[1,2,3],[4,5,6]] == [[1,4],[2,5],[3,6]]
@ transpose [[1,4],[2,5],[3,6]] == [[1,2,3],[4,5,6]]
```

Cross Product

- A Function and two lists. Applying the function on pairwise elements between the two lists. Order matters.
- Remember, in Matrices, the “elements” are “rows”.

```
> cross :: (a -> b -> c) -> [a] -> [b] -> [[c]]
> cross f m1 m2  = [ [f x y | y <- m2] | x <- m1 ]

@ cross pair [1,2,3] [4,5,6] ==
  [[[1,4],[1,5],[1,6]],[[2,4],[2,5],[2,6]],[[3,4],[3,5],[3,6]]]
@ cross pair [[1,2],[3,4]] [[5,6],[7,8]] ==
  [[[[1,2],[5,6]],[[1,2],[7,8]]],[[[3,4],[5,6]],[[3,4],[7,8]]]]
```

Matrix Multiplication

- Matrix 1's rows are cross with Matrix 2's columns
- Multiply element by element and sum the products.

```
> mul :: (Num a) => [[a]] -> [[a]] -> [[a]]
> mul m1 m2 = [ map sum $ map mulLists x |
>               x <- cross pair m1 (transpose m2) ]
>     where mulLists list = split (inner (*)) list

@ mul [[1,2],[3,4]] [[5,6],[7,8]] == [[19,22],[43,50]]
@ mul [[1,2,3]] [[4],[5],[6]] == [[32]]
@ mul [[4],[5],[6]] [[1,2,3]] ==
      [[4,8,12],[5,10,15],[6,12,18]]
```

Matrix Multiplication

- Matrix 1's rows are cross with Matrix 2's columns
- Multiply element by element and sum the products.

```
> mul :: (Num a) => [[a]] -> [[a]] -> [[a]]
> mul m1 m2 = [ map (sum . mulLists) x |
>               x <- cross pair m1 (transpose m2) ]
>     where mulLists list = split (inner (*)) list

@ mul [[1,2],[3,4]] [[5,6],[7,8]] == [[19,22],[43,50]]
@ mul [[1,2,3]] [[4],[5],[6]] == [[32]]
@ mul [[4],[5],[6]] [[1,2,3]] ==
      [[4,8,12],[5,10,15],[6,12,18]]
```

Matrix Multiplication

- Matrix 1's rows are cross with Matrix 2's columns
- Multiply element by element and sum the products.

```
> mul :: (Num a) => [[a]] -> [[a]] -> [[a]]
> mul m1 m2 = map (map (sum . mulLists))
>                 $ cross pair m1 (transpose m2)
>     where mulLists list = split (inner (*)) list

@ mul [[1,2],[3,4]] [[5,6],[7,8]] == [[19,22],[43,50]]
@ mul [[1,2,3]] [[4],[5],[6]] == [[32]]
@ mul [[4],[5],[6]] [[1,2,3]] ==
      [[4,8,12],[5,10,15],[6,12,18]]
```

Information

- These Slides are Available at: http://www.cs.rit.edu/~jjp1820/fp/Presentation.pdf
- Will's Reading List: http://www.cs.rit.edu/~jjp1820/fp/
- Joe's Reading List: http://conquerant.org/Haskell.aspx