COMP105 Lecture 17

Scan

Recap: `foldr`

Folds turn a list into a **single value**

```
ghci>  foldr (+) 0 [1,2,3,4]
10
```

```
ghci>  foldr (*) 1 [1,2,3,4]
24
```

```
ghci> foldr (++) [] ["one", "two", "three"]
"onetwothree"
```

Recap: `foldr`

`foldr` has an **accumulator** that is modified as the list is processed

```
ghci> foldr1 (\ x acc -> x ++ " " ++ acc)
                         ["one", "two", "three"]
"one two three"
```

```
acc = "three"
acc = "two" ++ " " ++ "three"
acc = "one" ++ " " ++ "two three"
```

scan

Scan is like fold, but it outputs the accumulator at each step

```
ghci> scanr (+) 0 [1,2,3,4]
[10,9,7,4,0]
```

```
ghci> scanr (*) 1 [1,2,3,4]
[24,24,12,4,1]
```

```
ghci> scanr1 (\ x acc -> x ++ " " ++ acc)
                                ["one", "two", "three"]
["one two three","two three","three"]
```

scanr implementation

```haskell
scanr' :: (a -> b -> b) -> b -> [a] -> [b]

scanr' _ init [] = [init]
scanr' f init (x:xs) =
    let
        recursed = scanr' f init xs
        new = f x (head recursed)
    in
        new : recursed
```

scan variants

There are also **left to right** versions of `scan`

```
ghci> scanl (+) 0 [1..10]
[0,1,3,6,10,15,21,28,36,45,55]
```

```
ghci> scanr (+) 0 [1..10]
[55,54,52,49,45,40,34,27,19,10,0]
```

```
ghci> :t scanl
scanl :: (b -> a -> b) -> b -> [a] -> [b]
```

Fibonacci the higher order way

```
fib_pairs n = scanl (\ (a, b) _ -> (b, a + b))
                                   (0, 1) [1..n]
```

```
ghci> fib_pairs 7
[(0,1),(1,1),(1,2),(2,3),(3,5),(5,8),(8,13),(13,21)]
```

```
fib_to_n n = map fst (fib_pairs n)
```

```
ghci> fib_to_n 7
[0,1,1,2,3,5,8,13]
```

Exercises

1. Use scan to write a function `prefixMaximum` that takes a list of numbers and returns a list of the same length, where element i is the largest number in the first i elements of the input list. So
   `prefixMaximum [1,2,3,2,4] = [1,2,3,3,4]`

2. Use scan to write a function `powersOfTwo` that takes a number `n` and outputs a list containing 2^0, 2^1, ..., 2^n