Map and Foldr

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Objectives

- Define *higher order function* and give some examples.
- Define the foldr and map functions.
- Use foldr and map to implement recursion patters we saw earlier.
- Understand the lambda form and how to use eta-expansion.

Mapping functions the hard way

What do the following definitions have in common?

Example 1

```
1 incL [] = []
2 incL (x:xs) = x+1 : incL xs
```

Example 2

```
1 doubleL [] = []
2 doubleL (x:xs) = x*2 : doubleL xs
```


Mattox's Law of Computing

The computer exists to work for us; not us for the computer. If you are doing something repetitive for the computer, you are doing something wrong.
Stop what you're doing and find out how to do it right.

Mapping functions the easy way

Map Definition

$$map\, f\, [x_0, x_1, \ldots, x_n] = [f\, x_0, f\, x_1, \ldots, f\, x_n]$$

```
1 map :: (a->b) -> [a] -> [b]
2 map f [] = []
3 map f (x:xs) = f x : map f xs
4
5 incL = map inc
6
7 doubleL = map double
```

- `inc` and `double` have been transformed into recursive functions.
- I dare you to try this in Java.

Folding functions

What do the following definitions have in common?

Example 1

```
1 sumL [] = 0
2 sumL (x:xs) = x + sumL xs
```

Example 2

```
1 prodL [] = 1
2 prodL (x:xs) = x * prodL xs
```


foldr

Fold Right Definition

$$foldr\, f\, z\, [x_0, x_1, \ldots, x_n] \;=\; f\, x_0\, (f\, x_1\; \cdots\; (f\, x_n\; z) \cdots)$$

- To use `foldr`, we specify the function *and* the base case.

```
1 foldr :: (a -> b -> b) -> b -> [a] -> [b]
2 foldr f z [] = z
3 foldr f z (x:xs) = f x (foldr f z xs)
4
5 sumlist = foldr (+) 0
6 prodlist = foldr (*) 1
```


Encoding Recursion using `fold`

Recursive Style

```
1 flatten [] = []
2 flatten (x:xs) = x ++ flatten xs
```

- Notice the pattern between the recursive version and the higher order function version.
- **Note well**: the second parameter of the function argument to `foldr` represents the *result* of the recursive call.

Higher Order Style

```
1 flatten = foldr (++) []
```

Something to think about...

- You can write `map` using `fold`. Try it!
- Note, the reverse direction will not work. Why not?

Other Functions

Other Examples

- foldl
- zipWith
- exists
- filter
- fix

- There are many many other HOFs defined.
- Learn them; they will help you.