HASKELL STYLE

Curt Clifton
Rose-Hulman Institute of Technology

SVN Update *HaskellInClass* folder, open *basics.hs* and *style.hs*

THE GOLDEN RULE OF HASKELL INDENTATION

Code which is part of some expression should be indented farther in than the line containing the beginning of that expression

Live it.
Learn it.
Pass it on.

BINARY TREE DATATYPE

- What do we need to store for a binary tree node?
- Nodes contain other nodes
- Need *recursive data type*:
    - *data BinaryTree a =*
      ExtNode
      | IntNode a (BinaryTree a) (BinaryTree a)
      deriving Show

RECORDS

- Define custom data types with named "fields"
- Automatically create accessor functions

```
type CustomerID = Int
type Address = [String]

data Customer = Customer {
        customerId      :: CustomerID,
        customerName    :: String,
        customerAddress :: Address
    } deriving (Show)
```

CONTROL FLOW EXPRESSIONS

Matches a pair and binds x and y

- *case* expression:
  ```
  eo xs =
      case xs of
          [] -> []
          (x:[]) -> [x]
          (x':_:xs') -> x' : eo xs'
  ```
- Cases must have **same type**
- Uses pattern matching

- *guards*:
  ```
  data Pair a b = Pair a b

  twins::Eq a => Pair a a -> String
  twins (Pair x y)
              | x == y  = “yep”
              | otherwise   = “nope”
  ```
- Similar to cases, but use *Bool* values to select

Q2,3

BINDING EXPRESSIONS

- *let* expressions define local names for values
  - Not "variables"
  - Can't mutate them
- *where* expressions give supporting definitions at the end of a function

Example

```
fib n = fst (ffp n)
  where
    ffp 0 = (0, 0)
    ffp 1 = (1, 0)
    ffp n =
      let (nm1, nm2) = ffp (n-1)
      in (nm1 + nm2, nm1)
```

INFIX OPERATORS

- Surrounding binary function names with `backticks` lets us use them as infix operators:

```
> 4 `div` 2
> "foo" `isPrefixOf` "foolish"
```

- Surround infix operators with parentheses lets us treat them like function names:

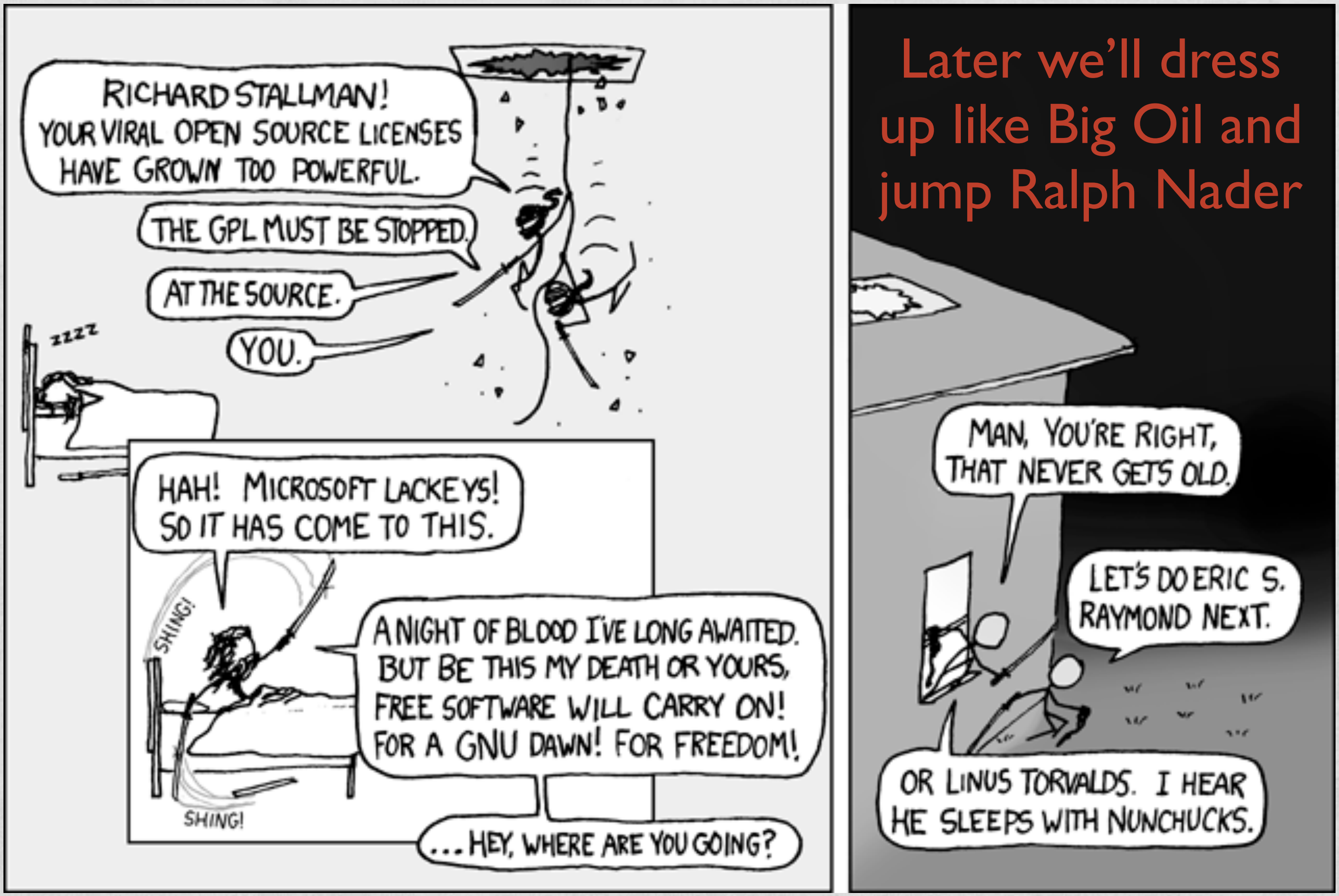
```
(<-*) :: BinaryTree a -> BinaryTree a -> BinaryTree a
_ <-* ExtNode = error "Can't add a left child to ext. node"
t <-* (IntNode x _ right) = IntNode x t right
```

LOADS OF LIST FUNCTIONS

length	(++)	take	elem
null	concat	drop	notElem
head	reverse	splitAt	filter
tail	and	takeWhile	isPrefixOf
last	or	dropWhile	isInfixOf
init	all	span	isSuffixOf
lines/unlines	any	break	zip

See *http://www.haskell.org/ghc/docs/latest/html/libraries/*

OPEN SOURCE



Later we'll dress up like Big Oil and jump Ralph Nader

EXAMPLE: ADLER-32

- Concatenates two 16-bit checksums
  - First is the sum of all the input bytes, plus 1
  - Second is the running total of the intermediate values of the first checksum
  - Both are modulo 65521

LEFT FOLD

operation

accumulator

```
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl op acc (x:xs) = foldl op (op acc x) xs
foldl _  acc _      = acc
```

list to process

Q6

ADLER-32 WITH FOLDL

```haskell
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl op acc (x:xs) = foldl op (op acc x) xs
foldl _  acc _      = acc
```

```haskell
adler32_v3 :: String -> Int
adler32_v3 xs = let (chSum1,chSum2) = foldl procByte (1,0) xs
                in (chSum2 `shiftL` 16) .|. chSum1
    where procByte (chSum1,chSum2) x =
              let chSum1' = (chSum1 + (ord x .&. 0xff))
              in (chSum1' `mod` base, (chSum1' + chSum2) `mod` base)
```

RIGHT FOLD

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr op acc (x:xs) = op x (foldr op acc xs)
foldr _  acc []     = acc
```

Consider: *foldr (+) 0 [1..3]*

Input: *1 : (2 : (3 : []))*
Result: *1 + (2 + (3 + 0))*

THE POWER OF FOLDR

```
-- filter using foldr
myFilter :: (c -> Bool) -> [c] -> [c]
myFilter pred xs = foldr op [] xs
    where op x acc | pred x      = x : acc
                   | otherwise = acc

-- map using foldr
myMap :: (c -> d) -> [c] -> [d]
myMap f xs = foldr op [] xs
    where op x acc = (f x) : acc

-- append using foldr
append :: [c] -> [c] -> [c]
append xs ys = foldr (:) ys xs
```

Try to match types here to types in foldr's signature

SPACE LEAKS

- *foldl* generates big thunks
  - take lots of space to store and evaluate
  - can use *foldl'* for strict (non-lazy) version
- *foldr may* generate big thunks…
  - …but most applications don't if they leave right-side unchanged or ignore it