Informatics 1

Functional Programming Lecture 8

Tuesday 21 October 2008

Class Exam Review

Philip Wadler

University of Edinburgh

Part I

Class exam review

1a

```
f :: Char -> Int
f x | 'a' <= x && x <= 'z'  =  ord x - ord 'a'
    | 'A' <= x && x <= 'Z'  =  ord x - ord 'A'

test1a = f 'A' == 0 && f 'B' == 1 && f 'Z' == 25 &&
         f 'a' == 0 && f 'b' == 1 && f 'z' == 25
```

Alternative solution

```
f x | isAlpha x  =  ord (toLower x) - ord 'a'
```

Poor solution

```
f x | 'a' <= x && x <= 'z'  =  ord x - 97
    | 'A' <= x && x <= 'Z'  =  ord x - 65
```

1b

```
g :: String -> Int
g xs  =  sum [ f x | x <- xs, isAlpha x ]

test1b = g "aBc4e" == 7 && g "?!" == 0
```

Incorrect solution

```
g :: String -> Int
g xs  =  [ sum (f x) | x <- xs, isAlpha x ]
```

1c

```
h :: String -> Int
h []  =  0
h (x:xs) | isAlpha x  =  f x + h xs
         | otherwise  =  h xs

test1c = h "aBc4e" == 7 && h "?!" == 0
```

Incorrect solution

```
h :: String -> Int
h (x:xs)  =  f x + h xs
```

2a

```
c :: [Int] -> [Int] -> [Int]
c xs ys | length xs == length ys
  =  [ x-y | (x,y) <- zip xs ys ]

test2a = c [5,7,3] [1,2,4] == [4,5,-1]
```

Incorrect solution

```
c :: [Int] -> [Int] -> [Int]
c xs ys | length xs == length ys
  =  [ x-y | x <- xs, y <- ys ]
```

```
Main> zip [1,2,3] [4,5,6]
[(1,4), (2,5), (3,6)]

Main> [ (x,y) | x <- [1,2,3], y <- [4,5,6] ]
[(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)]
```

2b

```
d :: [Int] -> [Int] -> [Int]
d [] []           =  []
d (x:xs) (y:ys)   =  x-y : d xs ys

test2b = d [5,7,3] [1,2,4] == [4,5,-1]
```

Poor solution

```
d :: [Int] -> [Int] -> [Int]
d [] []  =  []
d (x:xs) (y:ys) | length xs == length ys
  =  x-y : d xs ys
```

2c

```
e :: [Int] -> [Int] -> Bool
e xs ys  =  and [ z == 0 | z <- c xs ys ]

test2c = e [3,3,3] [3,3,3] &&
         not (e [3,3,3] [3,3,2]) &&
         e [] []
```

Alternative solution

```
e xs ys  =  c xs ys == replicate 0 (length xs)
```

2c

Question 2c assumed (without saying so—a mistake!) that both lists to be compared for equality are the same length. If they may be different lengths, we need to check for this first.

```
e :: [Int] -> [Int] -> Bool
e xs ys  =  length xs == length ys &&
            and [ z == 0 | z <- c xs ys ]

test2c' = e [3,3,3] [3,3,3] &&
          not (e [3,3,3] [3,3,2]) &&
          not (e [3,3,3] [3,3])
          e [] []
```