Resit Exam Software Specification and Testing, January 11, 2016
with some answers

There are 4 theory questions (T) and 4 lab questions (L).
If you are only taking the resit for theory, you can skip the lab questions, and if you are only taking the resit for lab, you can skip the theory questions.

Question 1 (T)

A property p on a domain D is a function $D \to \{\top, \bot\}$, where $\top$ and $\bot$ are the two truth values. A property p on D is stronger than a property q on D if it holds for all $x \in D$ that $p(x)$ implies $q(x)$. A property p on D is weaker than a property q on D if q is stronger than p. (You have seen these notions during the workshop sessions.)

Which of the following pairs of properties on the integers is stronger?

1. $\lambda x \mapsto x < 3$ and $\lambda x \mapsto x \geq x$.
2. $\lambda x \mapsto x + y = 0$ and $\lambda x \mapsto x + y \geq 0$.
3. $\lambda x \mapsto x \geq 0$ and $\lambda x \mapsto x \leq 0$.
4. $\lambda x \mapsto x^2 > 0$ and $\lambda x \mapsto x \neq 0$.
5. $\lambda x \mapsto x - y > 0$ and $\lambda x \mapsto x > y$.

Question 2 (T) Give all symmetric relations on the set $\{0, 1\}$. Express these relations as sets of pairs, and draw pictures of them.

Question 3 (T) Let the relation R on the set $A = \{0, 1, 2\}$ be given by

$$R = \{(0, 1), (1, 2)\}.$$

1. Determine R^{-1}
2. Determine $R \cup R^{-1}$
3. Determine $R^* \cup (R^{-1})^*$
4. Determine $(R \cup R^{-1})^*$

Question 4 (T) Prove with induction:

$$1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1.$$

This is the end of the Theory part.

```
module ResitExamAnswers where

import Data.List
import System.Random
import Test.QuickCheck
```

Question 5 (L) Let the following function be given:

```
g :: [Int] -> (Int,Int) -> [Int]
g xs (i,j) = take (j-i) (drop i xs)
```

Thus, `g xs i j` gives the subsequence of `xs` starting with `xs!!i` and ending at `xs!!(j-1)`. Here is a check (recall that the first item of a list has index 0).

```
*ResitExam> g [-2,3,-4,11,13,-1] (0,2)
[-2,3]
*ResitExam> g [-2,3,-4,11,13,-1] (0,0)
[]
*ResitExam> g [-2,3,-44,11,13,-1] (2,5)
[-4,11,13]
```

Use this to write a function `maxval` for finding the value of a subsequence with a maximal sum.

This should give:

```
*Main> maxval [-2,3,-4,11,13,-1]
24
```

The subsequence consisting of just $[11, 13]$ has a maximal sum, and this sum is 24.

Test your solution with at least two QuickCheck properties.

Answer:

```
maxval :: [Int] -> Int
maxval xs = let
   l = length xs
 in
   maximum [ sum (g xs (i,j)) | i <- [0..l], j <- [i..l] ]
```

Some QuickCheck properties for this:

```
prop_maxval1 xs = maxval xs == maxval (reverse xs)

prop_maxval2 xs ys =
  maxval (xs++ys) >= max (maxval xs) (maxval ys)
```

Both of these tests succeed.

Question 6 (L) Continued from the previous. Write a function

`maxseq :: [Int] -> [Int]`

for finding a subsequence of a list of integers with maximal sum.

Your program should give:

```
*ResitExam>  maxseq [-2,3,-4,11,13,-1]
[11,13]
```

But, of course, a single test case is not enough. Your task is to test your function with QuickCheck.

Answer:

```
maxm :: Ord a => [(a,b)] -> b
maxm xs = let
  ys = sortBy (\ (x,y) (u,v) -> compare u x) xs
 in
  snd (head ys)

maxseq :: [Int] -> [Int]
maxseq xs = let
   l = length xs
 in
   maxm [ (sum $ g xs (i,j),g xs (i,j)) | i <- [0..l], j <- [i..l] ]
```

Again some QuickCheck properties:

```
prop_maxseq xs = maxval xs == sum (maxseq xs)

prop_maxseq2 xs =
  sum (maxseq xs) == sum (maxseq (reverse xs))
```

Question 7 (L) We have a set of cards with values (positive natural numbers). Example: suppose you have five cards, numbered $1, 2, 3, 4, 5$.

The task is to spit this set into two piles X and $\overline{X}$ (the remaining cards of the set), with the *sum* of the values of the cards in X equal to the *product* of the values of the cards in $\overline{X}$.

Example: if there are just five cards, numbered $1, 2, 3, 4, 5$, then there is only one solution, namely $[3, 5]$, for $3 + 5 = 8$ and the complement of $[3, 5]$ satisfies $1 * 2 * 4 = 0$.

Write a program for the general case, and give all solutions for the card set $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. The type of the `solutions` function should be:

```
solutions :: [Int] -> [[Int]]
```

For the example case $[1, 2, 3, 4, 5]$ we should get:

```
solutions [1,2,3,4,5] = [[1,3]]
```

for, as we have seen, for this case `[1,3]` is the only solution.

Give code for `solutions`. Next, test your code for correctness.

Answer:

```
solutions cards =
  [ xs | xs <- sublists cards,
         sum xs == product (cards \\ xs) ]

sublists :: [a] -> [[a]]
sublists [] = [[]]
sublists (x:xs) = sublists xs ++ map (x:) (sublists xs)
```

This gives the following solutions for [1..10]:

```
ResitExam> solutions [1..10]
[[4,5,6,8,9,10],[2,3,5,6,7,8,9],[1,2,3,4,5,8,9,10]]
```

That these solutions are correct is easy to test.

Here is a general QuickCheck property (we need to restrict the lengths of the input lists, for otherwise the tests take too long):

```
prop_solutions xs = let
   xs' = map abs xs
   p   = product xs'
 in
   length xs' < 15 ==> elem [p] (solutions (p:xs'))
```

This test succeeds.

Question 8 (L) Your programmer Red Curry has written the following function for generating lists of floating point numbers.

```
probs :: Int -> IO [Float]
probs 0 = return []
probs n = do
             p <- getStdRandom random
             ps <- probs (n-1)
             return (p:ps)
```

He claims that these numbers are random in the open interval (0..1). Your task is to test whether this claim is correct, by counting the numbers in the quartiles

$(0..0.25), [0.25..0.5), [0.5..0.75), [0.75..1)$

and checking whether the proportions between these are as expected.

E.g., if you generate 10000 numbers, then roughly 2500 of them should be in each quartile.

Implement this test, and report on the test results.

Answer:

Here is one way of doing it:

```
prob2quartile :: Float -> Int
prob2quartile p | p < 0.25  = 1
                | p < 0.50  = 2
                | p < 0.75  = 3
                | otherwise = 4
```

```
type Count a     = (a, Integer)

addCount :: Eq a => [Count a] -> Count a -> [Count a]
addCount []     y    = [y]
addCount (x:xs) y
    | fst x == fst y = (fst x, snd x + snd y):xs
    | otherwise      = x:addCount xs y

freqCount :: Ord a => [a] -> [Count a]
freqCount xs = sortBy (\ (x,_) (y,_) -> compare x y)
                 (foldl' addCount [] [(x, 1) | x <- xs])

countQuarts :: [Float] -> [Count Int]
countQuarts = freqCount . map prob2quartile

prsInspect = do
               xs <- probs 10000
               return (countQuarts xs)
```

Here is a session with this:

```
*ResitExamAnswers> prsInspect
[(1,2487),(2,2492),(3,2524),(4,2497)]
*ResitExamAnswers> prsInspect
[(1,2394),(2,2505),(3,2583),(4,2518)]
*ResitExamAnswers> prsInspect
[(1,2464),(2,2527),(3,2502),(4,2507)]
*ResitExamAnswers> prsInspect
[(1,2469),(2,2480),(3,2510),(4,2541)]
```

These results show that the claim of Red Curry was correct.