FUNCTIONAL PROGRAMMING, 2017-2018 SPRING, FINAL EXAM

105 minutes May 31, 2018

Id	Full Name	Signature

Q 1	Q 2	Q 3	Q 4	Total
/ 25	/ 25	/ 20	/ 30	/ 100

No questions are allowed. Answer the questions to the best of your understanding. If you need to make extra assumptions, state them clearly. Make sure that all your answers are sufficiently (and mathematically) explained.

1. Consider the functions given below.

```
foo []     = return []
foo (x:xs) = do v <- x
                vs <- foo xs
                return (v : vs)
```

```
getInt = do line <- getLine
            return (read line :: Int)
```

(a) What is the most general type for the function `foo`? Explain.

(b) What would be the behavior of the function call `foo [getInt, getInt, getInt]`?

(c) Write a function that will make the function call in (b) and print the result.

2. Consider the code given below for calculating the roots of a second degree polynomial.

```
data VariedResult a = None | Single a | Multiple [a]
                      deriving Show

roots :: (Float, Float, Float) -> VariedResult Float
roots (a, b, c)
  | disc <  0 = None
  | disc == 0 = Single ((-b) / (2 * a))
  | otherwise = Multiple [x1, x2]
  where
    disc = b * b - 4 * a * c
    discr = sqrt disc
    x1 = (-b + discr) / (2 * a)
    x2 = (-b - discr) / (2 * a)
```

(a) What are the results of the following expressions? Briefly explain.

i. `roots (1, -4, 4)`

ii. `roots (1, 1, -6)`

iii. `roots (1, 1, 1)`

(b) Make the `VariedResult` type a `Functor` instance so that the expressions below will produce the signs of the roots. The `signum` function returns the sign of a real number (1.0 for positive, 0 for zero, -1.0 for negative). Explain.

```
signum <$> roots (1,  1,  1)   -- expected result:  None
signum <$> roots (1, -4,  4)   -- expected result:  Single 1.0
signum <$> roots (1,  1, -6)   -- expected result:  Multiple [1.0, -1.0]
```

Id	Full Name	Signature

3. The `Prelude.sqrt` function returns the `NaN` value ("not a number") for negative inputs. In question (2), assume that the `sqrt` function is defined as follows:

```
sqrt :: Float -> Float
sqrt x
  | x < 0     = error "negative input"
  | otherwise = Prelude.sqrt x
```

Explain how the code in question (2) would behave under this definition and why. Compare this behavior to a similarly structured code in Python.

4. In the questions below, check only one option and for the other options VERY BRIEFLY EXPLAIN why it is false.

(a) Choose the equivalent of the following list comprehension `[f x | x <- xs, g x]`:.

`[ ] map g (map f xs)`

`[ ] filter g (map f xs)`

`[ ] map f (filter g xs)`

`[ ] map f (takeWhile g xs)`

(b) Choose the option that implements the Prelude function `map`:

```
[ ] map f = foldr (\x xs -> xs ++ [f x]) []
```

```
[ ] map f = foldr (\x xs -> f x ++ xs) []
```

```
[ ] map f = foldl (\xs x -> f x : xs) []
```

```
[ ] map f = foldl (\xs x -> xs ++ [f x]) []
```

(c) Choose the option that implements the Prelude function `filter`:

```
[ ] filter p = foldl (\xs x -> if p x then x : xs else xs) []
```

```
[ ] filter p = foldr (\x xs -> if p x then x : xs else xs) []
```

```
[ ] filter p = foldr (\x xs -> if p x then xs ++ [x] else xs) []
```

```
[ ] filter p = foldl (\x xs -> if p x then xs ++ [x] else xs) []
```