Sample midterm solutions

29 October 2018

You have 80 minutes to complete these questions. Answer on the paper given. If you need additional paper let me know, but you must turn it in also. You may not use a computer or calculator. All notes and electronic devices must be put away.

```
thwok 'a' = 'y'
thwok 'n' = 'z'
thwok 'm' = 'g'
thwok  _  = 's'
```

1. The preceding function definition, `thwok`, uses pattern matching on characters. What is the result of each of these expressions?

   a. `thwok 'm'` ↪ `'g'`

   b. `thwok 'y'` ↪ `'s'`

   c. `map thwok "panama"` ↪ `"syzygy"`

2. Write down a type signature for `thwok`.

   ```
   thwok :: Char -> Char
   ```

```
flim x
  | even x = 2*x + 1
  | x > 10 = 3*x - 2
  | otherwise = x + 1
```

3. The preceding function, `flim`, uses Boolean guards to distinguish three cases. What is the result of each of these expressions?

   a. `flim 2` ↪ `5`

   b. `flim 11` ↪ `31`

   c. `flim 3` ↪ `4`

   d. `map flim [8..12]` ↪ `[17,10,21,31,25]`

```haskell
square x = x*x
eek y 0 = 1
eek y z
  | even z = eek (square y) (z `div` 2)
  | otherwise = y * eek y (z - 1)
```

4. Use the preceding functions, `square` and `eek`, to derive the result of the following expression. `square` does exactly what it says. `eek` is recursive and uses both pattern-matching (on zero) and guards.

   `eek 3 5` ↪

   ```
   eek 3 5 ↪
   3 * eek 3 4 ↪
   3 * eek 9 2 ↪
   3 * eek 81 1 ↪
   3 * 81 * eek 81 0 ↪
   3 * 81 * 1 ↪
   243
   ```

```haskell
grup :: [Integer] -> [Integer]
grup = filter (< 20)

bink :: [Integer] -> Integer
bink = sum . grup . take 5

bonk :: [Integer] -> Integer
bonk = sum . take 5 . grup
```

5. The preceding functions are defined using partial application and function composition. What is the result of each of these expressions?

   a. `bink [80,13,92,19,15,-7]` ↪ `47`
   b. `bonk [80,13,92,19,15,-7]` ↪ `40`
   c. `bink []` ↪ `0`
   d. `bonk [18..25]` ↪ `37`

---

6. Recall that a type is a `Functor` if it has a function `fmap` that can apply a function to its element type(s). For lists, `fmap` is the same as `map`. But `fmap` also works on `Maybe` types, the `Right` side of an `Either` type, and the *second* element in a pair. That is, we have all these instances:

```
fmap :: (a -> b) -> [a] -> [b]
fmap :: (a -> b) -> Maybe a -> Maybe b
fmap :: (a -> b) -> Either c a -> Either c b
fmap :: (a -> b) -> (c, a) -> (c, b)
```

   What is the result of each of these expressions?

   a. `fmap (*2) (Just 5)` ↪ `Just 10`
   b. `fmap (*2) (Left 5)` ↪ `Left 5`
   c. `fmap (*2) (Right 5)` ↪ `Right 10`
   d. `fmap (*2) (5,6)` ↪ `(5,12)`
   e. `fmap (+1) $ fmap (*2) [1..4]` ↪ `[3,5,7,9]`

```
pelt :: [Integer] -> [Integer]
pelt [] = [0]
pelt (h:t) = h : h : pelt t
```

7. The preceding function, `pelt`, is recursive and uses pattern-matching on a list argument. What is the result of each of these expressions?

   a. `pelt []` ↪ `[0]`
   b. `pelt [6]` ↪ `[6,6,0]`
   c. `pelt [7,2]` ↪ `[7,7,2,2,0]`

```
korn :: String -> String
korn = zipWith max "jjjjj"
```

8. The preceding function, `korn`, is defined as a partial application using `zipWith` and `max`. Recall that `max` returns the greater of its two arguments:

- `max 3 5` ↪ `5`
- `max 'a' 'b'` ↪ `'b'`
- `max 'z' 'k'` ↪ `'z'`

What is the result of each of these expressions?

   a. `korn "hello"` ↪ `"jjllo"`
   b. `korn "quiz"` ↪ `"qujz"`
   c. `korn "haskell"` ↪ `"jjskj"`

```
main = putStrLn "All done!"
```

Extra questions

The above indicates the approximate length of the real exam, but here are some additional practice questions.

```
quan :: [a] -> [a]
quan [] = []
quan (h:t) = quan t ++ [h]
```

9. Recall that `(++)` is the list concatenation operator.

- `"test" ++ "two"` ↪ `"testtwo"`
- `[1..5] ++ [8..10]` ↪ `[1,2,3,4,5,8,9,10]`

What is the result of each of these expressions?

   a. `quan "hello"` ↪ `"olleh"`
   b. `quan [2..5]` ↪ `[5,4,3,2]`

```haskell
twee :: [Integer] -> [Integer]
twee = map (+3)

florm :: [Integer] -> [Integer]
florm = filter even

pink :: [Integer] -> Integer
pink = sum . florm . twee

ponk :: [Integer] -> Integer
ponk = sum . twee . florm

punk :: [Integer] -> Integer
punk = sum . twee . florm . twee
```

10. The preceding functions are defined using partial application and function composition. What is the result of each of these expressions?

    a. `pink [4..7]` ↪ 18

    b. `ponk [4..7]` ↪ 16

    c. `punk [4..7]` ↪ 24