Midterm Test

Multiple Choice Section (36 points, 4 points per question)

Question 1

Consider the sets $A = \{1, 3, 5\}$ and $B = \{2, 5, 6, 8\}$. How many elements are in the **sum** of these two sets $A + B$?

- ○ 7
- ○ 6
- ○ 12
- ○ 4

Question 2

Select the **false** statement from below.

- ○ Haskell describes computations as mathematical functions that transform data.
- ○ Testing can be used to show that a complex program is free of errors.
- ○ The library of built-in functions that Haskell modules automatically import is called the `Prelude`
- ○ Higher level programming languages are easier to program in, at the cost of some direct control over hardware.

Question 3

Consider a function to take the average of two floating-point numbers. What would be an appropriate type signature for such a function?

-
  ```
  average :: (Int -> Int) -> Int
  ```

-
  ```
  average :: Int -> (Int -> Int)
  ```

-
  ```
  average :: (Double -> Double) -> Double
  ```

-
  ```
  average :: Double -> (Double -> Double)
  ```

Question 4

Here is a definition of a function to convert temperature measurements from Fahrenheit to Celsius scales:

```
fahrenheitToCelsius :: Int -> Int
fahrenheitToCelsius n = (n - 32) / 1.8
```

Identify an issue with this function definition.

- The type `Int` is bounded, which is not appropriate for temperature values, as overflow may happen.
- The type `Int` doesn't support fractional division `(/)`, so a type like `Double -> Double` should be used.
- The type `Int` doesn't include the unit of measure (C or F), so a `String` type should be used instead.
- There are no issues with the above function

Question 5

What is the type of the Haskell expression `"x":[]`?

- ○ `String`

- ○ `[Char]`

- ○ `[String]`

- ○ `Char`

Question 6

A programmer is debugging their program and discovers that an `Int` value that they had assumed to be a large positive number was in fact a number like `-9223372036854775808`. What is most likely to have happened?

- ○ The computer ran out of memory and therefore returned an unrelated number.
- ○ The programmer has tried to divide an `Int` by a `Double`, resulting in a type error.
- ○ The number has exceeded `maxBound :: Int` and therefore *overflowed*.
- ○ A parse error in the Haskell code led to the wrong number being read by the compiler.

Question 7

Consider the following type definitions.

```haskell
type B = Bool
data X = A X
       | B B
       | C Y
```

Which of the following names stand for *types* in the above definitions? Check all that apply.

- [ ] `A`
- [ ] `B`
- [ ] `C`
- [ ] `X`
- [ ] `Y`

Question 8

Here is a function to produce a list of even numbers from a given starting point to a given ending point:

```
evenFromTo :: Int -> Int -> [Int]
evenFromTo n m
  | n > m = []
  | otherwise = n : evenFromTo (n+2) m
```

What is wrong with this function?

- ○ It uses the `Int` type which may lead to overflow errors.
- ○ It doesn't handle the case where `m > n` and may therefore loop forever.
- ○ It doesn't produce a list of even numbers when the initial `n` is odd
- ○ There are no problems with this function.

Question 9

Suppose there exists a function `mix`, that, given two lists, produces a list containing items from both lists in alternating order. For example:

```
>>> mix [1,2,3] [10,20,30]
[1,10,2,20,3,30]

>>> mix "ABC" "abcdef"
"AaBbCcdef"

>>> mix [True, False] []
[True,False]
```

What is the most general type for this function?

- ○ `[a] -> [b] -> [a]`
- ○ `[a] -> [b] -> [(a,b)]`
- ○ `[a] -> [a] -> [a]`
- ○ `[Int] -> [Int] -> [Int]`

Coding Question 1: Area of a Circle (10 points)

The area of a circle of diameter d is given by the formula:

$$A = \pi \times \left(\frac{d}{2}\right)^2$$

.

Write a function `circleArea :: Double -> Double` which computes this area, given a diameter length `d :: Double`. The constant π can be used in Haskell by typing `pi`.

You may assume the input is not negative. You will not be tested on any negative input.

Coding Question 2: Dragon Years (12 points)

In the Chinese Zodiac, *dragon* years are considered especially lucky. The years 2000, 2012 and 2024 are all dragon years -- that is, they occur every 12 years.

Write a function `dragonYear :: Int -> String` that, given a year, returns a message of the format described by the examples below:

```
>>> dragonYear 2000
"that is a dragon year"
>>> dragonYear 2001
"that is not a dragon year"
>>> dragonYear 1988
"that is a dragon year"
```

You may assume that the given year is a positive number. You will not be given zero nor any negative input.

Hint: consider the `mod` function to determine if it is a dragon year or not.

⚠ Make sure your strings are **exactly** the same format as the example strings (including all lowercase, no punctuation, no extra words etc.). The testing is highly sensitive to exact formatting!

Coding Question 3: Run-length Encoding (20 points)

Run-length encoding is a common way to compress strings that have many adjacent repeated characters. Each character is paired with a number that says how many times it is repeated. For example, the string `"nnnnooooo"` has a run-length encoding of `[('n',4), ('o',5)]`.

Write a function:

```
expand :: [(Char, Int)] -> String
```

which, given a run-length encoding like `[('n',4), ('o',5)]`, produces a string like `"nnnnooooo"`. You may assume that the integers in the provided inputs are all positive numbers (not zero and not negative).

Hint: Recall the `replicate` function (see the Prelude documentation linked from your desktop):

```
ghci> replicate 4 'a'
"aaaa"
```

Coding Question 4: Radiation Dosages (22 points)

The health risk from radiation exposure is usually measured in Microsieverts, or μSv. However, to a layperson who is inexperienced with this unit of measure, the health risk can be hard to understand.

We have defined a data type Exposure for more human-readable measures of radiation exposure. Define a function mapping Microsieverts to this type according to the following instructions:

- You may assume the input is not negative, and you will not be tested on any negative input.
- A *banana equivalent dose* is 0.1μSv, the amount of radiation exposure caused by eating one average-sized banana (as bananas are very mildly radioactive). If the radiation dose is zero or low, report the dose with the `Bananas` constructor, rounded down to the nearest `Int`.
- One medical CT scan of the chest is 70000 banana equivalent doses. If the input is equivalent to one CT scan or higher, but not extremely dangerous, report the dose with the `CTScans` constructor, again rounded down to the nearest `Int` .
- If the dose is equivalent to 500 CT scans or higher, return `ExtremeDanger`.

Hint: The Prelude function floor will map a `Double` to the nearest `Int`, rounded down.