Midterm Study Guide

You should read all the notes we have discussed so far (up to but NOT including random numbers), and the corresponding textbook sections. These questions are here to help guide your studies, but are not meant to be exhaustive of everything you should know (though they do try to touch all the areas).

In all "coding" problems below, you must always include a type for each value/function/action you write.

1. Describe 5 ways in which Haskell differs from most other languages.
2. List at least 7 methods that operate on lists, write their type and explain what they do.
3. What are the differences between **lists** and **tuples**?
4. In Haskell a function that is meant to have two integer inputs can be written in two different ways. Describe those ways and the corresponding types for the function.
5. Write the types of the following functions, and describe their use: `zip`, `map`, `filter`, `(++)`, `zip`, `unzip`, `take`, `drop`. Write (possibly recursive) implementations for these functions.
6. Explain what **parametric polymorphism** and **ad-hoc polymorphism** are and how they differ. Provide concrete examples from any language other than Haskell you wish. Also describe how these two features are present in Haskell.
7. Describe what **pattern matching** is and provide examples. Be able to write functions that use pattern matching in various ways.
8. Describe what guards are and how they work. Give examples.
9. What are **type aliases**? What are **custom data types**? How do they differ?
10. Write list comprehensions to perform the following:
    a. Given two lists of integers, it builds a list of all pairs of integers, one from each list, where the first integer is less than the second integer.
    b. Given a list of string, it builds a new list of pairs, where one part of the pair is the string and the other part is the length of that string, but only for those strings whose length is at least 5.
    c. Given a list of integers and a list of operators (e.g. `[(+), (*)]`) return a list of triples `(x, y, z)` where `x` and `y` come from the list of integers, and `z` is the result of applying one of the operators. The final result for a list with `n` integers and a list with `m` operators would have `n*n*m` entries.
    d. Build the list of all positive multiples of 3 (at least three different ways to do it, two with regular expressions and one with the `[x,y..z]` syntax).

11. Write operator sections for the following functions:

    a. Dividing by 3. Also, integer-dividing by 3 (done via the `div` function).
    b. Testing if a number is negative.
    c. Appending the string `"!"` at the end of a string.
    d. Squaring a number.
    e. Testing if a string contains the character `'a'` (using the `elem :: a -> [a] -> Bool` function).
    f. Testing if a character is one of the vowels (use a string containing the vowels).

12. Use function composition and operator sections and/or direct functions to write functions that perform the following:

    a. Compute `3x+2` for a given number `x`.
    b. Find the largest string length for a list of strings.
    c. Test if all numbers in a list are greater than 10 (and get a single True/False answer).

13. Write (recursive, maybe using pattern matching or guards, possibly with a helper) functions for the following:

    a. Test if a list of numbers is in decreasing order.
    b. Find the largest of a list of numbers (error for empty list).
    c. Reverse a list.
    d. Split a list after n characters (n being an input to the function), returning the lists of the two parts, as a pair.
    e. Split a list in two parts (returning a pair), one list containing the negative numbers, the other containing the positive numbers, and discarding any zeroes.
    f. Given a list, group up every 3 elements together in a tuple and return a list of those tuples (possibly discarding up to 2 elements at the end).

14. Write IO actions, or functions producing IO actions, to perform the following (primitives you can use at `putChar`, `putStr`, `putStrLn`, `getLine`):

    a. Ask the user via a prompt, then read a string in and return the string.
    b. Given a list of string, produce the action that prints each string on its own line.
    c. Reads one line at a time, until an empty line is provided, then returns the resulting list of strings (one for each line).
    d. Reads an integer on its own line (you can use `read` to turn a string into an integer) and returns it.

e. Given a function `f: a–> b` and an action `IO a` create an action `IO b` which uses the provided action to get an `a` value, then uses `f` to obtain a `b` value and returns that.

15. Provide the type and implementation for `foldr`.

16. Implement the following using `foldr`:

    a. `length`

    b. `sum` and `prod` (type `[Int] –> Int`)

    c. `and` and `or` (type `[Bool] –> Bool`)

    d. `any` and `all` (type `(a–>Bool) –> [a] –> Bool`)

    e. `concat` (type `[[a]] –> a`)