Midterm Study Guide

You should read all the notes we have discussed so far (up to but NOT including list comprehensions), 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).

1. What are the primary features that differentiate functional programming from iterative/procedural programing? Demonstrate with examples of each feature that demonstrate the differences.
2. Describe 5 ways in which Haskell differs from most other languages.
3. Describe what **lazy evaluation** is and demonstrate with a specific example where a function behaves differently depending on whether the language uses lazy or strict (non-lazy) evaluation.
4. List at least 7 methods that operate on lists, write their type and explain what they do.
5. Describe what **type inference** is and demonstrate its use in relation to the function `f x y = if x then y + 1 else 4`. Practice with other functions.
6. What are the differences between **lists** and **tuples?
7. 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.
8. Describe what **currying** for a function is, and why it can be useful.
9. Write the types of the following functions, and describe their use: `zip`, `map`, `filter`, `(++)`. Write recursive implementations for these functions.
10. Explain what **parametric polymorphism** and **ad-hoc polymorphism** are and how they differ. Provide concrete examples from any language you wish. Also describe how these two features are present in Haskell.
11. List the standard **type classes** and the functions they contain, along with the type of those functions.
12. How does a **type class** relate to a type?
13. Describe what guarded expressions are, what their general syntax is and how they work.
14. Describe what **pattern matching** is and provide examples.
15. Write, including type class constraints, the types of the following functions, and write recursive implementations for them:

a. A function that finds the maximum element in a list.

b. A function that determines if a list is sorted.

c. A `lookup` function that searches for a key in a key-value list, and if it finds a match returns the corresponding value, as a `Maybe` type.

16. *Without using list comprehensions*, write a function that given two lists of type `[a]` and `[b]` respectively returns a list of type `[(a, b)]` consisting of all possible pairs of an element from the one list and an element from the other list.

17. A recursion can be **numeric** or **structural**, and can be **stateless** or **stateful**. Describe what these mean, and provide examples for all four possible combinations (e.g. a stateless numeric recursion example, a stateless structural recursion example and so on).

18. Describe via an example how stateful recursion is very analogous to using a for loop in other languages. Identify particular elements of this analogy.

19. What are **type aliases**? What are **custom data types**? How do they differ?

20. Describe the `Maybe` type and why it is useful. What problem does it solve? What are its advantages over other solutions to the same problem?