COMP304 Programming Languages (2016)
Assignment 1 : Basic Haskell (Due midnight, Friday 29 July)

This assignment asks you to write some small programs in Haskell. The aim is to help you to understand how you can use the basic constructs of Haskell we have looked at so far. You should think about how the programs can be implemented using these constructs, try to make your programs simple and easy to read, and to be reasonably efficient (e.g. do not use a quadratic time algorithm if a straightforward linear time one can be used instead). If you can see different ways of implementing/expressing the solution, briefly describe a couple of possibilities and explain why you chose a particular one, or show a couple of versions in detail and compare them.

You should submit Haskell source code for all of the programs, as a literate Haskell script called `comp304a1.lhs`, using the online submission system. All Haskell code must be suitably commented to explain how it is intended to work, and should include code for test cases. Other discussion of what you have done and the results you obtained, may be be embedded as comments or presented in a separate document. You may use predefined Haskell functions where appropriate, unless doing so makes the exercise pointless — if in doubt, ask me!

Parts of the assignment will be marked automatically, so it is important that you define functions with the name and argument(s) indicated in the questions. Also make sure that everything you submit shows your name, the course code and the assignment number.

1. **Variations on a search** (15 marks 5 each)

   (a) Write a function `count` returns the number of times that a given item occurs in a list of items. For example, `count 1 [1,2,1,2,1]` should return `3`, while `count 1 []` should return `0`.

   (b) Write a function `allPos` which returns a list of all positions at which a given item occurs in a list of items, in the order that they occur. For example, `allPos 1 [1,2,1,2,1]` should return `[1,3,5]`, while `allPos 3 [1,2,1]` should return `[]`.

   (c) Write a function `firstLastPos` which returns a pair giving the positions of the first and last occurrences of a given item in a list of items. The function should return `(i,i)` if the item only occurs once, at position `i`, and `(0,0)` if the item does not occur in the list. For example, `firstLastPos 1 [1,2,1,2,1]` should return `(1,5)`, `firstLastPos 2 [1,2,1,2,1]` should return `(2,4)`, `firstLastPos 5 [1,2,5,2,1]` should return `(3,3)`, while `lastPos 3 [1,2,1]` should return `(0,0)`.

2. **Sorting** (20 marks 10 each)

   For this question, you are to implement two algorithms in Haskell to sort a list into ascending order (or, more precisely, non-descending order, so as to allow for lists with repeated elements).

   (a) Write a function `sort1` which implements an n^2 sorting algorithm, e.g. insertion sort, selection sort or bubble sort.

(b) Write a function `sort2` which implements an $n \log n$ sorting algorithm, e.g. quicksort or merge sort.

You should explain why you chose the algorithms that you implement, and discuss alternative ways of implementing it that you considered.

For bonus marks, you may implement additional sorting algorithms or give versions of the algorithms you have already implemented using more advanced features of Haskell such as list comprehensions and high-order functions.

3. **Implementing a map** (10 marks)

   Implement a map data structure where the map is stored as a list of key-value pairs. Your implementation should provide the following functions:

`emptyMap :: Map a b`	Return an empty map
`hasKey :: a -> Map a b -> Bool`	Check if a given key is defined in a map
`setVal :: a -> b -> Map a b -> Map a b`	Set the value for a key in the map
`getVal :: a -> Map a b -> b`	Get the value of a given key in a map
`delKey :: a -> Map a b -> Map a b`	Delete a key (and its value) from a map

   `Map a b` is the type of maps with keys of type `a` and values of type `b`. We will see how to define such types in lectures.

   `setVal` should over-ride any existing value for the given key, while `getVal` and `delKey` should give an error if the key is not present.

   Keys are not required to be stored in order, so if `setVal` inserts a new key into the map, it should add the new pair at one end of the list or the other - whichever is easiest.

4. **Building a map** (5 marks)

   Write a function `buildMap` which takes a list of items and returns a map containing all of the items in the list and the number of times that each one occurs. For example, given the input `[1,2,3,2,1]`, the function would return the map `[(1,2), (2,2), (3,1)]`.

   This should use your map implementation from Question 3.

Total marks possible = 50.