Assessment Brief Proforma

1. Module number	*SET07106*
2. Module title	*Maths for Software Engineering*
3. Module leader	*Peter Chapman*
4. Tutor with responsibility for this Assessment Student's first point of contact	*As above*
5. Assessment	*Practical coursework*
6. Weighting	*40% of module assessment*
7. Size and/or time limits for assessment	*None*
8. Deadline of submission	Your attention is drawn to the penalties for late submissions *7th April 2017 before 2355*
9. Arrangements for submission	*Upload your .hs file (appropriately renamed – see detailed instructions on later pages) to the Moodle submission page.*
10. Assessment Regulations	All assessments are subject to the University Regulations

11. The requirements for the assessment	*Rename the cwk.hs file to your _student_number.hs and complete the exercises.*
12. Special instructions	*None*
13. Return of work and feedback	*The solutions, and general feedback, will be provided by the end of week 15.*
14. Assessment criteria	*Each function will be tested on 5 inputs. The following sheets give details of how many marks each test is worth. If your function returns the correct output for the test, you get the assigned marks.*

SET07106 - Haskell Coursework

General Remarks

In this coursework you will be asked to create functions which solve specific problems. Some of these problems will be variations of problems you have seen in the practical exercises, and some will be derived from material we have covered in the lectures. You will be given the type-signatures, the order of the arguments, and you will be required to replace the definitions, of the functions in the associated `cwk.hs` file.

> **DO NOT CHANGE THESE TYPE SIGNATURES OR THE ORDER IN WHICH ARGUMENTS APPEAR**

The coursework will be marked automatically, and if you change the type signatures or order of the arguments then you will make it impossible for you to get the answers correct. Your code will not be checked, which gives you freedom to use whichever method you want to create your functions (i.e. list comprehension, using `map`, using recursion, if done correctly, will give the same results, and hence will gain the same marks.)

The marks for each function are contained in this document. Each function will be tested on 5 different inputs, and your mark displayed is the mark each *test* carries. The total number of marks for the coursework is then 100.

Instructions

- Download the file `cwk.hs`, and change the file name to `YOURSTUDENTNUMBER.hs`. For example, if your student number was 40101916, then your file would become `40101916.hs`.
- For each function, replace the dummy definition in the file[1]. You may add as many extra functions to your file as you wish. However, `import` statements are not allowed. If you wish not to answer a particular question, just leave the dummy definition as it is.
- Upload your file to the submission point on Moodle by **2355 on Friday 7th April 2017**.
- To allow you to judge the effectiveness of your functions, in the file `cwk.hs` are some examples of sample behaviour for your functions, which give you the chance to self-assess.

[1] Every function is initially defined as giving a helpful error message.

Questions

Question 1 - Sets

Write a function (`complement`) which, given a set A and a *prospective* universal set U, returns the complement of A with respect to U, wrapped in the `Just` type constructor. Note that if the set A is not a subset of U, then you should return `Nothing`. (**3 marks**)

Multisets are sets which allow multiple copies of the same element, but are unordered. In list form, for example, `[1,1,2,3]` is equal to `[1,2,3,1]`, but not to `[1,2,3]`. Write a function (`mUnion`) that returns the multiset union of two multisets. Note that you do not need to worry about the order of the elements in your return value. (**2 marks**)

Question 2 - Functions and Relations

The *reflexive closure* of a relation R is the smallest relation bigger than R which is reflexive. In other words, it is R with whatever pairs added to make R reflexive. Write a function (`reflClosure`) which takes a list of pairs (standing for R) and returns a list of pairs which is the reflexive closure of R. You do not need to worry about the order in which pairs appear in your return value. (**3 marks**)

Question 3 - Combinatorics

Write a function (`choose2`) which takes a list of distinct integers, `xs`, and returns a list of pairs of integers which encode *all* possible ways of choosing two values from `xs`. Order each pair so that the smaller integer appears first in that pair; i.e. $(3, 4)$ not $(4, 3)$. (**2 marks**)

Question 4 - Primes

Write a function (`factors`) which takes an integer n and returns a list of all the factors of n. (**2 marks**)

Write a function (`primeFactorisation`) which takes an integer n and returns a list representing the prime factorisation of n. In other words, it is possible that elements in your list will appear more than once. It should be the case that if you multiply all of the numbers in your list together, you get n. (**2 marks**)

Write a function (`nextPrime`) which takes an integer n and returns the next prime number strictly larger than n. (**1 mark**)

Question 5 - RSA

Write a function (`eTotient`) which takes an integer n, and returns the number of integers less than n which are coprime with n. (**2 marks**)

Write a function (`encode`) which takes two numbers p and q, a message m and a number e and, if p, q and e are suitable for use in the RSA encryption algorithm, returns the encoded message wrapped in a `Just` type constructor. If they are unsuitable, return `Nothing`. (**3 marks**)