# Assessment Brief Proforma

| | |
|---|---|
| **1. Module number** | *SET07106/SET07406* |
| **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<br>*17th April 2020 before 1500* |
| **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/SET07406 - 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 $100$. There are special instructions for question $3$, which you should read carefully.

## 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 **1500 on Friday 17th April 2020**.
- 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 (`bigUnion`) which, given a list of lists $A, B, C, \ldots$, returns the union of all of them. As an example, `bigUnion [[a,b,c],[c,d,e],[f,g,c]]` would return `[a,b,c,d,e,f,g]`. The order of the elements in your resulting list is not important. (**1 marks**)

Write a function (`bigIntersection`) which, given a list of lists $A, B, C, \ldots$, returns the intersection of all of them. As an example, `bigIntersection [[a,b,c],[c,d,e],[f,g,c]]` would return `[c]`. The order of the elements in your resulting list is not important. (**2 marks**)

Write a function (`howManySetsContain`) that takes as input a list of lists of type `a`, and returns a list of pairs `(a,Int)` where each `Int` represents the number of the lists that each element belongs to. As an example `howManySetsContain [[a,b,c],[c,d,e],[f,g,c]]` would return `[(a,1),(b,1),(c,3),(d,1),(e,1),(f,1),(g,1)]`. The order of the elements in the resulting list is not important. (**2 marks**)

### Question 2 - Functions and Relations

Write a function (`oneHop`) which takes an element of type `a`, a list of pairs of type `(a,a)`, and returns a list of elements of type `a`. The list of pairs represents a relation, and the resulting list represents everything that is related to the initial element. For example:

```
oneHop 2 [(1,2),(2,3),(2,4),(4,1)]
```

should return `[3,4]`. The order of the elements in your return list is not important. (**1 mark**)

A *path* is a list of elements, where the next element in the list is related to the previous one by a given relation. For example, if $R = \{(1,2),(2,3),(3,4),(1,4),(2,4)\}$ then the following are all paths: $[1,2],[1,2,4].[2,3,4]$, and so on. Write a function (`nextSteps`) which takes a list of elements of type `a`, a list of pairs of elements of type `(a,a)`, and returns a list of lists representing the paths which are one element longer than the input list. For example:

```
nextSteps [1,2,4] [(1,2),(2,4),(4,1),(4,3),(2,5),(3,5)]
```

would return `[[1,2,4,1],[1,2,4,3]]`. The order of the paths is not important. (**2 marks**)

When a path has a repeating segment, we call it a *cycle*. For example, $[1,2,3,2]$ and $[1,2,3,1]$ are both cycles. Write a function (`allPathsFrom`) which takes an element of type `a`, a list of pairs of type `(a,a)` (the relation), and returns a list of lists representing all possible paths from the input element according to the input relation. Cycles should stop at the first repeating element. For example:

```
allPathsFrom 1 [(1,2),(2,1),(1,3),(2,3)]
```

would return `[[1,2],[1,3],[1,2,1],[1,2,3]]`. (**3 marks**)

QUESTION 2 CONTINUES ON NEXT PAGE

A *complete cycle* is one where the whole path repeats. For example, $[1, 2, 3, 1]$ is a complete cycle, but $[1, 2, 3, 4, 3]$ is not. Write a function (`shortestCycle`) which has an element of type `a` and a list of pairs of type `(a,a)` (the relation), which returns a list representing the shortest complete cycle from the input element using the input relation. This return list should be wrapped in the `Maybe` constructor. If no complete cycle exists, the function should return `Nothing`. Your function will only be assessed on inputs where there is a *unique* shortest cycle. (**1 mark**)

### Question 3 - Primes

***Special instructions.*** *For this question, the test cases are ranked in terms of size:*

- *a* $2$*-digit number (**1 mark**)*
- *a* $4$*-digit number (**1 mark**)*
- *a* $5$*-digit number (**2 marks**)*
- *a* $6$*-digit number (**2 marks**)*
- *a* $10$*-digit number (**4 marks**)*

*There is a time-limit set on the automatic marking script: if your submission times out, the entirety of the question causing the time-out will be commented out and the script run again. In other words, if you choose to attempt the* $10$*-digit number parts and they time-out, you will get no credit for otherwise correct answers to the* $4$*-digit parts, for example. The time-limit is set to* $2$ *minutes per submission. You then need to decide whether or not it is worth the risk to try to solve the larger numbers.*

Write a function (`lastPrime`) which takes an integer $n$ and returns the biggest prime strictly smaller than $n$.

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 $n$.

### Question 4 - 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`. (**2 marks**)