Edinburgh Napier
UNIVERSITY

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 ***30th April 2021 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. In particular, the submissions of the entire cohort will be checked for similarity against each other.

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* ***14th May 2021****.*
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 2021
First Diet

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. You do not receive marks for coding style, 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 submissions for the entire cohort will be passed through a similarity-checker.

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, rather than deleting it.
- Upload your file to the submission point on Moodle by **1500 on Friday 30th April 2021**.
- 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 and Lists

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 mark**)

A *partial sum* of a list is the sum up to a given a given index. For example, the fourth partial sum of `[3,5,1,6,2,8,1]` is $3+5+1+6 = 15$. Write a function (`partialSums`) which, given an input of type `[Int]`, returns a list of type `[Int]` representing the partial sums. For example,

```
partialSums [3,5,1,6,2,8,1]
```

would return `[3,8,9,15,17,25,26]`.(**2 marks**)

Write a function, `maxIndex`, which returns the index at which the partial sum is maximised. The function has an input of type `[Int]`, and gives an output of type `Maybe Int`. For example,

```
maxIndex [-4,3,0,3,-2,1]
```

would return `Just 4`. If there are multiple indices where the partial sum is maximised, return the first index. In other words, `maxIndex [-4,3,0,3,-1,1,0,-2,1]` would still return `Just 4`. If there is no maximising index, the function should return `Nothing`. (**2 marks**)

Question 2 - Functions and Relations

A *commutative binary function* is one where the order of the arguments does not matter. For example, addition is commutative, because $4+3$ is equal to $3+4$; whereas subtraction is not commutative, since $3-4$ is not equal to $4-3$. Given a partial binary function `f`, represented as a list `[((Int,Int),Int)]`, write a function called `makeCommutative` that returns a list `[((Int,Int),Int)]` representing the smallest function that:

- contains `f`,
- is total,
- is the identity function when applied to the same argument twice (i.e. $g(a, a) = a$), and
- is commutative.

In other words, extend `f` so that it becomes a total commutative function. For example,

```
makeCommutative [((1,2),5),((3,2),1),((1,3),3)]
```

should return:

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

The order within the list is not important. **(3 marks)**

QUESTION 2 CONTINUES ON NEXT PAGE

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 binary 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. (**1 mark**)

Write a function called `allElementsReachable`, which takes an input n of type `Int`, an element x of type `a`, and a binary relation rs of type `[(a,a)]`. The function returns a list of the (unique) elements which can be reached in exactly n steps from x, using the relation rs. For example:

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

should return `[2]`. This is because the only path of length 3 from the element 2 is $2 \to 4 \to 1 \to 2$. By definition, `allElementsReachable 0 x rs = [x]`. The order of the list is not important.

(**2 marks**)

QUESTIONS CONTINUE ON NEXT PAGE

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 (**3 marks**)*
- *a 7-digit number (**3 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 7-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[2]*. You then need to decide whether or not it is worth the risk to try to solve the larger numbers.*

Write a function (lastPrimes) which takes an integer n and returns the **three** biggest primes strictly smaller than n.

Write a function (primeFactors) which takes an integer n and returns a list of type Maybe Int, containing the unique prime factors. In other words, prime factors will appear in your list at most once. If there are no strictly smaller prime factors of n, then the function returns Nothing.

Question 4 - RSA

Note. *Inefficient solutions to the first part of this question can cause timeouts, similar to question 3. If your code times-out, then the first part of the question will be discarded during marking.*

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**)

[2] All submissions will be run on a Intel Quad-core i7, running Ubuntu 20.04.