Module Code
COM00025I

BEng, BSc, MEng and MMath Degree Examinations 2020-21

Department Computer Science

Title Software 3 - Formative 1

Time Allowed 24 Hours (NOTE: papers late by up to 30 minutes will be subject to a 5 mark penalty; papers later than 30 minutes will receive 0 marks).

Time Recommended ONE hour

Word Limit `Not Applicable.`

Allocation of Marks:

This paper has one question and it is worth 40 marks.

Instructions:

Candidates should answer **all** questions using Haskell. Failing to do so will result in a mark of 0%. For every question, a template has been provided (*Q<qnumber><roman-numeral>[<partnumber>].hs*) which corresponds to the question number, and that is the **only** file to modify and submit. Each **.hs** file has the question description, declaration – implementation as undefined and a testing function.

You may use any values defined in the standard 'Prelude', but you may not import any other module unless explicitly permitted. You may use Neil Mitchell's $[Hoog\lambda e]$ to discover useful functions in the *Prelude*.

Where tests accompany the English description of the problem, it is not necesarily enough for your implementation to pass the tests to be correct. Partially functional solutions with *undefined*, may be considered for part of the full marks.

Download the paper and the required source files from the VLE, in the "Assessment" section. Once downloaded, unzip the file. You **must** save all your code in the `FormativeOne` folder and the related *Q<qnumber><roman-numeral>[<partnumber>].hs* file provided. **Do not** save your code anywhere else other than this folder.

Submit your answers to the Department's Teaching Portal as a single **.zip** file containing the `FormativeOne` folder and its files. Failing to include the `FormativeOne` folder will result in a mark of 0%. Other archival format such as .rar and .tar files will not be accepted.

A Note on Academic Integrity

We are treating this online examination as a time-limited open assessment, and you are therefore permitted to refer to written and online materials to aid you in your answers.

However, you must ensure that the work you submit is entirely your own, and for the whole time the assessment is live you must not:

- communicate with departmental staff on the topic of the assessment
- communicate with other students on the topic of this assessment
- seek assistance with the assignment from the academic and/or disability support services, such as the Writing and Language Skills Centre, Maths Skills Centre and/or Disability Services. (The only exception to this will be for those students who have been recommended an exam support worker in a Student Support Plan. If this applies to you, you are advised to contact Disability Services as soon as possible to discuss the necessary arrangements.)
- seek advice or contribution from any third party, including proofreaders, online fora, friends, or family members.

We expect, and trust, that all our students will seek to maintain the integrity of the assessment, and of their award, through ensuring that these instructions are strictly followed. Failure to adhere to these requirements will be considered a breach of the Academic Misconduct regulations, where the offences of plagiarism, breach/cheating, collusion and commissioning are relevant: see AM1.2.1 *(Note this supercedes Section 7.3 of the Guide to Assessment)*.

1 (40 marks) Short questions

(i) [1 mark] The code must be written in the provided file *Q1i.hs*.

Consider:

```
type Predicate a = a -> Bool
```

Write a function `isPass` that takes a list of integers and returns `True` if the sum is greater than or equal to 40 and `False` otherwise. Your solution should satisfy:

```
testP :: Bool
testP = (isPass [] == False) &&
(isPass [2, 5, 6, 7] == False) &&
(isPass [12, 15, 16, 17] == True) &&
(isPass [5, 8, 12, 15] == True)
```

(ii) [1 mark] The code must be written in the provided file *Q1ii.hs*.

Consider:

```
type Predicate a = a -> Bool
```

Write a function `isAlphabet` that takes a string and returns `True` if the string has only Latin alphabetic characters (as used for standard English) and `False` otherwise.

Your solution should satisfy:

```
testAl :: Bool
testAl = (isAlphabet "" == True ) &&
(isAlphabet "hello!" == False) &&
(isAlphabet "hello" == True) &&
(isAlphabet "Hello" == True) &&
(isAlphabet "SOF3" == False) &&
(isAlphabet "Software" == True)
```

(iii) [2 marks] The code must be written in the provided file *Q1iii.hs*.

Write a function `cmbList`, such that, given two lists, it returns a single list with all odd-position elements taken from the first list and even-position elements from the second list. Note, lists are *0-indexed*.

Your solution should satisfy:

```
testcmbList :: Bool
testcmbList =
(cmbList ['s'] "comp"  == "cs") &&
```


Continued.

```
(cmbList "otae""sfwr" == "software") &&
(cmbList [1, 3, 5] [0, 2, 4, 6]  == [0,1,2,3,4,5]) &&
(cmbList ["1", "2", "3"]["THE", "SOF", "SYS"] ==
["THE","1","SOF","2","SYS","3"])
```

(iv) [2 marks] The code must be written in the provided file *Q1iv.hs*.

Given two lists over a `Num` type, `x`, return a single list whose length is the same as the minimum of the two and each element is the product of the numbers at the same position from the two lists.

Your solution should satisfy:

```
testcmbProd :: Bool
testcmbProd =
(cmbProd [] [5, 6]  == []) &&
(cmbProd [2, 3, 4] [5, 6]  == [10,18]) &&
(cmbProd [0.23, 3.4, 7.88, 9.21] [3.4, 5] == [0.782,17.0]) &&
(cmbProd [0.23, 3.4, 7.88, 2*0.3] [3.4, 1.3, 2.1, 2]  ==
[0.782,4.42,16.548000000000002,1.2])
```

(v) [3 marks] The code must be written in the provided file *Q1v.hs*.

Write a function `sqDiff` that returns a list of squares of the difference between two consecutive numbers in a list, where the first number is greater than the second number in the list. Your solution should satisfy:

```
testsqDiff :: Bool
testsqDiff =
(sqDiff []               == []) &&
(sqDiff [4, 6]           == []) &&
(sqDiff [6, 4]           == [4]) &&
(sqDiff [4, 6, 3, 1, 8]  == [9, 4])
```

(vi) [3 marks] The code must be written in the provided file *Q1vi.hs*.

Write a function `maybe2int :: [Maybe Int] -> Int`, which returns the sum of all the integers in the list. The presence of `Nothing` values does not affect the result.

Your solution should satisfy:

```
testM2int :: Bool
testM2int =
(maybe2int []        == 0) &&
(maybe2int [Just 23] == 23) &&
(maybe2int [Nothing] == 0) &&
(maybe2int [Just 2, Nothing, Just 3, Just 16, Nothing] == 21)
```

(vii) [4 marks] The code must be written in the provided file *Q1vii.hs*.

Write a function `cmb :: Num a => String -> [a] -> [a] -> [a]`, which returns a list as long as the shorter of the two input lists with all odd-position elements taken from the first list and even-position elements from the second list when the string is *"List"* or the product of the numbers at the same position from the two lists when the string is *"Prod"*. Any other string apart from *"Prod"* and *"List"* returns an empty list.

Your solution should satisfy:

```
testcmb :: Bool
testcmb = (cmb "Hello" [] [] == []) &&
(cmb "Prod" [2, 3] [] == []) &&
(cmb "List" [] [2, 3] == []) &&
(cmb "Prod" [2, 3, 4] [5, 6] == [10,18]) &&
(cmb "List" [2, 3, 4] [5, 6] == [5,2,6,3]) &&
(cmb "Haskell" [2, 3, 4] [5, 6] == []) &&
(cmb "Prod" [2, 3, 4] [5, 6, 7] == [10,18,28]) &&
(cmb "List" [2, 3, 4] [5, 6, 7] == [5,2,6,3,7,4])
```

(viii) [8 marks] Consider the type of Stage 1 students record `CS1`:

```
data CS1 = Student {name :: String, the1, sof1, sys1 :: Int}
```

A section of records for five Stage 1 students are available in the database `s1Db`. Each record for a student has the student's name and marks for three modules (THE1, SOF1 and SYS1).

```
s1Db :: [CS1]
s1Db = [Student {name = "Beth", the1 = 65, sof1 = 58, sys1 = 79},
Student {name = "Adam", the1 = 55, sof1 = 68, sys1 = 61},
Student {name = "Lisa", the1 = 60, sof1 = 72, sys1 = 65},
Student {name = "Will", the1 = 71, sof1 = 52, sys1 = 49},
Student {name = "Mark", the1 = 67, sof1 = 78, sys1 = 50}]
```

(a) [2 marks] The code must be written in the provided file *Q1viiia.hs*.

Write a function `the1Mk` that takes any `CSYear1` records like `s1Db` and returns a list of pairs of the name and THE1 mark of each student in the same order as they appear in the database.

Your solution should satisfy:

```
test1MK :: Bool
test1MK = the1Mk s1Db == [("Beth",65),("Adam",55),
                          ("Lisa",60),("Will",71),("Mark",67)]
```

(b) [3 marks] The code must be written in the provided file *Q1viiib.hs*.

Write a function `tSOF1` that takes `s1Db` and return a list of tuples of the name, SYS1 mark and THE1 mark for all students who scored more than *70* in **SOF1**.

Your solution should satisfy:

```
testSOF1:: Bool
testSOF1 = tSOF1 s1Db ==
      [("Lisa",65,60),("Mark",50,67)]
```

(c) [3 marks] The code must be written in the provided file *Q1viiic.hs*.

Write a function `avgSYS1` that returns the average mark for **SYS1**.

Your solution should satisfy:

```
testavg:: Bool
testavg = avgSYS1 s1Db == 60.8
```

(ix) [4 marks] The code must be written in the provided file *Q1ix.hs*.

Write a function `vowelDigit` which returns `True` when applied to a string with alternating vowels and digits and `False` otherwise. The vowel comes first in each pair and the input list must have even length. Your solution should satisfy:

```
testvowelDigit :: Bool
testvowelDigit =
(vowelDigit ""         == False) &&
(vowelDigit "a2"       == True) &&
(vowelDigit "aa22"     == False) &&
(vowelDigit "a2a2"     == True) &&
(vowelDigit "b2b2"     == False) &&
(vowelDigit "a2a21"    == False) &&
(vowelDigit "2a4o"     == False) &&
(vowelDigit "a2ab2"    == False) &&
(vowelDigit "a2o5u8A0" == True) &&
(vowelDigit "b2o5u8A0" == False)
```

(x) [7 marks] Consider:

```
data BinTree x = Lf Int | Branch (BinTree x) x (BinTree x)
        deriving (Eq, Show)
```

A tree is *balanced* if the length of the path from the root to the deepest leaf on the left is the same as the right and each subtree has the same structure. For a *"Length-regular"* tree, the value at a leaf is the length of the path to the leaf.

(a) [1 mark] The code must be written in the provided file *Q1xa.hs*.

Define '`nullBR`' to be the smallest possible balanced, length-regular tree.

(b) [4 marks] The code must be written in the provided file *Q1xb.hs*.

Write a function `isTreeBal` which returns `True` when applied to a balanced tree, and `False` otherwise. Remember, for a length-regular, balanced tree the value in a leaf is the length of the path from the root to the leaf. You can *assume* that the tree is length-regular when testing balance.

Your solution should satisfy:

```
testBal :: Bool
testBal = (isTreeBal (nullBR) == True ) &&
(isTreeBal (Branch (Lf 1) 1 (Branch (Lf 2) 2 (Branch (Lf 3) 3 (Lf 3))))
  == False ) &&
(isTreeBal (Branch (Branch (Lf 2) 1 (Lf 2)) 2 (Branch (Lf 2) 3 (Lf 2)))
 == True )
```

(c) [2 marks] The code must be written in the provided file *Q1xc.hs*.

Write a function `treeNodes` which returns the total number of interior nodes in a given tree. Note, the tree need not be balanced.

Your solution should satisfy:

```
testN :: Bool
testN =
(treeNodes nullBR == 0) &&
(treeNodes (Branch (Lf 1) 3 (Lf 1)) == 1) &&
(treeNodes (Branch (Lf 1) 3 (Branch (Lf 2) 8 (Lf 2))) == 2) &&
(treeNodes (Branch (Lf 1) 1 (Branch (Lf 2) 2 (Branch (Lf 3) 3 (Lf 3)))) == 3) &
(treeNodes (Branch (Branch (Lf 2) 1 (Lf 2)) 2 (Branch (Lf 2) 3 (Lf 2))) == 3)
```

(xi) [5 marks] The code must be written in the provided file *Q1xi.hs*.

Write a function `toBarcode` that takes a binary string (a string composed of 0s and 1s only) as parameter and returns a string representing a bar-code. The 0s are transformed into **'.'** and 1s into **'|'**. In addition, the function must return Nothing if the string contains a character that is not a 0 or a 1. You can also assume that a string (including an empty string) is always given.

Your solution should satisfy:

```
testBcode:: Bool
testBcode =
(toBarcode ""    == Just "") &&
```


Continued.

```
(toBarcode "00" == Just "..") &&
(toBarcode "1111" == Just "||||") &&
(toBarcode "0010111" == Just "..|.|||") &&
(toBarcode "01120" == Nothing) &&
(toBarcode " " == Nothing)
```

End of examination paper