Module Code
COM00025I

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

Department Computer Science

Title Software 3

Issued: 09:00am on Wednesday 19th May 2021.

Submission due: 09:00 on Thursday 20th May 2021.

Feedback & Marks due: Thursday 17th June 2021.

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).

Notification of errors in the paper may be made up to **one hour** after the start time. If you wish to raise a possible error it must be done through ⟨`cs-exams@york.ac.uk`⟩ with enough time for a response to be considered and made within the first hour.

Time Recommended THREE hours

Word Limit `Not Applicable.`

Allocation of Marks:

This paper has two questions, each worth 45 marks. In addition, there are **10 marks** allocated to style, clarity, and quality of code.

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 `ClosedExamination` 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 `ClosedExamination` folder and its files. Failing to include the `ClosedExamination` folder will result in a mark of 0%. Other archival format such as .rar and .tar files will not be accepted.

If a question is unclear, answer the question as best as you can, and note the assumptions you have made to allow you to proceed. Please inform ⟨`cs-exams@york.ac.uk`⟩ about any suspected errors on the paper immediately after you submit.

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 (45 marks) Short questions

(i) [1 mark] Write a function `isHaskell` that takes a string and returns `True` if the string is "Haskell" and `False` otherwise.

Your solution should satisfy:

```
testHaskell :: Bool
testHaskell =
(isHaskell "Haskell" == True) &&
(isHaskell "Software" == False) &&
(isHaskell ""   == False) &&
(isHaskell "haskell" == False)
```

(ii) [1 mark] Write a function `lowerVowel` that takes a string and returns `True` if the string has only lowercase vowels (a, e, i, o, u) and `False` otherwise.

Your solution should satisfy:

```
testlv:: Bool
testlv = (lowerVowel "ue" == True) &&
(lowerVowel "ueA" == False) &&
(lowerVowel "uea" == True)
```

(iii) [2 marks] Write a function `prodCube` that computes the product of the cube of numbers in a list that are divisible by 2 but not divisible by 4. Your solution should satisfy:

Your solution should satisfy:

```
testprodCube :: Bool
testprodCube = (prodCube []     == 1)
        && (prodCube [4, 8]        == 1)
        && (prodCube [4, 6, 8]     == 216)
        && (prodCube [4, 6, 8, 12] == 216)
        && (prodCube [2..11]       == 1728000)
```

(iv) [2 marks] Write a function `consDiff` that computes the difference between the number of consonants "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ" and the number of digit characters in a string. Your solution should satisfy:

```
testconsDiff :: Bool
testconsDiff =
(consDiff ""                          == 0)
&& (consDiff "SOf3in2021"             == -2)
&& (consDiff "Software123andTheory123" == 5)
&& (consDiff "HASkellprogramming2021"  == 9)
```

(v) [4 marks] The International Air Transport Association's (IATA) Location Identifier is a unique

and all uppercase 3-letter code used in aviation to identify an airport, for example LHR, JFK and BHX. For the purposes of this question, IATA location identifier should consist of only consonants "BCDFGHJKLMNPQRSTVWXYZ".

(a) [2 marks] Write a function `isIATA` that tests whether or not a string is an IATA location identifier.

Your solution should satisfy:

```
testisIATA :: Bool
testisIATA =
(isIATA ""     == False) &&
(isIATA "MAN"  == False) &&
(isIATA "LHR"  == True) &&
(isIATA "LHRT" == False) &&
(isIATA "lhr"  == False) &&
(isIATA "JFK"  == True) &&
(isIATA "BHX"  == True)
```

(b) [2 marks] Write a function `countIATA` that counts the number of IATA location identifiers in a list of strings.

Your solution should satisfy:

```
testcountIATA :: Bool
testcountIATA =
(countIATA ["LHR"]                                    == 1) &&
(countIATA ["LHR", "Lhr", "MAN", "JFK", "", "jfk"]    == 2) &&
(countIATA ["LHR", "BHX", "MAN", "JFK", "ACC", "LRHT"] == 3)
```

(vi) [5 marks]

Consider the type of students record ‘Students‘:

```
type Students = [(Name, Age, College)]
type Name = String
type Age = Int
type College = String
```

A bus will normally carry students between campuses and records of all the students travelling on a particular bus are maintained. If a student boards the bus, his/her record is added to the bus database and the record is removed when the student gets off the bus. Below is the current state of **bus 66** in a form of a database with students records *onBus66*.

```
onBus66 :: Students
onBus66 =
```

```
[("Zain", 18, "Halifax"), ("Julia", 20, "Constantine"),
("Mandy", 22, "Goodricke"), ("Jack", 24, "Constantine"),
("Emma", 21, "Langwith"), ("Zack", 19, "Halifax"),
("Alice", 21, "Halifax"), ("Bob", 19, "Alcuin"),
("Lui", 22, "Goodricke")]
```

(a) [1 mark] Write a function `colleges` that takes an age and returns a list of all colleges with students currently onboard bus 66 with that age.

(b) [2 marks] Write a function `joinBus` that adds the record of a student to the database once the student gets onboard the bus.

(c) [2 marks] Write a function `offBus` that removes the record of a student from the database once he/she gets off the bus. There shouldn't be any change to the database if the record does not exist.

(vii) [5 marks] Consider the type

```
type BoolD a = (Bool, a)
```

A value `(True, x)` is to be interpreted as `x` is a valid result and `(False, x)` as `x` is invalid and should not be used. A better way of representing such values is to use a `Maybe` type.

(a) [1 mark] Write a function `bd2m` that converts `BoolD a` to `Maybe a`.

(b) [3 marks] Write a function `bDSum` that returns a `BoolD Int`; the sum of all digits in a list of characters. Assume the sum is zero for a list without digits.

(c) [1 mark] Write a function `mBSum` which uses the function `bd2m` and `bDSum` to return a `Maybe` type from the sum of all digits in a list of characters as described in question (b).

(viii) [9 marks] Consider

```
data TreeP a = Leaf Int | Node (TreeP a) a (TreeP a) deriving (Eq, Show)
```

In a *regular tree* the value in a leaf is the length of the path from the root to the leaf. Hence we define:

```
emptyTreeP :: TreeP a
emptyTreeP = Leaf 0
```

In an *ordered* tree, all the values in the left-hand subtree are strictly smaller than the value at the root, all the values in the right-hand subtree are strictly greater than the value at the root, and both subtrees are ordered.

(a) [6 marks] Write a function `itp` to insert a value into a regular, ordered tree.

(b) [3 marks] Write a function `treeList` that returns a list with the values of a regular, ordered tree in ascending order.

(ix) [8 marks] Consider

```
data DBTree = DBLeaf | DBNode DBTree (Int, String) DBTree deriving (Eq, Show)
```

In a *binary search tree* if *k* is the key in a node, then all keys in the left subtree must be less than *k*, and all the keys in the right subtree must be greater than *k*. Consider a database table (like **`smallDB`**) with student-id and name pair, represented as a binary search tree.

```
smallDB :: DBTree
smallDB = DBNode (DBNode DBLeaf (2,"James") DBLeaf) (3,"Maxwell")
   (DBNode DBLeaf (6,"Helen") DBLeaf)
```

Write a function `stdUpdate` which takes a student-id, a new name and a database organised as a binary search tree, and returns a new tree with the student's new name. However, you are expected to return an error message if a valid record with the provided student-id does not exist in the database. You will need to define an appropriate type for `` `Error` ``, as part of your solution.

Your solution should satisfy:

```
testUpdate :: Bool
testUpdate =
(stdUpdate 6 "Abi" smallDB == Right (DBNode (DBNode DBLeaf (2,"James")
 DBLeaf) (3,"Maxwell") (DBNode DBLeaf (6,"Abi") DBLeaf))) &&
(stdUpdate 8 "Mandy" smallDB == Left "There is no such student with ID: 8")
```

(x) [8 marks]

Recall the definition of the 'ProofLayout' type constructor:

```
infixr 0 :=: -- the fixity and priority of the operator
data ProofLayout a = QED | a :=: ProofLayout a deriving Show
```

Consider the definitions of the *Prelude* functions:

```
head :: [a] -> a
head (x:_) = x      -- head.0

foldr :: (a->b->b) -> b -> [a] -> b
foldr _ z []     = z                  -- foldr.0
foldr f z (x:xs) = f x (foldr f z xs) -- foldr.1

const :: a -> b -> a
const x y = x -- const.0
```

Define:

```
caput :: [a] -> a
caput = foldr const undefined -- caput.0
```

Give a value of type *ProofLayout a* that captures the proof of

```
∀ x::a, xs ::  [a], caput (x:xs) == head (x:xs)
```

For full marks, each step must be annotated with the rule that justifies the step.

```
caputHead :: a -> [a] -> ProofLayout a
caputHead = undefined
```

2 (45 marks) Extended example

It is not necessary to know anything about the card game *Contract Bridge* (or just *Bridge*) for this question. Anything needed will be explained. Where you may be familiar with different rules to those described in the accompanying files, **the rules and tests in the accompanying files take precedence**.

(i) [2 marks] Define a *pack* of cards as a list of *'Card'*, containing each card exactly once. The suits must be arranged in the order: *Clubs*, *Diamonds*, *Hearts*, *Spades*; within each suit the cards must be arranged in the order *Two*, *Three*, ..., *Ten*, *Jack*, *Queen*, *King*, *Ace*.

Types *'Card'*, *'Suit'* and *'Value'* are defined in the imported module *'Pack'*, which you are recommended to read (but must **not** modify) as well as utility functions for printing structures containing cards.

Your solution should satisfy:

```
packTest :: Bool
packTest =
        pack!!3 == Card Clubs Five
        && pack!!4 == Card Clubs Six
        && pack!!16 == Card Diamonds Five
        && pack!!31 == Card Hearts Seven
        && pack!!48 == Card Spades Jack
```

To gain full marks your solution should take advantage of the fact that the types used to describe *'Card'* are instances of *'Enum'*.

In the next group of questions you will build a function that mimics the way that people prepare the game, by *shuffling* the pack and then *dealing* the pack.

(ii) [3 marks] A stream of pseudorandom natural numbers can be generated by the *linear congruential method*. Given any element of the stream, *'n'*, the next element is:

```
a*n+c `mod` m
```

for suitable values of *'a'*, *'c'*, and *'m'*. Picking these values can be difficult, but one reasonable assignment is *'*$a == 7^5$*'*, *'*$c == 0$*'*, and *'*$m == 2^{31} - 1$*'*.

Define *'psrn'* to compute the next element of the stream by the linear congruential method, using these values.

Your solution should satisfy:

```
psrnTest :: Bool
psrnTest = psrn 1234567890 == 395529916 && psrn 2468013579 == 1257580448
```

(iii) [5 marks] Implement a function, *'insertMod'*, to insert an element into a list at a given

position. If the given position is larger than the length of the list then insertion should be at the position found by taking the input modulo the number of insertion positions.

Your solution should satify:

```
insertModTest :: Bool
insertModTest =    insertMod    0 "hello" 'x' == "xhello"
 && insertMod    5 "hello" 'x' == "hellox"
 && insertMod    3 [0..4]  9   == [0, 1, 2, 9, 3, 4]
 && insertMod   24 "hello" 'x' == "xhello"
 && insertMod  131 "hello" 'x' == "hellox"
 && insertMod 1011 [0..4]  9   == [0, 1 ,2, 9, 3 ,4]
```

(iv) [6 marks] Implement a function, *'shuffleStep'* that calculates a single step of a shuffle (the next question asks for the whole shuffle). The function *'shuffleStep'* takes a function over numbers, a number–list pair, and an element. The result is also a number-list pair. The number in the output pair is given by applying the first argument to the input number. The list in the output pair is obtained by inserting the potential element into the list at the position given by the number in the input pair, following the same rules that *'insertMod'* follows.

Your solution should satisfy:

```
shuffleStepTest :: Bool
shuffleStepTest =
 shuffleStep id   (         3, "hello")  'x' == (         3, "helxlo")
 && shuffleStep psrn (1234567890, [0 .. 4]) 9   == ( 395529916, [9,0,1,2,3,4])
 && shuffleStep psrn (2468013579, [0 .. 4]) 9   == (1257580448, [0,1,2,9,3,4])
```

(v) [10 marks] We can get the effect of randomising, or *shuffling*, of a list, by repeatedly taking an element of a list and inserting it into a second, initially empty, list at a random position. To obtain the stream of random numbers, a *seed*, or starting point, and a *next–number* function are parameters of the function. Implement the *'shuffle'* function.

Your solution should satisfy:

```
shuffleTest :: Bool
shuffleTest =
 shuffle psrn 1234567890 [0 .. 9] == [9,4,1,0,3,7,6,5,8,2]
 && shuffle psrn 2468013579 [0 .. 9] == [2,4,1,3,5,8,9,6,0,7]
 && shuffle id            0 [0 .. 9] == [9,8,7,6,5,4,3,2,1,0]
 && shuffle (+1)          0 [0 .. 9] == [0,1,2,3,4,5,6,7,8,9]
```

(vi) [7 marks] Implement a function, *'deal'*, to distribute a list as evenly as possible into four ordered sublists.

Your function should satisfy:

```
dealTest :: Bool
```

```
dealTest =
 deal [0..9]          == ([0,4,8],[1,5,9],[2,6],[3,7])
 && deal "hello world!" == ("hor"," el","dlw","!lo")
```

You may use the function *'insertOrd'* in your solutions:

```
insertOrd :: Ord a => a -> [a] -> [a]
insertOrd w = foldr f [w]
  where
    f x ys@(z:zs) | x > w     = z:x:zs
       | otherwise = x:ys

deal :: Ord a => [a] -> ([a], [a], [a], [a])
deal = undefined
```

Using your function *'deal'*, implement a function, *'shuffleDeal'* that, given a seed, and a next number function, shuffles and deals a full pack of cards for a game of bridge.

When pretty printed using the function *'pp'* defined in module *'Pack'* available in *Pack.hs*, your function should print as shown in *Q2vi.hs*. **Note:** the pretty print *'pp'* output on Windows OS may use different fonts.

(vii) [12 marks] A game of bridge is played either with

* one suit being designated the *trump* suit, or
* no suit being designated the trump suit.

Give a suitable data type to represent the current trump suit, or that there is no trump suit. Define the special value *'noTrumpSuit::Trumps'* which represents there being no trump suit, and the function *'theTrumpSuit :: Suit -> Trumps'*, so that *'theTrumpSuit s'* is a value that represents *'s'* being the trump suit.

```
noTrumpSuit :: Trumps
theTrumpSuit :: Suit -> Trumps

type Trumps = ()
-- TREAT AS UNDEFINED TYPE;
you may replace `type` by `newtype` or `data`

noTrumpSuit = undefined
theTrumpSuit = undefined
```

In each round one player leads, and then the other players follow the lead, in rotation. The following rules are used to decide the winner of the round, or *trick*:

* if there is no trump suit, or there is a trump suit but no cards of that suit have been exposed, then the highest card of the same suit as the led card (the first card exposed)

wins.

* if there is a trump suit, and cards of that suit have been exposed, then the highest card of the trump suit wins.

* The type *'Card'* is declared as deriving *'Ord'*, in a way such that the maximum card in a list matches the Bridge notion of highest card in a trick.

Implement a function, *'trickWinner'*, that takes:

* a trump suit, or no trumps,

* a led card, and

* a list of three following cards,

and determines the winning card of the four.

Your solution should satisfy

```
trickWinnerTest :: Bool
trickWinnerTest =    twh noTrumpSuit             == Card Spades Ace
                  && twh (theTrumpSuit Diamonds) == Card Spades Ace
                  && twh (theTrumpSuit Hearts)   == Card Hearts Four
  where
    twh t = trickWinner t (Card Spades Two)
    [Card Hearts Four, Card Spades Ace, Card Spades King]
```

End of examination paper