CS3518 Haskell In Course Assessment, 2017-2018

What you should submit: You should submit a **single lastname.hs file (where lastname is your last name all in lower case)** that can be loaded and run unchanged into GHCi. Any information that is not Haskell code should be included in Haskell comments. The file should be a simple ASCII file (not a Word document). **Submissions in any other form (i.e. folders, text files, or incorrect filenames) will be rejected**.

How you should submit: Submit your assignment via MyAberdeen for the course by the deadline.

When you should submit: The deadline is Wednesday, April 4 by 17:00. Only one attempt is allowed.

Penalties: Work submitted up to one day late attracts a penalty of 10%, up to one week late, 25%; work handed in more than one week late is marked and returned but is counted as a 'no paper'.

Note: You are meant to do your coursework by yourself: collusion will lead to points being deducted.

General information: See the specific problems below. For each problem, specify the type of each function. Try to make your program as elegant, readable and concise as possible. There are many possible answers which produce the correct behaviour. Any such answer will get a good mark, but credit will be given for functions that are general, succinct, and accompanied by good comments.

For each of the functions that you define, please show a few test runs of your function. It ought to function on the example inputs as well as a couple of your own additional sample inputs. It is possible that the function you have coded does not work properly for all possible (i.e., all type-correct) inputs. In such cases, it is important that you choose the most widely applicable code and that you formulate the appropriate conditions on the arguments of the function (i.e., where if these conditions are not met, the function is undefined or gives an incorrect result). Stating the right preconditions for a function can constitute an important part of your mark.

To reinforce the connection with other parts of the course, a number of the functions below focus on formal languages, as you will see. You can use functions defined in Haskell's Prelude, and you can call any of the functions that you define elsewhere in your submission. Otherwise, you should not use functions defined by others.

SEE NEXT PAGE

Specific problems (answer all of them):

1. Write a function *inlist* which, given a list of Integers and an Integer n, returns a Boolean indicating whether n occurs in the list. [max 11 points] For example,

   Main> inlist [2,3,2,4,7,9] 7
   True
   Main> inlist [2..100] 101
   False
   Main> inlist [2..] 101
   True

2. Write a function *exactlyonce* which takes a list of Integers and an Integer n, then returns a Boolean indicating whether n occurs *exactly once* in the list. Can you give a function that produces a result which includes the last example? If not, why not? Explain in a couple of sentences. [max 11 points] For example,

   Main> exactlyonce [2,3,2,4,3] 3
   False
   Main> exactlyonce [3..30] 15
   True
   Main> exactlyonce [1..] 15
   ?

3. Define a function *equalones* which takes two lists of integers and decides whether it's true that each of the lists contain equal numbers of ones. [max 11 points] For example,

   Main equalones [1,2,0] [3,5,1,1]
   False
   Main equalones [1,0] [0]
   False
   Main equalones [1,0,0,1] [0,1,1,0]
   True

4. Write a function *replacenew* which takes an integer x and a list of integers. It returns a list where each element y of the list equals (x-y)*(x-y). [max 11 points] For example,

   Main> replacenew 2 [3,6,9]
   [1,16,49]

SEE NEXT PAGE

5. Define a function *addthemup* that takes a list of lists of integers, sums the numbers in each of the lists within the list, then multiplies the resulting sums with each other. Hint - you will need to review and use the function *foldr*. [max 11 points] For example,

   ```
   Main addthemup [[1,3],[3,7]]
   40
   Main addthemup [[1,2,3],[9]]
   54
   Main addthemup [[1,2],[]]
   0
   Main> addthemup [[1,2],[1,3],[4,5,7],[2]]
   384
   ```

6. Define a function *repeatnew* that repeats the application of a function to an argument a given number of times. [max 15 points] That is,

   repeatnew f 1 2 = f 2,
   repeatnew f 2 2 = f (f 2),
   repeatnew f 3 2 = f (f (f 2)), and so on.

   For example, where *square y = y*y*,

   ```
   Main> repeatnew square 1 2 = 4
   Main> repeatnew square 2 2 = 16
   Main> repeatnew square 4 1 = 1
   Main> repeatnew square 10 0.999 = 0.35897928
   ```

   (For working correctly on integers, your function *repeatnew* gets a maximum of 8 points. If it works on fractions (as in the last two examples) too, you get a maximum of 12 points. 3 additional points are awarded for giving *repeatnew* its most general polymorphic type.)

7. Define a function *antepenultimate1* such that, if *xs* is a list of integers then *antepenultimate1 xs* is *True* iff the antepenultimate member of *xs* is *1*. [max 15 points] For example,

   ```
   Main> f2 []
   False
   Main> f2 [1,0,1,1,0]
   True
   Main> f2 [1,0,0,1,1]
   False
   Main> f2 [1,0,1,0,1,0]
   False
   ```

SEE NEXT PAGE

8. Define a function *sequenceones* such that *sequenceones xs = True* if and only if *xs* contains the substring *11*. [max 15 points] For example,

```
Main> sequenceones []
False
Main> sequenceones [1,0,1,1,0]
True
Main> sequenceones [1,0,0,1,0,1]
False
Main> sequenceones [1,0,1,0,1]
False
```

END OF ASSIGNMENT