CSCE314: Programming Languages

Class Test 1/Midterm

Mar 10/11, 2016

Name: _____

UIN: _____

Alias: _____

Section: _____ (501 Shell/500Keyser)

Total time: 50 minutes

Total marks: 100, where 100 = 100%

Total total pages: 8

Question 1 (Compound Interest)	10
Question 2 (Rock-Paper-Scissors)	20
Question 3 (Computing Averages)	15
Question 4 (Parsing)	10
Question 5 (List Compression)	15
Question 6 (Types)	15
Question 7 (Short Questions)	15
Total	100

Plan and use your time wisely!

I have read and understand the Aggie Honor Code:

"An Aggie does not lie, cheat or steal or tolerate those who do."

Signature: _____

Question 1: Compound Interest (10 points)

Alice is thinking about investing some money in an annuity. The package she is considering has a fixed interest rate that is compounded annually. Write a function to allow her to see the growth of her funds by producing a list of values showing the balance over a series of years. For example, if she invests $100.00 at 10% rate for 5 years, then she should evaluate the function as follows (truncating at the sixth entry, because the first entry represents her base investment): [10 points]

```
Main> take 6 (compoundReturn 100.00 10)
[100.0,110.0,121.0,133.1,146.41,161.051]
```

Question 2: Rock-Paper-Scissors (20 points)

In the game rock-paper-scissors, each player picks one of the three options. If one player chooses rock and the other paper, paper will win. If one chooses paper and the other scissors, scissors will win. If one chooses scissors, and the other rock, then rock will win. If both players pick the same option, it is a tie.

2.1 Create a data type that can be used for this game. [4 points]

2.2 Using the type from part (2.1), write the type description for a function, `rps`, that will take two players' moves and return either 1 if player 1 wins, 2 if player 2 wins, or 0 if it's a tie. [4 points]

2.3 The function `rps` can be implemented in several ways, including with an if conditions, using guards, and using pattern matching. Pick one of these three ways and write `rps`. [6 points]

2.4 Write `rps` in a second way, but using a different technique (e.g., using if conditions, guards, pattern matching) from your answer in (2.3). [6 points]

Question 3: Computing Averages (15 points)

To compute a weighted average of several values, you assign each value v_i a weight w_i. The weighted average is the average, where each value is added in proportion to its weight:

$$\frac{\Sigma w_i v_i}{\Sigma w_i}.$$

Write code for the function `weightAvg`, that takes in a list of weights and a list of values, and computes a weighted average of the values.
You are welcome to write helper functions as needed.

Question 4: Parsing (10 points)

This question asks you to define a parser for strings. You may use the standard parsing functions (e.g., those in Hutton, Ch. 8, or those in `Parsing.hs`). You can also assume that all the character processing operations (as provided in Hutton, A. 3) are available to you, as if you had `import Data.Char` at the beginning of your program.

Write a parser that recognizes strings $s_1 s_2 s_3, \ldots$ where s_1 is the uppercase character of s_2. That is to say, strings starting with a capital letter, followed by that same letter in lowercase form.

Here are some examples of how your parser, `myParser`, should behave on various inputs:

```
Main> parse myParser "aA"
[]
Main> parse myParser "Bbbc"
[("Bb","bc")]
Main> parse myParser "Abbc"
[]
```

Question 5: List Compression (15 points)

Run-length encoding, or RLE, is a common form of compression for data that are repetitious. Write a function, `myRLE`, which takes as an input a list, and compresses it as follows:

```
Main> myRLE [1,1,1,2,3,1,4,4,5,5,5,1,2]
[(3,1),(1,2),(1,3),(1,1),(2,4),(3,5),(1,1),(1,2)]
Main> myRLE "hhhelllllllo worrrrld"
[(3,'h'),(1,'e'),(7,'l'),(1,'o'),(1,' '),(1,'w'),(1,'o'),(4,'r'),(1,'l'),(1,'d')]
```

You might find it useful to define a helper function.

Question 6: Types (3×5 points)

6.1 What would Haskell return as the type of this function: needs to needs to needs to needs to needs to

```
palindrome xs = reverse xs == xs
```

That is, what would the command `:type palindrome` return? Don't forget any class constraints that might be necessary!

6.2 What is the type of the following construct?

```
fls = [take 12]
```

That is, what would the command `:type fls` return?

6.3 What is the type of the following function?

```
tripler f x = f( f ( f x ))
```

What would the command `:type tripler` return?

Question 7: Short Questions (3×5 points)

7.1 Consider the following function:

```
myLister:: [[x]] -> [x]
myLister ls = [x | xs <- ls, (length xs `mod` 2 == 0), x <- xs]
```

What does it return on the following input:

```
Main> myLister ["This","is","a","longish","sentence"]
```

7.2 What does the following function do?

```
functionr :: (Enum t, Num t) => t -> [a] -> [[a]]
functionr x ls = [ls++ls | _ <- [1..x]]
```

Describe what it does and give two examples of its output for different inputs.

7.3 There are errors in this pair of functions for determining whether a non-negative natural number is even or odd. Find the bugs and fix them.

```
isOdd 1 = True
isOdd n = isEven n-1

isEven 0 = True
isEven m = isOdd m-1
```