Assignment 1

CISC 260, Winter 2018

Topics: "Introduction To Haskell" and Recursion

Due Date: 9 a.m. on Friday, Jan 26.
For this assignment and every other assignment in CISC 260, I will program a 24-hour "grace period" into OnQ. This means you can submit your assignment up to 24 hours late if you need to, but we will deduct 10% from your mark. After the grade period is over we will not accept any more submissions. If there are special circumstances such as illness that prevent you from completing the assignment on time please inform the instructor via e-mail.

This first Haskell assignment requires you to use recursion for some of the problems, but not lists or strings. In fact, **you are *not allowed* to use lists or strings** in this assignment except for the strings used in error messages. Solutions that use lists or strings (except for error message strings) will get zero points, even if they work.

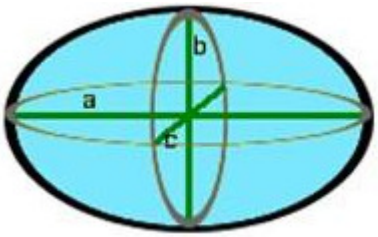
Some advice and notes for this assignment and every other one in this class:

- Before you tackle an assignment, do the assigned readings for the required topic(s). If you had to miss one of the lectures about a topic, watch the video from OnQ. It's also a good idea to do the practice problems first, since they will generally be simpler than the assignments and be a good way to prepare.
- Don't wait until the last minute to work on an assignment. If you start early you won't be under so much pressure and there will be a better chance of getting help if you need it.
- You may send quick assignment questions to the instructor via e-mail – for example, questions to clarify exactly what is required. But please don't send mail with questions asking for long explanations, or asking me to help debug your program over e-mail. This is a large class and I simply don't have time to write long essays to lots of individual students. A better way to get help is to visit one of my office hours or talk with me after class, so we can have a real conversation.
- The TAs for this course are not available to answer questions via e-mail. However, I hope to schedule a TA help session for each assignment shortly before its due date, where you can drop in to get help as you're finishing the assignment. These will be announced on OnQ.

Marking Scheme: There are four problems in this assignment, each worth 5 points.

You must provide a type declaration for every function in your solution. This includes the required functions and also helper functions if you choose to create any. Haskell doesn't require type declarations, but using them is a huge help while debugging and I want you to get into the habit from the start. One point of the mark for each of the functions in this assignment is for the type declaration; the remaining 4 points are for correctness.

Problem 1: An "ellipsoid" is a three-dimensional shape that can roughly be thought of as a sphere that may have been "squashed" or elongated along one or more of three perpendicular axes. Here's a diagram to illustrate:



This diagram was copied from http://www.web-formulas.com/Math_Formulas/Geometry_Volume_of_Ellipsoid.aspx and you may refer to this web site if you would like more information about ellipsoids.

If the lengths of all three of the axes are equal, the ellipsoid is called a sphere. For the purposes of this assignment, let's define a "football" as an ellipsoid in which exactly two of the axis are equal.

Write a function called `shape` that takes three parameters, the lengths of the three axis of an ellipsoid, and returns a character describing the kind of ellipsoid it is: 'S' for a sphere, 'F' for a football, 'E' for an ellipsoid which is not a sphere or football, and 'X' if any of the parameters are equal to or less than zero. The parameters for this function should all be `Doubles` and the result should be a `Char`.

Example Run Using My Solution:

```
*Assignment1> shape 3 3 3
'S'
*Assignment1> shape 7.3 7.3 7.3
'S'
*Assignment1> shape 2 5.1 5.1
'F'
*Assignment1> shape 9 9 3
'F'
*Assignment1> shape 3 4 3
'F'
*Assignment1> shape 7 2 4
'E'
*Assignment1> shape 3 5 9
'E'
*Assignment1> shape 0 3 4
'X'
*Assignment1> shape 5 (-2) 3
'X'
*Assignment1> shape 7 6 (-5)
'X'
```

Note (referring to the last two test cases): In Haskell it is often necessary to put parenthesis around negative numbers. If you write `shape 5 -2 3`*, Haskell will interpret this as* `shape (5-2) 3`*, which is not what you probably meant at all!*

Problem 2: If the lengths of the three axes of an ellipsoid are a, b, and c (as in the diagram for Problem 1), the volume of the ellipsoid is 4/3 * π * a * b * c. In the case of a sphere, where a and b and c are all equal to the same value r, this leads to the familiar formula 4/3 π r^3 for the volume of a sphere.

Write a Haskell function called `volume` which takes three parameters of type `Double` (the lengths of the three axes) and returns a result of type `Double` (the volume of an ellipsoid with those axes).

If any of the axes are negative, your function should abort with an error message instead of returning a result. If one or more of the axes are zero (but none are negative), your function should just return zero.

Note: There is a constant called `pi` built into Haskell. Please use it instead of typing a value of your own like 3.14 or 3.14159. The answers we test for will assume you're using Haskell's value for pi.

Example Run With My Solution:

```
*Assignment1> volume 1.1 2.2 3.3
33.45167857542412
*Assignment1> volume 8 7 7
1642.0057602762652
*Assignment1> volume 6 0 3
0.0
*Assignment1> volume (-1) 9 4
*** Exception: ellipsoid with negative side(s)
CallStack (from HasCallStack):
  error, called at assn1Solution.hs:18:33 in main:Assignment1
```

Problem 3: Write a function called `logSum` that takes two `Int` parameters a and b. Its value must be the sums of the natural logarithms of all the numbers between a and b inclusive. In other words, the value must be log a + log (a+1) + log (a+2) + log (a+3) + … + log (b-1) + log b. If a == b, the value should be log a. If a > b, the value should be zero (since there are no integers between a and b in this case). The result of `logSum` should be a `Double`.

To compute natural logarithms, use the Haskell `log` function.

In a programming language like Java or Python, you'd use a loop to add up all of these logs, but you can't do that in Haskell. You'll have to use recursion to get the same effect.

An additional challenge in Haskell is that it is very precise about types. This often is useful and prevents certain kinds of errors, but this is one of the times when it can be a bit of a pain. The Haskell `log` function expects its parameter to be a `Double`, and you'll be getting `Int`s as parameters to your functions. If you pass an `Int` to the Haskell `log` function you'll get a type error. The solution is to use the `fromIntegral` function to convert an `Int` to a `Double`. We'll be talking about the details of the Haskell type system later; for now, just take my word for it that `fromIntegral` will help you out here.

Example Run With My Solution:

```
*Assignment1> logSum 1 4
3.1780538303479458
*Assignment1> logSum 5 20
39.15756263040554
*Assignment1> logSum 3 3
1.0986122886681098
*Assignment1> logSum 5 4
0.0
```

Problem 4:
Astronauts have discovered a strange new kind of life form that grows in isolated spots on some other planets in the solar system. They've named these beings "ET"s and have brought some back to earth to study. Scientists have concluded that ETs have peculiar grown patterns that seem to obey the following rules in Earth gravity:

- If the mass of an ET is less than 1 gram its mass will double in one day. For example, if an ET weighs 0.75 grams today it will weight 1.5 grams tomorrow.
- If the mass of an ET is at least 1 gram but less than 20 grams its mass after one day will be 1.5 times the starting mass plus 2 grams. For example, an ET that weighs 2 grams today will weigh 5 grams tomorrow (1.5 *2 + 2)
- If the mass of an ET is at least 20 grams but less than 100 grams its mass after one day will be 1.2 times its starting mass plus 1 gram. For example, an ET that weighs 50 grams today will weigh 61 grams tomorrow (50*1.2 + 1)
- If the mass of an ET is 100 grams or more its mass will be 1.1 times its starting mass plus 0.5 grams. For example, an ET that weighs 200 grams today will weigh 220.5 grams tomorrow (200*1.1 + 0.5).

The scientists aren't very good at programming and have asked you to provide them with a Haskell function that will predict the growth of an ET in a given number of days. This function should be called `growET` and should take two parameters (in the order given):

1. The initial mass of the ET in grams (a Double)
2. The number of days the ET will be allowed to grow (an Int)

It should return the predicted mass of the ET in grams (a Double) at the end of the growth period.

The function should abort with a descriptive error message instead of calculating a result if any of the following conditions are true:

- The initial mass is less than zero
- The initial mass is non-negative but the number of days is less than 0. (Zero days is not an error; the ET just doesn't grow at all.)
- The initial mass is non-negative but the number of days is more than 100. (This computation would not be useful because no ET has been observed to survive for more than 100 days.)

By "descriptive error message", I mean an error message that tells the user what the problem is. For example, "initial mass less than zero" is descriptive. "ERROR" or "Error #3" are not descriptive.

Important Restriction: You must solve this problem using recursion. Don't waste time looking for a way to do it with exponentiation or logarithm functions.

Problem 4 example run on the next page….

Example Run With My Solution:

```
*Assignment1> growET 5 1
9.5
*Assignment1> growET 17 1
27.5
*Assignment1> growET 55 1
67.0
*Assignment1> growET 102 1
112.7
*Assignment1> growET 3 0
3.0
*Assignment1> growET 0.4 6
24.35
*Assignment1> growET 5 8
73.07104
*Assignment1> growET 82 4
146.5888
*Assignment1> growET 0.2 100
381629.6082808138
```

A Note About Accuracy: Digital computers use a finite number of digits to represent floating numbers, so floating-point arithmetic is not 100% accurate and solutions for problems 3 and 4 may accumulate small amounts of error. It's possible that the answers you get on your computer will be slightly different from the answers I've mentioned as examples, since floating point arithmetic may not be exactly the same for every make and model of computer. If you're getting answers that differ from the ones I've mentioned by less than .001, don't worry about this. We'll take this possibility into account while marking. In the same way, if you check your answers using a calculator (a physical one or a calculator program on your computer) the same sort of small differences may occur.

Administrative Stuff (applying to all 4 questions): Please read and follow the instructions carefully. We reserve the right to deduct an administrative penalty if you deviate from the instructions, even if you basically got the right answer. For example, if your solution for Problem 1 returns lower-case letters 's', 'f', 'e' and 'x' instead of the requested upper-case letters 'S','F','E','X', or if you give a function a different name from what was specified in the instructions. This may sound very picky, but at the time of writing there are almost 200 students in this class and we will be using scripts to mark your assignments. If your function doesn't work as required our scripts won't work and it will cost us quite a lot of marking time.

Handing in the Assignment: This assignment must be handed in to the Assignment 1 Dropbox area on our OnQ site, which will be available at least a week before the due date. You must hand in one file, called `Assignment1.hs`. Again, with a large class it causes a lot of extra work if students try to hand in by other means such as e-mail, or give us a file that has to be renamed for our testing script to work.