Lab 04 Specification – Exploring Haskell
Due (via your git repo) no later than 8 a.m., Tuesday, 23rd October 2018.
50 points

Lab Goals

Most of today's lab is a tutorial on Haskell and aimed as an extension to the in-class activity done on 10/15/2018.

This is an individual lab assignment, so the student would need to work on the assignment individually and do the final submission on the student's git repository before the deadline.
This is a friendly reminder, that there is no lab session next week 10/23/2018 as such, in order to observe Gator Day!

Why Haskell?

I'm no expert in Haskell, so I suggest you read the first few paragraphs (if you don't have time to read the whole thing) of "Haskell, the Language Most Likely to Change the Way you Think About Programming" (here's a link: `http://goo.gl/moQQzO`). This does not necessarily represent my opinion (I don't really have enough experience with Haskell to have an opinion).

If you want a little more detail try reading "Why Haskell Matters" at `https://wiki.haskell.org/Why_Haskell_matters`.

Why Now?

We are in the middle of a chapter on subroutines, so technically we could use any language to examine the concepts on subroutines. But I think one of the things people look forward to in this course is learning new languages, and based on a few favorable reactions from previous labs, I thought it might be a good idea to continue on in the "learn new languages" mode for a while.

Where Do I Start?

Go to the online tutorial "Learn You a Haskell for Great Good!" at `http://learnyouahaskell.com/` and just start reading. Skip the parts you think you don't need, but try the examples.

I'd also suggest "Ten Things You Should Know About Haskell Syntax" at `https://goo.gl/LpBUB3` (but it's a little advanced, so you might want to skip it for now).

Here are a few things you might miss if you're not reading carefully:

- Some of the examples in the tutorial have a "`ghci`" prompt; others don't. The ones that don't were intended to be entered into a file and then "l"oaded using the "`:l`" command.
- When typing negative numbers like "-3" it's always a good idea to put parentheses around them: "(-3)".
- If you type "`x = 10`" into the `ghci` environment it won't work (you have to type "`let x = 10`"). But in a Haskell file the word "`let`" is not needed.

- The files that you upload with the "`:l`" command are not "scripts" in the usual sense of the word—they do not contain commands to be executed. Instead, they contain definitions of names and functions. You must still enter commands in the `ghci` environment to make use of these functions and definitions.
- Function arguments don't require parentheses. Thus, "`sqrt 10`" means "call `sqrt` with argument 10" and "`take 3 [1,2,3,4,5]`" takes the first three elements of the list `[1,2,3,4,5]`. (But watch out— if you want $\sqrt{\sqrt{10}}$ then you must write "`sqrt (sqrt 10)`"; you'll get an error with "`sqrt sqrt 10`".)
- You can convert a two-operand function into an infix operator using back-quotes (this is on the upper left key on your keyboard):

  `take 3 [1,2,3,4,5]` is the same as `3 ` `take` ` [1,2,3,4,5]`
- Comments in Haskell begin with two hyphens "`--`" and extend to the end of the line, like this:

```
-- Aravind Mohan
-- Haskell function definitions
```

We have `ghc` and `ghci` installed on the lab machines.

What Am I Supposed to Hand In?

As you work your way through the tutorial, I'd like you to answer some questions about Haskell. These all refer to topics that are mentioned in the book. Place these answers in a reflection document (don't forget to put your name and the Honor Code statement in your document; please number the answers).

Then I'll ask you to write some Haskell programs and place all of these into a second document. Again, be sure to fully comment this document.

Questions

1. Is Haskell a strongly-typed language? (You might want to review the definition and say a few words about how Haskell meets, or fails to meet, the definition.) If you have trouble, you may look for Web resources to help you answer, but be sure to cite them.
2. Haskell exhibits "referential transparency," an idea that was mentioned in our book back in chapter 6 (p. 225). Can you explain "referential transparency" in your own words or with an example? (If you can't, can you find—and cite—a good definition and/or example from the Web different from the one in the Haskell tutorial I've asked you to read.)
3. Haskell has multiple exponentiation operators. Look them up, explain their differences, and give examples.
4. Is there a clear operator precedence? Try to construct an operator precedence for at least the arithmetic and relational and boolean operators. Either find this somewhere (and cite it) or determine it through experimentation. (Remember about the parentheses around negative constants! What happens if you leave them out?)
5. How do lists in Haskell differ from lists in Python? (This is vague, but there should be at least one very clear difference.)
6. Given your very limited experience with Haskell in the tutorial, what aspects of Haskell (so far) do you find . . .
   - the most interesting or unusual?
   - the hardest to understand?
   - the most frustrating?

- the most fun?

7. Create three Haskell functions. Feel free to be whimsical and to have fun. (These functions do not have to have "real-world" applications.) Indicate in your comments how the functions are to be used and provide examples of their usage. I will test them based upon your description of how to use them. Make sure that these Haskell functions are different from that of the in-class activity.

   I understand the temptation to look up a Haskell function on the Web and use that; please feel free to experiment with such functions, but in the end produce something that is original to you. If you rely heavily on an example found online, *be sure to cite the source* in your comments!

   (a) At least one of them should manipulate a list in a nontrivial and interesting way (you have to justify the terms "nontrivial and interesting" in your header comments).

   (b) At least one of them should do something interesting with strings. Remember, a string is (internally) just a list of characters; try to use at least two of the functions "`head`", "`tail`", "`init`", and "`last`" in your function. Justify "interesting"!

   (c) The last one should perform a nontrivial and interesting calculation on one or more numeric arguments. If possible, use conditionals ("`if`" boolean operators, etc.) to create several cases. Explain what your function does. And what makes it nontrivial and interesting.

Required Deliverables

Please submit electronic versions of the following deliverables to your GitHub repository by the due date:

1. Save your functions in a single file, comment them, and upload the file to your repository.
2. A reflection document in PDF format, with your answers to the textual questions.

Grading Rubric

1. If you complete [Task 1 - Task 6] completely as per the requirement outlined above, you will receive 30 points (5 points each).
2. If you complete Task 7 completely as per the requirement outlined above, you will receive 20 points. The sub section 7(a) and 7(b); are worth 5 points each and the sub section 7(c); is worth 10 points.
3. If you fail to upload the lab solution file to your git repo by the due date, there will be no points awarded for your submission towards this lab assignment. Late submissions will be accepted based on the late submission policy described in the course syllabus. It is the student's responsibility to communicate to the professor if it is is a late submission. If not communicated in advance about the late submission, the students lab work shall not be graded as such.
4. Partial credit will be awarded, based on the work demonstrated in the lab submission file.
5. If you needed any clarification on your lab grade, talk to the instructor. The lab grade may be changed if deemed appropriate.