Andreas Landgrebe
Computer Science 220
Lab 6: Haskell

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.

   Haskell is a strongly-typed language. This is the case because it is not possible for the programmer to work around the restrictions imposed by the type system. An example of a weak-typed language is C. This is the case because any pointer type is convertible to any other pointer type by casting.
   The following information was found on this website.

   `http://stackoverflow.com/questions/2690544/what-is-the-difference-between-a-strongly-typed-language-and-a-statically-typed`

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 cant, can you findand citea good definition and/or example from the Web different from the one in the Haskell tutorial Ive asked you to read.)

   The following definition was found on this website.
   `https://wiki.haskell.org/Referential_transparency`
   Referential transparency is a is a property that makes it easier to reason about the behavior of programs. This usually means that an expression can evaluate to the same result in any context.

3. Haskell has multiple exponentiation operators. Look them up, explain their differences, and give examples
   The following explanation was found on the following website:
   `http://stackoverflow.com/questions/6400568/exponentiation-in-haskell`

```
There are three exponentiation operators.
^, ^^ and **.
 ^ is non-negative integral exponentiation,
 ^^ is integer exponentiation,
and ** is floating-point exponentiation.
Below are examples:
```

```
(^)  :: (Num a, Integral b) => a -> b -> a
(^^) :: (Fractional a, Integral b) => a -> b -> a
(**) :: Floating a => a -> a -> a
```

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?)
   The following information was found through this website. `https://www.haskell.org/onlinereport/decls.html`

module	-> `module` modid [exports] `where` body	
	| body	
body	-> { impdecls ; topdecls }	
	| { impdecls }	
	| { topdecls }	
topdecls	-> topdecl_1 ; ... ; topdecl_n	(n>=1)
topdecl	-> `type` simpletype = type	
	| `data` [context =>] simpletype = constrs [deriving]	
	| `newtype` [context =>] simpletype = newconstr [deriving]	
	| `class` [scontext =>] tycls tyvar [`where` cdecls]	
	| `instance` [scontext =>] qtycls inst [`where` idecls]	
	| `default` (type_1 , ... , type_n)	(n>=0)
	| decl	
decls	-> { decl_1 ; ... ; decl_n }	(n>=0)
decl	-> gendecl	
	| (funlhs | pat^0) rhs	
cdecls	-> { cdecl_1 ; ... ; cdecl_n }	(n>=0)
cdecl	-> gendecl	
	| (funlhs | var) rhs	
idecls	-> { idecl_1 ; ... ; idecl_n }	(n>=0)
idecl	-> (funlhs | var) rhs	
	|	(empty)
gendecl	-> vars :: [context =>] type	(type signature)
	| fixity [integer] ops	(fixity declaration)
	|	(empty declaration)
ops	-> op_1 , ... , op_n	(n>=1)
vars	-> var_1 , ... , var_n	(n>=1)
fixity	-> `infixl` | `infixr` | `infix`	

5. How do lists in Haskell differ from lists in Python? (This is vague, but there should be at least one very clear difference.)
   Lists in Haskell differ from lists in Python due to the fact that Python's list comprehension is taken directly from Haskell. The following information was found on this website. `https://wiki.python.org/moin/PythonVsHaskell`

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 most interesting aspect of Haskell being able to import specific list to the graphical user interface in the terminal.

   - the hardest to understand?
     The hardest to understand is being able to understand the syntax of Haskell.

   - the most frustrating?
     The most frustrating is realize that tabs and indexes make a substantial difference just like in Python. I didn't realize this until later on.

   - the most fun?
     The most fun is realizing how less verbose this language compared to Java.