HOMEWORK 0
CS 4450/7450

Due: Wed., 9/17/14, 11:59pm.

Introduction

During the course of this semester, we'll be learning the functional programming language Haskell as the vehicle for a broader study in the theory of programming languages. In this assignment, we'll start by setting up Haskell and then writing some simple functions.

1. Download and install the latest version of the Haskell platform from here:

   `http://www.haskell.org/platform`

2. The book mentions (in section 1.1, "Sessions and scripts") a terminal for evaluating Haskell fragments. The interpreter we'll use for this is `ghci`. On page 2, the book has the following:

   ```
   ? 6 × 7
   42
   ```

   After we've installed the Haskell platform, we can replicate this experience by running `ghci`:

   ```
   $ ghci
   GHCi, version 7.8.3: http://www.haskell.org/ghc/  :? for help
   Loading package ghc-prim ... linking ... done.
   Loading package integer-gmp ... linking ... done.
   Loading package base ... linking ... done.
   Prelude> 6 * 7
   42
   ```

   Notice that the Haskell syntax given in the book will differ slightly from the actual characters you'd type in. Try running `ghci` and playing around with the interactive interpreter to verify your Haskell installation appears to be working.

3. But we don't want to always type our programs out inside the interpreter before running them. We can also save a program to a file and then run it with the interpreter (or compile it into an executable using `ghc`). For example, on page 2, the book gives definitions for the functions `square` and `smaller`:

   ```
   square :: Integer → Integer
   square x = x × x

   smaller :: (Integer, Integer) → Integer
   smaller (x, y) = if x ≤ y then x else y
   ```

   We can implement these definitions in Haskell by using a text editor to create a new file (named `hw0_pawprint.hs`, where `pawprint` is your MU username):

   File : hw0_pawprint.hs

   ```
   square :: Integer -> Integer
   square x = x * x

   smaller :: (Integer, Integer) -> Integer
   smaller (x, y) = if x <= y then x else y
   ```

   Now we can load these functions into our interactive `ghci` session and try them out:

```
Prelude> :load hw0_pawprint.hs
[1 of 1] Compiling Main             ( hw0_pawprint.hs, interpreted )
Ok, modules loaded: Main.
*Main> square 14198724
201578206334976
*Main> square (smaller (5, 3 + 4))
25
```

Two other interpreter commands you'll find useful are `:type` (to calculate the type of an expression) and `:quit`:

```
*Main> :type square
square :: Integer -> Integer
*Main> :quit
Leaving GHCi.
```

You can get a complete list of the available commands with `:help`.

Problems

1. Define a function `nextlet` that takes a letter of the alphabet and returns the letter coming immediately after it. Assume that the letter A comes after Z. If the argument is not a letter, return the argument unchanged. For example:

```
Main*> nextlet 'c'
'd'
Main*> nextlet 'X'
'Y'
Main*> nextlet '9'
'9'
```

2. Define a function `digitval` that converts a digit character (i.e., 0–9) to its corresponding numerical value (of type `Int`). If the argument is not a digit character, return `-1`. For example:

```
Main*> digitval '0'
0
Main*> digitval '8'
8
Main*> digitval 'a'
-1
```

3. Define a function `twine` with the following type:

```
twine :: (a -> b) -> (a -> c) -> a -> (b, c)
```

It accepts two functions as arguments and "twines" them together into a function that accepts a single argument and constructs a tuple from the result of applying the two functions to the argument. For example:

```
Main*> twine nextlet digitval 'a'
('b',-1)
Main*> twine nextlet (nextlet . nextlet) 'a'
('b','c')
```

4. Define the function `cond` such that:

$$\texttt{cond}\ p\ x\ y = \begin{cases} x & \text{if } p \text{ is } \texttt{True}, \\ y & \text{otherwise.} \end{cases}$$

5. Suppose that a date is represented by a tuple `(day, month, year)`, such that `day`, `month`, and `year` all have type `Int`. Define a function `age` that takes two dates, the first being the birthday of an individual and the second being the current date, and returns the age of the individual in whole years. For example,

```
Main*> age (13, 4, 1994) (9, 9, 2014)
20
Main*> age (13, 12, 1904) (9, 9, 2014)
109
```

Grading

What to submit (via Blackboard): a single file, named `hw0_pawprint.hs`, where `pawprint` is your MU username. The file should contain definitions for every function listed above in the Problems section (plus any other code needed for by your implementation). Furthermore, everyone should adhere to the following guidelines to get full credit:

I. **Your submission must successfully load and typecheck on babbage to get any points.** For example, executing:

```
$ ghci your_hw.hs
```

should not produce any errors. We won't attempt to grade assignments that fail to load.

II. Name all functions and data types exactly as they appear in the assignment. Grading will be partly automated, so incorrectly named functions are likely to be counted as undefined functions.

III. The code you submit must be your own. Exceptions: you may (of course) use the code we provide however you like, including examples from the slides and the book.

IV. **No late submissions—please start early!**

Correct `nextlet`:	8
Correct `digitval`:	8
Correct `twine`:	8
Correct `cond`:	8
Correct `age`:	8
Every function has a type annotation:	5
File named and submitted correctly:	5
Total	**50**